К. Ю. Поляков, Е. А. Еремин

ИНФОРМАТИКА

7 класс

Часть 2

Москва
БИНОМ. Лаборатория знаний
2017

УДК 004.9
ББК 32.97
П54

Поляков К. Ю.

П54 Информатика. 7 класс : в 2 ч. Ч. 2 / К. Ю. Поляков, Е. А. Еремин. — М. : БИНОМ. Лаборатория знаний, 2017. — 160 с. : ил.

ISBN 978-5-9963-3094-2 (Ч. 2)
ISBN 978-5-9963-3095-9

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

Рассмотрены вопросы устройства и управления компьютером, построение алгоритмов для исполнителей, технологии работы с числовой, текстовой, графической и мультимедийной информацией.

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

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

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

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

УДК 004.9
ББК 32.97

Учебное издание

Поляков Константин Юрьевич
Еремин Евгений Александрович

ИНФОРМАТИКА

7 класс
В 2 частях
Часть 2

Ведущий редактор *О. Полежаева*
Ведущие методисты *И. Сретенская, И. Хлобыстова*
Художник *Н. Новак.* Технический редактор *Е. Денюкова*
Корректор *Е. Клитина.* Компьютерная верстка: *В. Носенко*

Подписано в печать 27.02.17. Формат 70x100/16. Усл. печ. л. 13,0.
Тираж 3000 экз. Заказ № м4292.

ООО «БИНОМ. Лаборатория знаний»
127473, Москва, ул. Краснопролетарская, д. 16, стр. 1,
тел. (495) 181-5344, e-mail: binom@Lbz.ru
http://www.Lbz.ru, http://metodist.Lbz.ru

Отпечатано в филиале «Смоленский полиграфический комбинат»
ОАО «Издательство «Высшая школа». 214020, г. Смоленск, ул. Смольянинова, 1
Тел.: +7 (4812) 31-11-96. Факс: +7 (4812) 31-31-70
E-mail: spk@smolpk.ru http://www.smolpk.ru

ISBN 978-5-9963-3094-2 (Ч. 2)
ISBN 978-5-9963-3095-9

Условные обозначения

В учебнике есть основной материал (обязательный для изучения), и дополнительный (для углублённого курса). Материал для углублённого курса обозначен чёрными горизонтальными линиями, значком  в начале материала и значком  — в конце.

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

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

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

Значок  означает важное определение или утверждение.

Значок  служит для выделения дополнительного задания или разъяснения.

Значок  означает групповую работу.

Значок  выделяет межпредметные связи.

Задания повышенной сложности отмечены «звёздочкой» (*).

Глава 5
ОБРАБОТКА ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

§ 24
Растровый графический редактор

Ключевые слова:

- графический редактор
- панель инструментов
- палитра
- основной цвет
- фоновый цвет
- масштаб
- линия (отрезок)
- прямоугольник
- овал (эллипс)
- многоугольник
- контур
- заливка
- текстовые надписи

Вспомните:
- какие рисунки называются растровыми;
- что такое пиксель.

Для создания и редактирования рисунков разработаны специальные программы — **графические редакторы.**

Растровые графические редакторы служат для создания и редактирования растровых рисунков, т. е. рисунков, состоящих из отдельных пикселей. Сначала мы познакомимся с редактором **Paint** (для практических работ можно использовать и другие редакторы, например **KolourPaint**).

Окно графического редактора

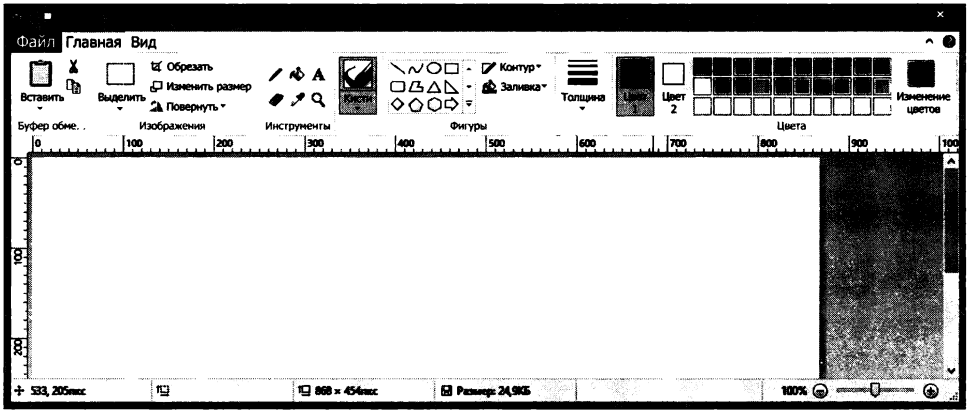


Рис. 5.1

В верхней части окна редактора *Paint* расположена *Лента* с тремя вкладками (рис. 5.1).

Выясните, что можно делать с помощью вкладки *Файл*.

На вкладке *Главная* расположены кнопки для работы с буфером обмена, инструменты для редактирования рисунков и цветовая палитра.

Вспомните, в каком случае полоса прокрутки становится активной (появляется движок в середине, полоса реагирует на управление мышью).

Вкладка *Вид* позволяет изменить масштаб просмотра (увеличить или уменьшить изображение на экране без изменения количества пикселей), вывести линейки (над рисунком и слева от него).

Снизу находится *строка состояния*: в ней программа показывает координаты указателя мыши (считая от верхнего левого угла рисунка), размеры рисунка в пикселях и объём файла. В правой части строки состояния с помощью движка можно изменять масштаб просмотра.

Рисование от руки

Простейший способ рисования — использование карандаша или кисти. Если включить инструмент *Карандаш*  , то при нажатой кнопке мыши можно рисовать линии на поле рисунка.

Меню *Толщина* (см. рис. 5.1) позволяет (для всех инструментов) выбрать толщину линии.

Цвет линии будет совпадать с цветом *Цвет 1* на панели *Главная*. Этот цвет ещё называют **основным** или **цветом переднего плана**. Если для рисования использовать не левую кнопку мыши, а правую, линия будет иметь *Цвет 2* **(фоновый цвет)**.

Для того чтобы изменить основной или фоновый цвет, нужно щелчком мышью выбрать *Цвет 1* или *Цвет 2*, а затем щёлкнуть на нужном цвете в палитре. Кнопка *Изменение цветов* позволяет добавить новые цвета в нижний ряд палитры.

Рисовать можно не только карандашом, но и различными кистями (меню *Кисти*), которые имитируют кисти художника и распылитель краски.

Чтобы стереть какие-то части рисунка, используют инструмент *Ластик*  . При нажатой левой кнопке мыши *Ластик* стирает (закрашивает цветом фона) все пиксели, через которые он проходит.

Исследуйте, как работает *Ластик* при нажатой правой кнопке мыши. 

С помощью инструмента *Заливка*  можно залить области одного цвета основным цветом (щелчком левой кнопкой мыши) или фоновым цветом (щелчком правой кнопкой мыши). Если контур, ограничивающий нужную область, имеет хотя бы небольшой разрыв, краска «вытекает» и может залить ту часть, которую закрашивать не нужно. В таких случаях можно отменить последние выполненные операции с помощью комбинации клавиш *Ctrl+Z* или кнопки  .

Инструмент *Масштаб*  позволяет увеличить нужную часть рисунка. Выполнив увеличение несколько раз, вы увидите, что изображение состоит из пикселей.

Выясните, какой наибольший масштаб можно установить в вашем графическом редакторе. 

Инструмент *Текст* A позволяет добавить надписи на рисунок. Выбрав этот инструмент, нужно щёлкнуть на поле рисунка, и в этой точке появится рамка для ввода текста. Размеры рамки можно изменять, перетаскивая маркеры в углах и на серединах сторон рамки. Саму рамку можно перетаскивать мышью за её границу.

На *Ленте* открывается дополнительная вкладка *Текст*, на которой можно выбрать нужный шрифт и установить прозрачный или непрозрачный фон (рис. 5.2).

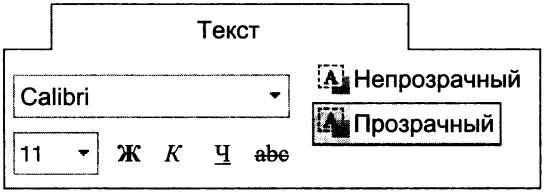


Рис. 5.2

Внутри рамки можно вводить и редактировать текст, как в обычном текстовом редакторе. Но как только вы щёлкнете мышью вне рамки, текст превратится в набор пикселей. После этого его можно редактировать только как растровый рисунок. Например, будет очень непросто заменить один символ на другой или удалить ненужный символ, особенно если надпись нанесена на сложный фон.

Геометрические фигуры

Рисовать всё от руки — весьма непростое занятие, ровную линию получить практически невозможно. Поэтому в любом графическом редакторе есть инструменты для рисования геометрических фигур — линий, прямоугольников, окружностей.

Чтобы нарисовать отрезок, нужно включить инструмент *Линия* \, нажать кнопку мыши в начальной точке отрезка, переместить мышь (при нажатой кнопке) в конечную точку и там отпустить кнопку.

Исследуйте, каков будет цвет линии, если рисовать её левой кнопкой мыши. Что изменится, если рисовать линию правой кнопкой мыши?

Выясните, что изменится, если при рисовании линии удерживать нажатой клавишу *Shift*.

Инструмент *Прямоугольник* служит для рисования прямоугольников и квадратов. Чтобы нарисовать прямоугольник, нужно нажать кнопку мыши в одном из углов будущего прямоугольника, провести мышь в противоположный угол и там отпустить нажатую кнопку. Размеры прямоугольника выводятся в строке состояния (до того, как вы отпустите кнопку мыши и зафиксируете прямоугольник на рисунке).

Проверьте, что будет, если рисовать прямоугольник при нажатой клавише *Shift*.

Прямоугольник — это замкнутая фигура, поэтому кроме контура для неё можно использовать заливку внутренней области. При работе левой кнопкой мыши контур прямоугольника будет основного цвета, а заливка — фонового цвета; при работе правой кнопкой мыши цвета меняются местами.

С помощью выпадающих меню Контур и Заливка можно выбрать нужные свойства контура и заливки, а также отключить контур или заливку вообще.

С помощью инструмента *Овал* (во многих программах он называется *Эллипс*) рисуют овалы и окружности. Овал начинают рисовать не от центра, а от угла прямоугольника, в который он вписан, т. е. от одной из пронумерованных точек на рис. 5.3.

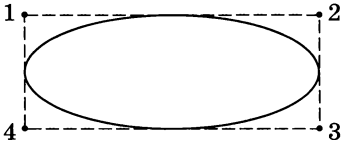


Рис. 5.3

Чтобы определить размеры овала, нужно протащить указатель мыши из любого угла описанного прямоугольника в противоположный угол при нажатой левой кнопке.

Выясните, какие будут цвета у контура и заливки овала при работе левой кнопкой мыши и правой кнопкой мыши.

Проверьте, что будет, если рисовать овал при нажатой клавише *Shift*.

Инструмент *Многоугольник* позволяет строить многоугольники (замкнутые ломаные линии). Первая сторона многоугольника рисуется так же, как отрезок, а потом все следующие вершины устанавливаются щелчками мышью. Двойной щелчок заканчивает рисование и замыкает ломаную линию в многоугольник. Многоугольник — это замкнутая фигура, для него можно выбрать режим заливки, так же как для прямоугольника и овала.

Выводы

- Растровые графические редакторы служат для создания и редактирования растровых рисунков, т. е. рисунков, образованных отдельными пикселями.
- Растровые редакторы позволяют рисовать линии, прямоугольники, овалы и другие фигуры, делать текстовые надписи.
- После того как фигура или текст зафиксированы в рисунке, он превращается в набор пикселей.
- При рисовании используются два цвета: основной и фоновый. Эти цвета можно выбрать из палитры.

Интеллект-карта

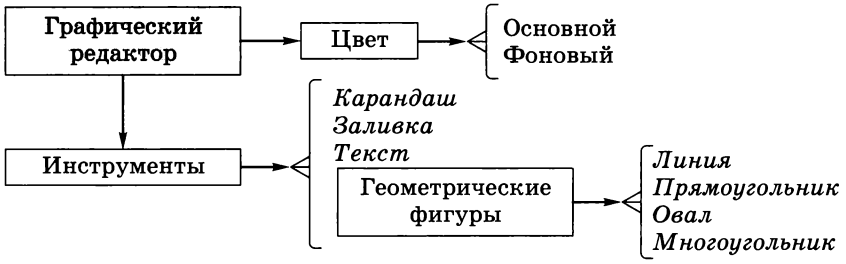


Рис. 5.4

Вопросы и задания

1. Зачем нужны два рабочих цвета?
2. В каких случаях может быть залито всё поле рисунка? Как это исправить?
3. Изменяются ли при масштабировании размеры рисунка в пикселях?
4. Как можно редактировать текст, который уже зафиксирован в растровом рисунке?

5. Чем отличается рисование фигур с помощью левой и правой кнопок мыши?
6. Как нарисовать только контур фигуры (без заливки)?
7. Как нарисовать фигуру без контура?
8. Как нарисовать круг, если вы знаете координаты его центра и радиус?
9. Почему уже нарисованную фигуру сложно переместить?
10. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Растровые графические редакторы»
б) «Графический редактор *Paint.NET*»
в) «Графический редактор *GIMP*»
г) «Графические онлайн-редакторы»
д) «Форматы растровых рисунков»
е) «Что такое разрешение?»
ж) «Как кодируется цвет?»
з) «Геометрические фигуры в редакторе *Paint.NET*»
и) «Геометрические фигуры в редакторе *GIMP*»

Интересные сайты

getpaint.net — графический редактор *Paint.NET*
gimp.org — графический редактор *GIMP*
gimpart.org – уроки по редактору *GIMP*
psd.ru — уроки по редактору *Photoshop*
apps.pixlr.com/editor/ — онлайн-редактор растровой графики
sumopaint.com — онлайн-редактор растровой графики

Практическая работа

Выполните практическую работу № 13 «Растровый графический редактор».

§ 25
Работа с фрагментами

Ключевые слова:

- фрагмент
- выделение области
- прямоугольная область
- область произвольной формы

Выделение области

С помощью инструментов группы *Выделение* (рис. 5.5) вы можете выделить некоторую область рисунка (**фрагмент**) для того, чтобы затем что-то с ней сделать: удалить, переместить, скопировать, повернуть, изменить размеры, сохранить на диске.

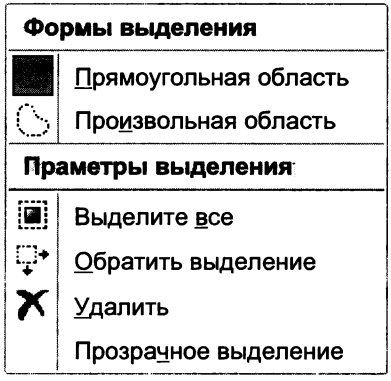


Рис. 5.5

Прямоугольная область выделяется так же, как рисуется прямоугольник, а произвольную область нужно обвести мышью при нажатой левой кнопке. Если установить флажок *Прозрачное выделение*, то пиксели, закрашенные цветом фона, не будут выделены.

Исследуйте, какая часть рисунка выделяется при нажатии комбинации клавиш *Ctrl+A*.

Выделенная область обводится штриховой рамкой. Эта линия временно видна только на экране. Она не меняет пиксели

рисунка в памяти компьютера и бесследно исчезает, если отменить выделение, щёлкнув мышью вне рамки или выбрав другой инструмент.

Что можно делать с фрагментом?

Проверьте, что произойдёт с фрагментом, если нажать клавишу *Delete*.

Выделенную область (фрагмент) можно перетаскивать мышью, при этом освободившееся место заливается цветом фона. Если при перетаскивании удерживать клавишу *Ctrl*, фрагмент будет скопирован.

Чтобы зафиксировать фрагмент при перемещении или копировании, нужно щёлкнуть мышью вне его рамки, при этом часть изображения, накрытая этим фрагментом, пропадает.

Чтобы изменить размер фрагмента, нужно перетащить мышью маркеры на рамке выделенной области.

Если снова выделить тот же фрагмент и передвинуть его, мы увидим только фоновый цвет. Это связано с тем, что простые графические редакторы (*Paint, KolourPaint*) работают только с **однослойными** (плоскими) рисунками, в которых все пиксели находятся на «одной высоте». Более сложные редакторы (*Adobe Photoshop, GIMP*) умеют работать с **многослойными** изображениями: слои располагаются друг над другом, и при изменении пикселей одного слоя пиксели остальных слоев не меняются. При этом можно легко создавать сложные рисунки из частей, не стирая пиксели, которые могут потом понадобиться. Например, надписи на рисунках удобно делать в отдельных слоях.

Кнопка Повернуть ▾ открывает меню, позволяющее поворачивать фрагмент на углы 90, 180 и 270°, а также строить его отражение по горизонтали и вертикали.

Проверьте, можно ли повернуть фрагмент с помощью контекстного меню.

Многие графические редакторы позволяют сохранять выделенную область в отдельном файле и вставлять в рисунок фрагменты с диска. Обычно это делается через контекстное меню, которое содержит команды *Копировать в файл* и *Вставить из файла*.

Выводы

- В растровых редакторах можно выделить фрагмент (часть рисунка) прямоугольной или произвольной формы.
- Фрагмент можно переместить, скопировать, удалить, изменить его размеры, сохранить в виде файла.

Интеллект-карта

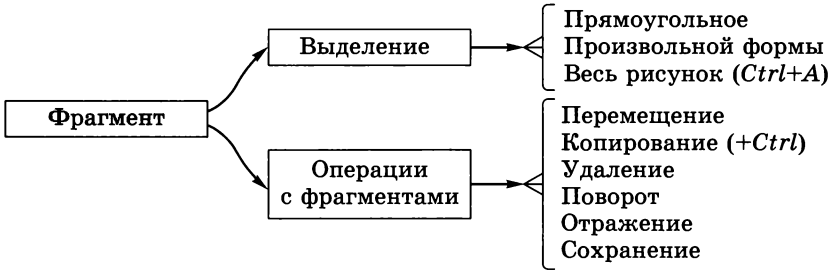


Рис. 5.6

Вопросы и задания

1. Сравните два способа выделения областей. В каких случаях применяется каждый из них?
2. Как выделить объект, не затрагивая фон?
3. Какие сложности могут возникнуть при перемещении фрагментов сложных рисунков? Чем они вызваны?
4. Объясните различие между однослойными и многослойными документами.
5. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Выделение областей в редакторе *Paint.NET*»
б) «Выделение областей в редакторе *GIMP*»
в) «Работа со слоями в редакторе *Paint.NET*»
г) «Работа со слоями в редакторе *GIMP*»

Практическая работа

Выполните практическую работу № 14 «Работа с фрагментами».

Проект

С помощью графического редактора нарисуйте открытку к любому празднику.

§ 26
Обработка фотографий

Ключевые слова:

- кадрирование
- вращение
- отражение
- перспектива
- гистограмма
- коррекция уровней
- яркость
- контраст
- коррекция цвета
- эффект «красных глаз»

Наверняка каждый из вас фотографировал с помощью цифрового фотоаппарата или мобильного телефона, а затем выкладывал фотографии в сеть Интернет. Профессиональные фотографы перед тем, как показать фотографию зрителям, всегда обрабатывают её — устраняют недостатки, вызванные плохими условиями съёмки или неправильной настройкой фотокамеры. С помощью графического редактора можно также исправить искажения, которые вносит сам фотоаппарат.

Так как растровый рисунок хранится как набор чисел, кодирующих цвета пикселей, коррекция фотографий сводится к математической обработке этих данных.

В этом параграфе вы познакомитесь с простыми приёмами обработки фотографий. Для выполнения практической работы можно использовать любой графический редактор, который умеет выполнять кадрирование, а также коррекцию уровней, цвета,

яркости и контрастности, например свободно распространяемый кроссплатформенный редактор **GIMP**.

В этом параграфе нет названий пунктов (частей параграфа) — места для них обозначены горизонтальными полосами. Прочитайте внимательно текст и запишите названия этих пунктов в тетрадь.

1. _____

После ввода изображения в компьютер (со сканера, фотоаппарата или телефона), как правило, выполняют **кадрирование**, т. е. выбирают границы изображения, обрезают лишнее. Рассмотрим случай, когда фотография при сканировании была положена неровно, так что её нужно сначала повернуть, а потом обрезать лишние поля (рис. 5.7).

Рис. 5.7

Сначала повернём изображение так, чтобы стороны фотографии стали параллельны сторонам экрана. Для этого нужно включить инструмент *Вращение* (в *GIMP* — кнопка  на панели инструментов) и выбрать нужный угол поворота (мышью или в диалоговом окне (рис. 5.8).

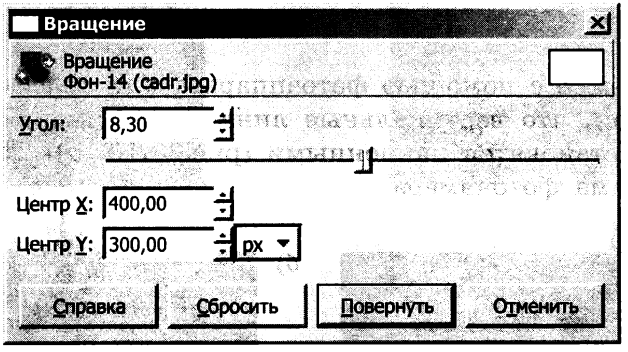


Рис. 5.8

Повернуть рисунок сразу на 90 и 180° можно с помощью верхнего меню (в *GIMP*: *Изображение* → *Преобразования*). Поворот на 90° особенно полезен: если ширина и высота рисунка сильно различаются, при повороте на большой угол части нужного изображения могут оказаться за границами рабочей области и будут обрезаны.

Теперь, когда стороны рисунка параллельны сторонам экрана, выполняем **кадрирование**[1)] с помощью инструмента *Кадрирование* (в *GIMP* — кнопка ). Выделим прямоугольную область, оставив только нужную часть рисунка. Углы выделенной области можно перетаскивать мышью (рис. 5.9).

После нажатия клавиши *Enter* поля будут обрезаны.

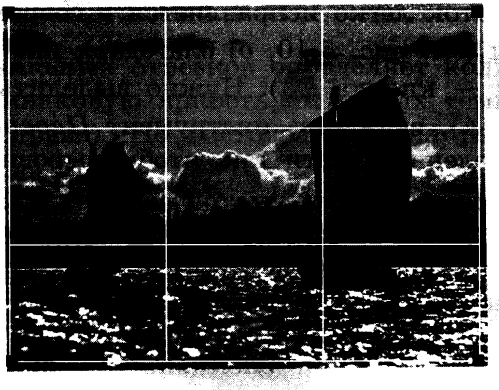


Рис. 5.9

1) В *Adobe Photoshop* кадрирование и поворот выполняются сразу (рамку кадра можно вращать).

2.

При съёмке с помощью фотоаппарата или телефона часто получается так, что вертикальные линии (например, стены домов) на снимке становятся наклонными (рис. 5.10, *а*). Такие искажения даёт сама фотокамера.

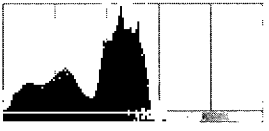
а)

Рис. 5.10

Этот недостаток легко исправляется в современных графических редакторах (см. рис. 5.10, *б*) благодаря инструменту *Перспектива* (в *GIMP* — кнопка Нужно выделить всю фотографию (клавиши *Ctrl+A*), включить инструмент *Перспектива* и перетащить углы выделенной части так, чтобы выровнять вертикальные линии.

3.

Очень полезную информацию для оценки изображения даёт **гистограмма** — диаграмма специального вида, которая показывает распределение пикселей **по яркости**. Чтобы увидеть гистограмму загруженного изображения, в редакторе *GIMP* нужно выбрать пункт меню *Окна* → *Прикрепляющиеся диалоги* → *Гистограмма.*



1

2

3

4

Рис. 5.11

Высота вертикальных отрезков определяет количество пикселей, имеющих одинаковую яркость, от самых тёмных (слева) до самых светлых (справа). По гистограммам на рис. 5.11 видно, что:

- фото 1 слишком тёмное, потому что все пиксели сосредоточены в области тёмных тонов (в левой части гистограммы);
- фото 2 слишком светлое (нет тёмных тонов);
- фото 3 малоконтрастное (есть средние тона, но нет тёмных и светлых);
- фото 4 сбалансированное, оно содержит пиксели всех уровней яркости[1)].

---

1) На этой фотографии — картина «Пристань» художника К. А. Гоголева, вырезанная из дерева

Изображениям 1–3 не хватает **контраста**. Чтобы их улучшить, нужно «растянуть» гистограмму так, чтобы она занимала весь диапазон тонов, от чёрного до белого. Эта операция называется **коррекция уровней** (в *GIMP* — меню *Цвет* → *Уровни*[1]). В появившемся окне нужно отрегулировать положение чёрного ▲, серого ▲ и белого △ движков под гистограммой (рис. 5.12, *а*).

а) *б*)

Рис. 5.12

Все пиксели слева от чёрного движка становятся чёрными. Те, что оказались справа от белого движка, станут белыми. Таким образом, вся область между чёрным и белым движками растянется на весь диапазон яркостей. Передвигая серый движок (по умолчанию он находится посередине между чёрным и белым), можно менять контраст средних тонов. При этом в окне изображения мы сразу видим результат (такой эффект называется **предварительным просмотром** — это просмотр результата какой-то операции до её применения).

После коррекции уровней внешний вид фотографии значительно улучшается, она становится более контрастной. Однако гистограмма после коррекции не сплошная, а состоит из отдельных полосок (рис. 5.12, *б*). Это значит, что пикселей с некоторыми значениями яркости нет совсем (подумайте почему).

В редакторе *GIMP* есть также отдельное окно для настройки яркости и контраста (меню *Цвет* → *Яркость-Контраст*).

---

[1] В *Adobe Photoshop* — меню *Изображение* → *Коррекция* → *Уровни*.

4.

Иногда цвет объектов на фотографиях отличается от того цвета, который видит глаз. Например, листва деревьев может оказаться синеватой, а красная крыша — фиолетовой. В этом случае можно использовать коррекцию цвета, используя свои знания о том, какие цвета имеют объекты в действительности. В программе *GIMP* для этого нужно выбрать пункт главного меню *Цвет* → *Цветовой баланс*.

С помощью окна на рис. 5.13 можно отдельно настраивать цвет тёмных участков (теней), пикселей средней яркости (полутонов) и светлых частей.

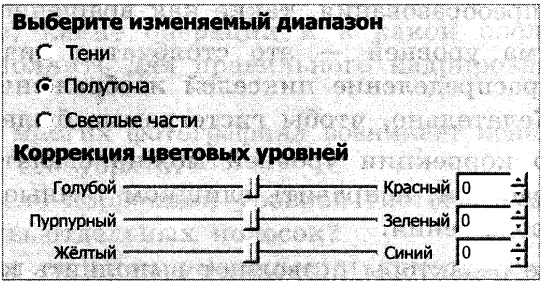


Рис. 5.13

Обратите внимание, что невозможно усилить или ослабить один какой-то цвет, оставив все остальные без изменения. Если уменьшить красную составляющую цвета, то автоматически усилится голубой цвет (см. рис. 5.13). Пары красный — голубой, зелёный — пурпурный (фиолетовый) и синий — жёлтый — это *дополнительные цвета* (увеличение одного из них вызывает уменьшение другого).

5.

При фотосъёмке со вспышкой нередко возникает **эффект «красных глаз»** — лучи вспышки отражаются от глазного дна человека (оно имеет красный цвет) и попадают в объектив фотокамеры. Это особенно заметно при съемке в тёмном помещении, когда зрачки глаз расширены. На самом деле, без вспышки мы

видим тёмный зрачок, который и нужно восстановить на фотографии.

Для устранения эффекта «красных глаз» в GIMP можно применить встроенный *фильтр* — процедуру автоматической обработки изображения. Фильтр *Улучшение → Удалить эффект красных глаз* находится в верхнем меню *Фильтры*.

Выводы

- Кадрирование — это выбор границ кадра. С помощью кадрирования можно сделать фотографию более выразительной, удалив лишние части.
- Графические редакторы позволяют выполнять с изображением различные преобразования, такие как вращение и отражение.
- Гистограммма уровней — это столбчатая диаграмма, показывающая распределение пикселей изображения по уровням яркости. Желательно, чтобы гистограмма была равномерной.
- С помощью коррекции уровней можно сделать фотографию более контрастной, исправить слишком тёмные или слишком светлые изображения.
- Графические редакторы позволяют выполнить коррекцию цвета отдельно для светлых, средних и тёмных областей фотографии.

Интеллект-карта

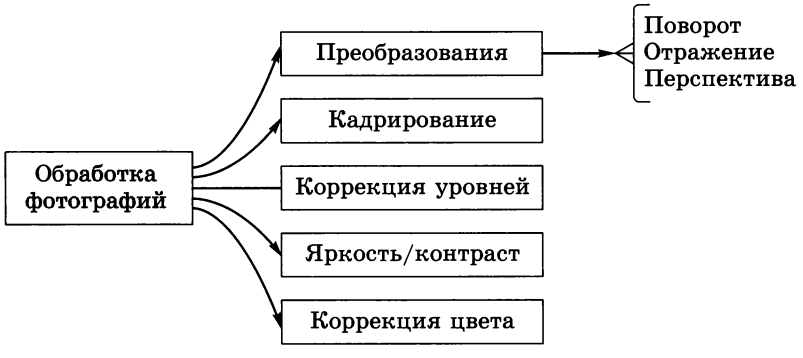
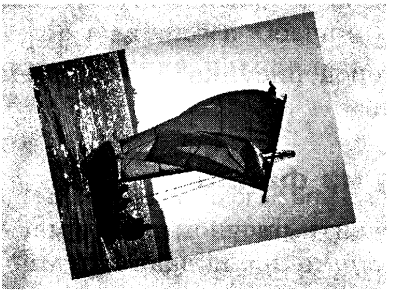


Рис. 5.14

Вопросы и задания

1. Зачем нужно кадрирование?
2. После сканирования получено такое изображение:



Определите, какие операции и в какой последовательности нужно выполнить для правильного кадрирования этой фотографии.

3. Почему на многих фотографиях возникает искажение перспективы? Как его убрать?
4. Почему после коррекции уровней гистограмма не сплошная, а состоит из отдельных полосок?
5. Почему при сильном уменьшении яркости синего цвета фотография приобретает желтоватый оттенок?
6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Как обесцветить фотографию?»
б) «Повышение резкости фотографий»
в) «Что такое ретушь?»

Интересные сайты

gimp.org — графический редактор *GIMP*
gimpart.org — уроки по редактору *GIMP*
apps.pixlr.com/editor/ — онлайн-редактор растровой графики

Практическая работа

Выполните практическую работу № 15 «Обработка фотографий».

Проект

Сделайте фотографию с помощью телефона или фотоаппарата и обработайте её в графическом редакторе.

§ 27
Вставка рисунков в текстовый документ

Ключевые слова:

- вставка рисунка
- изменение размеров рисунка
- обтекание текстом
- буфер обмена
- скриншот

Вставка рисунка из файла

Для того чтобы добавить рисунок в документ *Word*, нужно щёлкнуть на кнопке *Рисунки* на вкладке *Вставка* (рис. 5.15) и выбрать файл на диске.

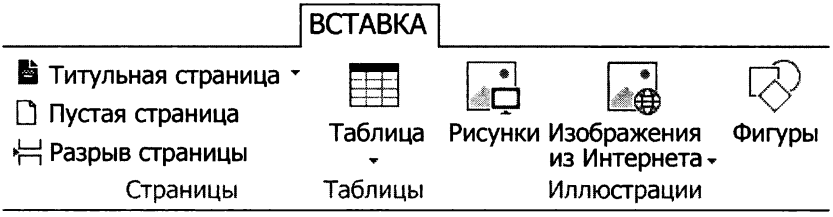


Рис. 5.15

Рисунок сначала вставляется в текст как «большая буква», раздвигая строки (рис. 5.16). Такой режим используется чаще всего для вставки небольших рисунков (например, изображения кнопки) прямо в текст.

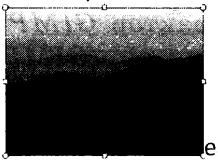
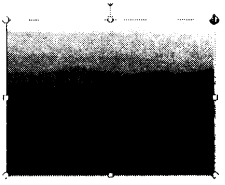
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex  ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Рис. 5.16

Рисунок можно перетаскивать мышью в любое место, его размеры изменяются с помощью маркеров на границе.

Для того чтобы большой рисунок обтекался текстом (рис. 5.17), нужно установить для него режим обтекания *Вокруг рамки*. Это можно сделать, например, с помощью контекстного меню (раздел *Обтекание текстом*). Обычно рисунок выравнивается так, чтобы граница рисунка находилась на одной вертикали с границей текста (рис. 5.17).



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Рис. 5.17

Почему рисунки с обтеканием текстом не вставляются в середину страницы, а сдвигаются к границе текста?

Вставка рисунка через буфер обмена

Как вы уже знаете, буфер обмена — это область оперативной памяти, через которую программы могут обмениваться данными. Предположим, что вы построили рисунок в графическом редакторе и хотите вставить его в документ без сохранения на диске. Для этого нужно скопировать рисунок в буфер обмена (клавиши *Ctrl+C*), перейти в окно текстового документа и вставить содержимое буфера обмена (*Ctrl+V*). Рисунок будет помещён в то место, где находится курсор.

Этот приём можно использовать и при вставке рисунков с веб-страниц: сначала рисунок копируется в буфер обмена, а затем вставляется из буфера обмена в документ.

Скриншоты

Сейчас для того, чтобы рассказать кому-то, как работать с той или иной программой, не нужно встречаться лично, а можно передать инструкцию по Интернету. Для объяснения удобно использовать снимки экрана, которые называются *скриншотами* (от англ. *screen* — экран и *shot* — фотоснимок).

Скриншот — это изображение экрана, сохранённое в памяти компьютера.

Для того чтобы сделать снимок всего экрана, нужно нажать клавишу *Print Screen* (на некоторых клавиатурах — *PrtScr*). Изображение экрана записывается в буфер обмена, откуда его можно вставить в любую программу, принимающую рисунки. Комбинация клавиш *Alt+Print Screen* сохраняет в буфере обмена только изображение активного окна.

Выводы

- В текстовые документы можно вставлять рисунки двумя способами: из файла на диске или через буфер обмена.
- Рисунок может находиться в тексте (как специальный «символ») или обтекаться текстом.
- Рисунки, для которых установлено обтекание текстом, обычно помещают у левой или правой границы страницы, так чтобы границы рисунки и текста были на одной вертикали.
- Скриншот — это изображение экрана, сохранённое в памяти компьютера.

Вопросы и задания

1. Сравните известные вам режимы обтекания рисунков. Когда они используются?
2. Какие действия нужно выполнить, чтобы вставить в документ изображение кнопки текстового процессора?
3. Как можно использовать скриншоты, если у вас возникла проблема при работе на компьютере?
4. Выполните по указанию учителя задания в рабочей тетради.

Практическая работа

Выполните практическую работу № 16 «Документы с рисунками».

Проект

Найдите в Интернете небольшой текст: сказку или рассказ (примерно на 1–2 страницы). Добавьте к тексту иллюстрации. Если вы нашли их в Интернете, не забудьте указать в конце документа авторов рисунков и/или сайты-источники.

§ 28
Векторная графика

Ключевые слова:

- векторный рисунок
- векторный редактор
- примитивы
- выделение объектов
- изменение порядка объектов
- выравнивание
- распределение
- кривая Безье
- гладкий узел
- угловой узел

Для чего нужна векторная графика?

Главный недостаток растровой графики состоит в том, что при изменении размеров рисунка он искажается: изменяется и форма объектов, и даже их цвет (вы наблюдали это во время практических работ). Всё дело в том, что программа воспринимает рисунок не как набор объектов, а как множество пикселей разного цвета. Человек же, глядя на рисунок, представляет нарисованные объекты у себя в сознании.

Растровая графика неприменима там, где нужно **масштабировать** рисунки, т. е. изменять их размеры без потери качества. Например, плакат должен хорошо выглядеть как на листе формата А4, так и на баннере, размеры которого измеряются в метрах. При увеличении растрового рисунка размер пикселей также увеличится, и изображение станет ступенчатым.

Сейчас инженеры разрабатывают новые автомобили, самолёты, приборы в системах автоматизированного проектирования. В них строится трёхмерная (объёмная) модель объекта, которую затем нужно просматривать с разных сторон и рассчитывать на прочность. В таких задачах растровая графика неприменима.

Поэтому появилась другая идея: хранить в памяти компьютера не отдельные пиксели, а информацию о *геометрических фигурах*, из которых составлен рисунок. Такие рисунки называют векторными. **Векторный рисунок** — это своеобразная программа для компьютера, выполнив которую он рисует изображение на экране.

Для работы с векторными изображениями используют специальные программы — **векторные графические редакторы**, например **Adobe Illustrator**, **CorelDraw**, **Inkscape**. В то же время рисовать простые схемы и рисунки можно даже с помощью офисных пакетов: в редакторе *Word* есть небольшой встроенный векторный редактор, а в состав *OpenOffice* входит векторный редактор **Draw**.

Примитивы

Векторный рисунок состоит из **примитивов** — элементарных фигур. К ним относятся отрезок, прямоугольник, овал (эллипс), кривая и др. На рисунке 5.18 показаны некоторые часто используемые фигуры в редакторе *Word*.

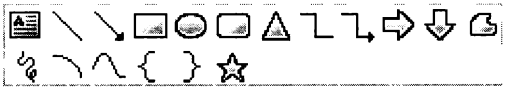


Рис. 5.18

Основные фигуры — отрезок, прямоугольник, овал (эллипс) — рисуются так же, как и в растровых редакторах, но после завершения рисования они не превращаются в набор пикселей, а остаются отдельными объектами, каждый из которых можно редактировать (перемещать, изменять свойства и даже удалить) независимо от других.

Чтобы что-то сделать с объектом, его нужно **выделить** щелчком мышью.

Попробуйте щёлкать мышью на объектах при нажатой клавише *Shift*. Что получается?

На рамке выделенного объекта появляются маркеры, перетащив которые, можно изменить размеры фигуры. Оба конца отрезка тоже можно перетаскивать мышью.

Чтобы переместить объект, нужно «схватить» его за контур.

Что получится, если при перетаскивании удерживать нажатой клавишу *Ctrl*?

Нажатие клавиши *Delete* удаляет выделенные объекты.

Для контура любой фигуры можно задать цвет, толщину, стиль линии (сплошная, штриховая, точечная и др.). На концах отрезка могут быть стрелки различной формы и размеров. В редакторе *Word* для настройки контура используется меню *Контур фигуры* на вкладке *Формат*, а в *OpenOffice Writer* — панель *Свойства*.

С помощью меню *Заливка фигуры* (для *Word* — на вкладке *Формат*) можно выбрать цвет заливки (или отключить заливку вообще). Кроме того, возможны и другие типы заливки:

- *градиент* — плавный переход между двумя или несколькими цветами;
- *рисунок* (для заполнения области используется изображение из файла);
- *текстура* — рисунок, имитирующий материал (бумагу, холст, дерево и др.);
- *узор* — рисунок, составленный из двух цветов.

Изменение порядка элементов

Новые фигуры добавляются на рисунок поверх существующих. Иногда нужно изменить порядок их расположения, например переместить какую-то фигуру впереди (выше) всех остальных объектов. Для этого в *Word* служат команды вкладки *Формат:*

- *На передний план* (поверх всех остальных объектов);
- *Переместить вперед* — поменять местами выделенный объект с тем, который находится непосредственно *над* ним;
- *На задний план* (позади всех остальных объектов);
- *Переместить назад* — поменять местами выделенный объект с тем, который находится непосредственно *под* ним.

Так можно перемещать (по уровням) сразу несколько выделенных объектов. Все эти команды доступны из контекстного меню.

На рисунке 5.19 показан результат применения команды *На передний план* к объекту с белой заливкой.

Рис. 5.19

Если выделить не белую фигуру, а самую верхнюю, какую команду нужно применить для того, чтобы получить точно такой же результат?

Выравнивание, распределение

При работе с векторными рисунками часто возникают такие задачи:

- **выровнять** несколько объектов по какому-то краю (например, по верхней границе) — рис. 5.20;

Рис. 5.20

- **распределить** несколько объектов так, чтобы они оказались на равных расстояниях по горизонтали или по вертикали (рис. 5.21).

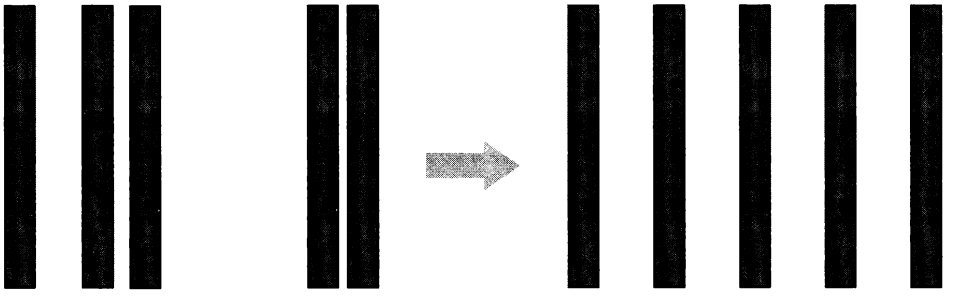


Рис. 5.21

Конечно, всё можно сделать вручную, передвигая мышью отдельно каждую фигуру, но это очень долго и неточно. В текстовых процессорах эти операции автоматизированы. В программе *Microsoft Word* они выполняются с помощью команд группы *Выровнять* на вкладке *Формат*, а в *OpenOffice Writer* — с помощью раздела *Выравнивание* в контекстном меню.

Вспомните, как выделить сразу несколько объектов.

Группировка

Предположим, что ваш векторный рисунок строится из сложных объектов (например, дом, автомобиль, собака, человек), каждый из которых в свою очередь тоже состоит из частей.

Для того чтобы передвинуть такой сложный объект, нужно переместить все его части, не сдвинув их друг относительно друга. Лучше всего заранее **сгруппировать** все эти части (например, все элементы дома) в единый объект с помощью команды *Сгруппировать* из контекстного меню. После этого весь объект выделяется одним щелчком мышью и перетаскивается за один раз.

Сгруппированный объект имеет общую рамку, поэтому очень легко изменять его размеры с помощью маркеров этой рамки (рис. 5.22). Сделать это без группировки крайне сложно.

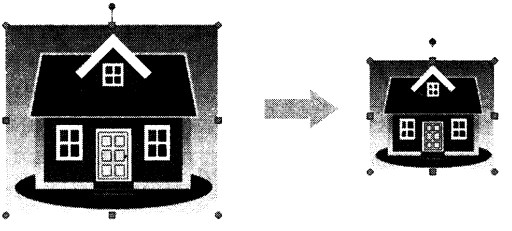


Рис. 5.22

Выясните, как в вашей программе можно отменить группировку и разделить сложный объект на составные части.

Увеличьте сгруппированный рисунок так, чтобы он занял весь экран. Появились ли искажения, которые возникают при увеличении растровых рисунков?

Половина бабочки нарисована в векторном редакторе как набор мелких элементов. Нужно построить полное увеличенное изображение бабочки и разместить его по центру прямоугольника с фоном (рис. 5.23). Выделите подзадачи, которые нужно решить, предложите последовательность действий, проверьте её на компьютере.

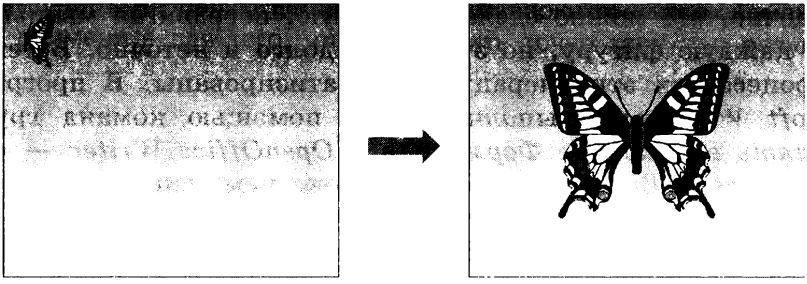


Рис. 5.23

Кривые

К сожалению, не все рисунки можно построить из готовых примитивов (отрезков, прямоугольников, овалов). Большинство элементов, из которых состоят рисунки профессиональных дизайнеров, — это **кривые.** В памяти компьютера хранятся координаты опорных точек (узлов), а сама форма кривой между узлами описывается математической формулой.

На рис. 5.24 изображена кривая с опорными точками *А*, *Б*, *В*, *Г* и *Д*.

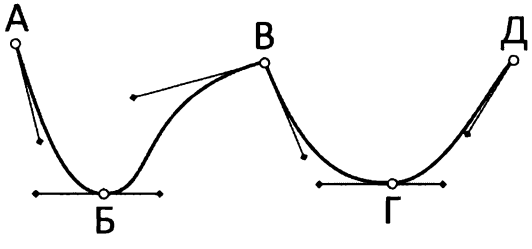


Рис. 5.24

У каждой из этих точек есть **рычаги** (*управляющие линии*); перемещая концы рычагов, можно менять кривизну всех участков

кривой (*сегментов*). Если оба рычага находятся на одной прямой, получается **гладкий узел** (*Б* и *Г*), если нет — **угловой узел** (*В*). Таким образом, форма кривой полностью задаётся координатами опорных точек и рычагов. Кривые, заданные таким образом, называют **кривыми Безье** в честь их изобретателя французского инженера Пьера Безье.

В редакторе *Word* кривую можно нарисовать с помощью инструментов *Кривая*  и *Полилиния*  . Для того чтобы редактировать кривую, нужно щёлкнуть правой кнопкой мыши на ней и выбрать команду *Начать изменение узлов* в контекстном меню. При этом появляется возможность:

- перемещать, удалять и добавлять узлы;
- изменять тип узла (с гладкого на угловой и обратно);
- изменять положение рычагов, регулирующих кривизну.

В *OpenOffice Writer* для рисования кривых служит инструмент *Объект кривых*  на панели *Рисование*. Кнопка *Изменение геометрии*  на этой же панели включает режим редактирования кривой.

Как вы думаете, в каких задачах лучше использовать растровую графику, а в каких — векторную?

Выводы

- Векторный рисунок хранится в памяти как набор команд для рисования простейших геометрических фигур (графических примитивов).
- Для каждой векторной фигуры определяется контур (цвет, толщина, стиль линии, стрелки) и заливка (сплошной цвет, градиент, текстура или растровый рисунок).
- Фигуры — это независимые объекты, каждый из которых можно редактировать отдельно от других.
- Фигуры можно перемещать друг относительно друга (как в плоскости рисунка, так и по уровням), выравнивать, распределять равномерно по горизонтали и вертикали.
- Чтобы было удобнее работать с составными объектами, их можно сгруппировать.
- Векторные кривые хранятся в памяти как набор узлов. Узлы могут быть гладкие и угловые. Форма кривой между узлами рассчитывается по математическим формулам.
- Качество векторных изображений не снижается при изменении их размеров.

Интеллект-карта

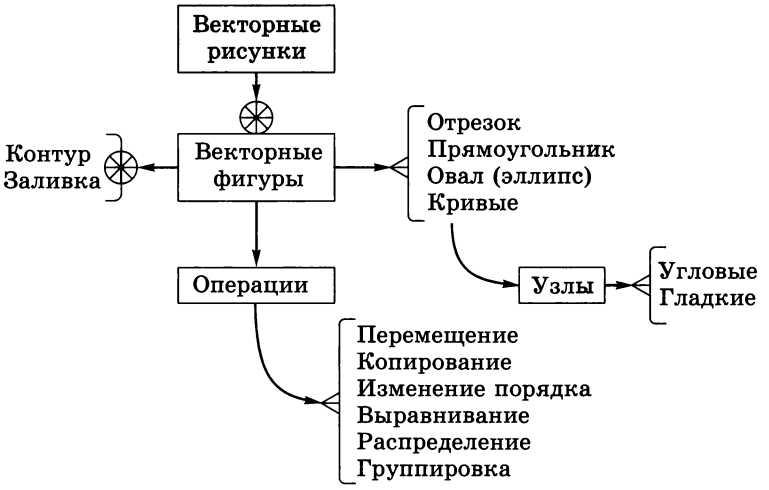


Рис. 5.25

Вопросы и задания

1. Обсудите, можно ли сохранить фотографию в векторном формате.
2. От чего зависит размер файла, в котором хранится векторный рисунок?
3. Почему векторные рисунки не искажаются при изменении размеров?
4. Как можно изменять порядок расположения элементов рисунка? Зачем это нужно?
5. Как можно переместить сложный объект, не группируя его? Сравните два способа — с группировкой и без группировки.
6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Векторная графика вокруг нас»
б) «Векторный редактор *OpenOffice Draw*»
в «Векторный редактор *Inkscape*»
г) «Векторная графика в редакторе *GIMP*»
д) «Векторная графика на веб-страницах»

Интересные сайты

inkscape.org — редактор *Inkscape*
inkscape.paint-net.ru — уроки по редактору *Inkscape*
wiki.linuxformat.ru/wiki/LXF74-75:Inkscape — уроки по редактору *Inkscape*
corel.demiart.ru — уроки по редактору *CorelDraw*
coreldrawgromov.ru — уроки по редактору *CorelDraw*

Практическая работа

Выполните практическую работу № 17 «Векторная графика».

Проект

С помощью векторного редактора нарисуйте эмблему (*логотип*) фирмы, которую вы хотели бы создать в будущем.

ЭОР к главе 5 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Компьютерная графика. Модуль 2. Графический редактор MS PAINT. 5–9 классы
Базовые инструменты в Paint
Особенности профессиональной работы в PhotoShop
Среда графического редактора Paint
Рисование геометрических фигур в Paint
Инструменты выделения
Работа с фрагментами изображения в Paint
Работаем со слоями
Анимация «Редактирование растровых изображений»
Открываем существующий графический файл
Растровая и векторная графика
Графические форматы. Векторные форматы. CDR (CorelDraw)
Интерфейс графического редактора Corel Draw
Действия с объектами в CorelDraw
Изображение объектов в CorelDraw

Глава 6
АЛГОРИТМЫ И ПРОГРАММИРОВАНИЕ

§ 29
Алгоритмы и исполнители

Ключевые слова:

- алгоритм
- исполнитель
- система команд исполнителя
- среда исполнителя
- состояние исполнителя
- дискретность
- понятность
- определённость
- конечность
- корректность
- массовость
- управление с пульта
- программное управление

Что такое алгоритм?

Многие действия, которые мы выполняем в жизни, можно записать как последовательность шагов.

Например, когда строят дом, сначала заливают фундамент, затем возводят стены, потом делают крышу. Бригадир строителей точно знает, как правильно выполнить каждую операцию. Менять порядок действий нельзя, иначе стройка закончится неудачно.

В этой главе мы будем изучать такие точно определённые правила выполнения действий. В информатике их называют *алгоритмами*.

Слово «алгоритм» происходит от имени средневекового арабского учёного **Мухаммеда аль-Хорезми**, который в IX веке описал правила вычислений с десятичными числами. Работы аль-Хорезми были переведены на латинский язык и стали известны в Европе. Через некоторое время слово «алгоритм» (от имени автора, которое по-латыни писали как *Algorizmi* или *Algorizmus*) стало обозначать любую систему вычислений по определённым правилам.

Опишите алгоритмы:

а) пополнения счёта мобильного телефона;

б) поездки на автобусе в соседний город.

Алгоритмы составляют люди, причём нередко на разработку алгоритмов решений сложных задач уходят многие годы. Но как только алгоритм придуман, можно поручить его выполнение автомату (например, компьютеру), который просто выполняет инструкции, не вникая в их смысл. Говорят, что такой исполнитель действует **формально**, не рассуждая и не обдумывая содержание задачи, которую он решает.

Как правило, любая задача может быть решена с помощью разных алгоритмов. Их можно сравнивать, например, по времени, которое требуется для решения задачи. Лучшим обычно считают алгоритм, который решает задачу быстрее всех.

Компьютерные программы — это алгоритмы, записанные на языке, понятном компьютеру. Поэтому изучать алгоритмы нужно для того, чтобы понимать, как работает компьютер, и научиться решать с его помощью сложные задачи.

Приведите примеры алгоритмов, которые умеет выполнять:

а) ребёнок-дошкольник;

б) собака;

в) мобильный телефон;

г) стиральная машина.

Сможет ли годовалый ребенок сходить в магазин за хлебом? Сможет ли собака оплатить счёт за квартиру? Конечно, нет. Любой алгоритм составляется для какого-то конкретного **исполнителя**.

Исполнитель — это человек, животное или машина, которые могут понимать и выполнять некоторые команды. Полный набор команд исполнителя называется **системой команд исполнителя (СКИ)**.

Теперь можно дать определение алгоритма.

Алгоритм — это точное описание порядка действий некоторого исполнителя.

Система команд простейшего **исполнителя Робот**, который умеет только передвигаться по ячейкам клетчатой доски, содержит команды `вверх`, `вправо`, `вниз` и `влево` (рис. 6.1).

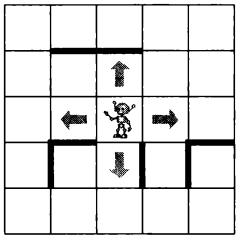


Рис. 6.1

Других команд Робот не понимает и выполнить не может.

Любой исполнитель работает в некотором окружении — **среде**. Среда исполнителя Робот — клетчатое поле со стенами, которые обозначены жирными линиями. Робот не может ходить сквозь стены. Например, Робот на рис. 6.1 не может выполнить последовательность команд

```
вверх
вверх
```

потому что столкнётся со стеной.

Исполнители могут находиться в разных **состояниях** и в зависимости от этого могут (или не могут) выполнять разные команды. Например, незаряженное ружьё не выстрелит, а лежащий человек не сможет прыгнуть, не изменив своё состояние. Грузовик с пустым кузовом не сможет засыпать яму песком.

Что нужно для того, чтобы описать состояние исполнителя Робот? В каких случаях этот исполнитель не сможет выполнить некоторые команды?

Свойства алгоритма

Есть три свойства, которыми обладают все алгоритмы. Это значит, что если какое-то описание действий не обладает хотя бы одним из этих свойств, то это уже не алгоритм.

1. **Дискретность** — алгоритм состоит из отдельных команд, каждая из которых выполняется ограниченное (не бесконечное) время.

2. **Понятность** — алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен.

3. **Определённость** — при каждом выполнении алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат. Другими словами, исполнитель должен однозначно понимать команду и последовательность выполнения команд и выполнять их каждый раз одинаково.

Алгоритмы часто используются для решения **массовых задач**, т. е. задач, которые нужно уметь решать при разных исходных данных. Примеры таких задач: найти наибольшее из двух чисел; подсчитать количество слов в тексте и т. п. В таких случаях иногда говорят ещё о некоторых дополнительных свойствах алгоритмов.

4. **Конечность (результативность)** — для любых допустимых исходных данных алгоритм должен заканчиваться с некоторым результатом. Если задача не имеет решения, алгоритм должен остановиться и сообщить об этом.

5. **Корректность** — для любых допустимых исходных данных алгоритм должен приводить к правильному решению задачи.

6. **Массовость** — алгоритм можно использовать для решения множества однотипных задач с различными исходными данными (при этом писать алгоритм заново не нужно!). Поэтому обычно для составления алгоритма задачу надо решить «в буквах», вводя имена для исходных данных.

Подчеркнём, что в отличие от свойств 1–3 свойства 4–6 — необязательные. Они выполняются не для всех алгоритмов. Например, при некоторых исходных данных работа алгоритма может никогда не заканчиваться. В этом случае результат работы алгоритма не определён; говорят, что алгоритм **зациклился**.

Обычно алгоритмы сравнивают по времени выполнения или по количеству действий. При этом лучшим считается тот алгоритм, который быстрее приводит к правильному результату (требует меньше действий исполнителя). Кроме того, нужно учитывать ещё и количество памяти, которое требуется для работы алгоритма. Поэтому выбор алгоритма зависит от особенностей задачи, которую вы решаете.

Как управляют исполнителями?

Исполнителями можно управлять двумя способами — вручную и по программе.

Ручное управление (непосредственное управление) означает, что человек по очереди отдаёт исполнителю одну команду за дру-

гой. Исполнитель тут же выполняет каждую введённую команду. Такое управление иногда называют «управлением с пульта», имея в виду пульт управления, с которого передаются команды исполнителю (рис. 6.2). При этом в начале работы у человека может вообще не быть никакого плана действий — он может просто играть с исполнителем.

Рис. 6.2

Однако в жизни часто без заранее составленного плана действий (алгоритма) не обойтись. Это особенно важно, если нужно передавать команды исполнителю очень часто, как, например, при управлении станком. В этом случае исполнителем управляет автомат (например, компьютер) по готовому алгоритму. Алгоритм работы исполнителя должен быть записан на специальном языке, понятном компьютеру.

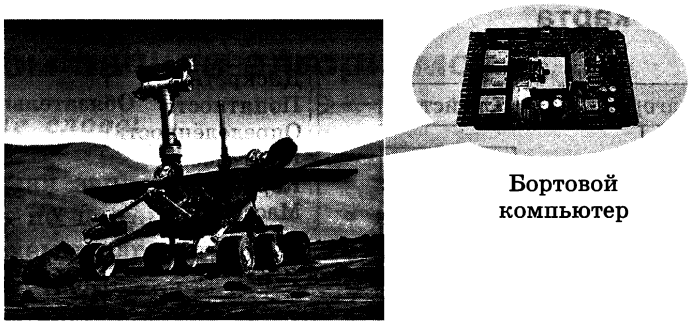
Алгоритм, записанный на языке, понятном компьютеру, называется **программой**.

Получив программу, компьютер выполняет её: посылает исполнителю команды в нужной последовательности. Всё это происходит без участия человека (рис. 6.3).

При **программном управлении** компьютер управляет исполнителем по готовой программе.

Программа должна быть заранее записана в память компьютера.

Как могло бы выглядеть управление стиральной машиной с помощью пульта? Почему такой способ не используется?



Бортовой компьютер

Марсоход

Рис. 6.3

Выводы

- Алгоритм — это точное описание порядка действий некоторого исполнителя.
- Любой алгоритм составляется для какого-то определённого исполнителя.
- Исполнитель — это человек, животное или машина, которые могут понимать и выполнять некоторые команды. Полный набор команд исполнителя называется системой команд исполнителя (СКИ).
- Исполнители могут находиться в разных состояниях и в зависимости от этого могут (или не могут) выполнять разные команды.
- Основные свойства алгоритма — дискретность, понятность и определённость.
- Как правило, для решения задачи можно придумать несколько различных алгоритмов. Алгоритмы сравниваются по скорости работы (по количеству действий исполнителя) и по используемой памяти.
- Исполнителем можно управлять вручную («с пульта») или по программе.
- Ручное (непосредственное) управление означает, что человек по очереди отдаёт исполнителю одну команду за другой. При этом никакого плана управления исполнителем может не быть вообще.
- При программном управлении в память управляющего компьютера заранее записывается алгоритм, который ему нужно выполнить.
- Алгоритм, записанный на языке, понятном компьютеру, называется программой.

Интеллект-карта

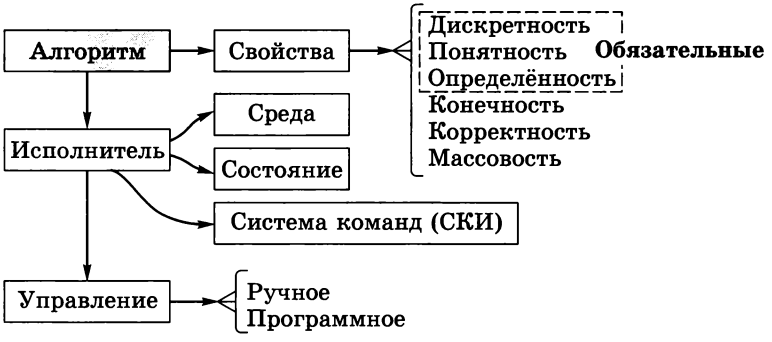


Рис. 6.4

Вопросы и задания

1. Сформулируйте алгоритм:
   а) заварки чая (как это делаете вы);
   б) перехода через улицу по пешеходному переходу со светофором;
   в) покупки бананов в магазине;
   г) заправки автомобиля топливом;
   д) оплаты мобильной связи через терминал.
2. Сформулируйте алгоритмы
   а) вычисления среднего арифметического двух чисел;
   б) вычитания однозначного числа из двузначного;
   в) умножения двузначного числа на однозначное;
   г) вычисления остатка от деления одного целого числа на другое.
3. Какими свойствами обладает любой алгоритм?
4. Чем отличаются управление с пульта и программное управление? В каких случаях лучше работает каждый из методов?
5. Что значит фраза «алгоритм зациклился»?
6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Аль-Хорезми и его вклад в науку»

Практическая работа

Выполните практическую работу № 18 «Управление исполнителем с пульта».

§ 30
Способы записи алгоритмов

Ключевые слова:

- словесная запись
- запись по шагам
- блок-схемы алгоритмов
- ручная прокрутка
- переменная
- присваивание
- программа
- язык программирования
- язык ассемблера
- языки высокого уровня
- логическое программирование

Словесная запись

Как можно записать алгоритм? В первую очередь на естественном языке (русском, английском и др.).

Алгоритм «О». Даны два натуральных числа. Пока первое число не меньше второго, заменять его на разность первого и второго. Результат работы алгоритма — полученное первое число.

Применим этот алгоритм к числам 5 и 2. Обозначим первое число буквой a, а второе — буквой b. Изменения этих величин во время работы алгоритма сведены в табл. 6.1.

Таблица 6.1

Число	Исходные данные	Шаг 1	Шаг 2
a	5	$5 - 2 = 3$	$3 - 2 = 1$
b	2	2	2

Алгоритм закончил работу, потому что после двух шагов первое число (a) стало равно 1 и оказалось меньше, чем второе (b). Результат работы алгоритма — первое число $a = 1$.

Словесная запись удобна и привычна, но есть одна проблема: во всех естественных языках есть неоднозначность, поэтому исполнитель-человек может понять алгоритм не так, как задумывал его автор. Особенно трудно разбираться в длинных словесных алгоритмах, занимающих больше десятка строк.

Запись по шагам

Для того чтобы в алгоритме легче было разобраться, отдельные шаги часто записывают в виде нумерованного списка. Алгоритм «О» может быть записан так:

Вход: два натуральных числа, a и b.
Шаг 1. Если $a < b$, перейти к шагу 4.
Шаг 2. Заменить a на $a - b$.
Шаг 3. Перейти к шагу 1.
Шаг 4. Стоп.
Результат: значение a.

Блок-схемы алгоритмов

Записывать алгоритмы на естественных языках не всегда удобно. Например, если записать алгоритм на русском языке, то люди, не умеющие читать по-русски, не смогут им воспользоваться. В сложных алгоритмах непросто отслеживать переходы между шагами. Поэтому придумали **графическую форму** записи алгоритмов — **блок-схемы**. Вот как выглядит блок-схема алгоритма «О» (рис. 6.5).

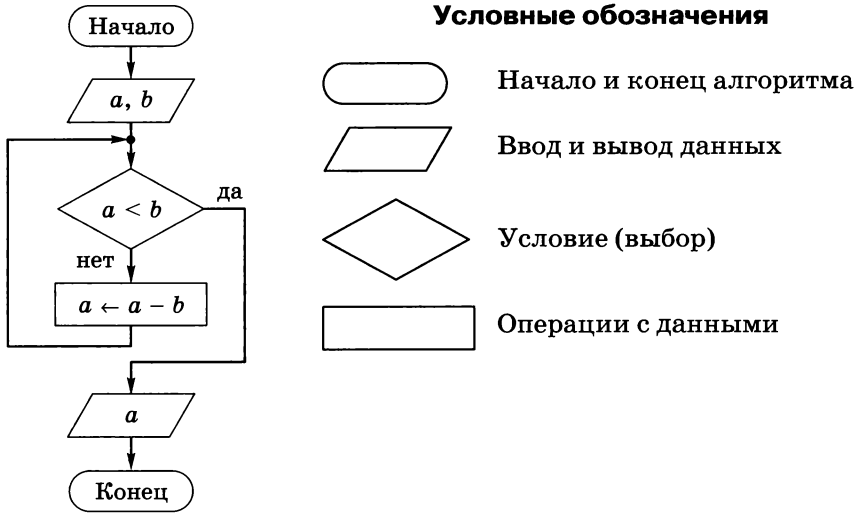
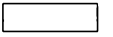
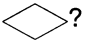


Рис. 6.5

Постарайтесь самостоятельно разобраться в этой схеме, сравнивая её со словесной записью и записью по шагам. Ответьте на вопросы.

— Что означают стрелки?
— Как найти исходные данные для работы алгоритма?
— Как увидеть результат работы алгоритма?
— Как записывается операция «заменить a на разность a и b»?
— Сколько входов и выходов имеет блок  ?
— Сколько входов и выходов имеет блок  ?
— Почему у двух выходов блока  записаны слова «да» и «нет»?
— Что означает замкнутый контур (возврат к одному из предыдущих действий)?

Ручная прокрутка

Иногда для того, чтобы разобраться в работе алгоритма или найти ошибку в нём, выполняют «ручную прокрутку» («проигрывание») алгоритма. Ручную прокрутку называют трассировкой.

Ручная прокрутка (трассировка) — это выполнения алгоритма человеком вручную.

Выполним вручную все шаги алгоритма «О» при $a = 19$ и $b = 5$. Чтобы не держать в памяти все промежуточные результаты, составим таблицу, в которой будем записывать выполняемые шаги, а также изменения величин a и b (табл. 6.2).

Таблица 6.2

	Действие	Условие верно?	a	b	
1			19	5	Исходные данные
2	$a < b$?	Нет			
3	$a \leftarrow a - b$		14		
4	$a < b$?	Нет			
5	$a \leftarrow a - b$		9		
6	$a < b$?	Нет			
7	$a \leftarrow a - b$		4		
8	$a < b$?	Да			
9	Стоп				

Как вы знаете, алгоритм выполняется по шагам (свойство дискретности). В первом столбце таблицы записаны шаги-действия. Это может быть, например, изменение значения какой-то величины (например, в строке 3 изменяется значение a) или проверка условия.

Остальные столбцы показывают изменения значений величин. В строке 1 записаны начальные значения a и b. После выполнения действия в строке 3 изменилось значение величины a, поэтому в столбце «a» в этой строке мы записываем новое значение a — число 14 и т. д.

Какие значения имеют величины a и b после выполнения действия в строке 5 в табл. 6.2? Как вы рассуждали?

Выполнение алгоритма начинается с шага 1 — проверки условия $a < b$ (строка 2 в таблице). Для начальных значений a и b условие неверно (ложно), поэтому перехода к шагу 4 (строка 9 в таблице) нет. Далее выполняется шаг 2: значение a уменьшается на величину b, получается 14. Записываем это новое значение в третий столбец. Затем переходим к шагу 1 алгоритма (строка 4 в таблице), условие $a < b$ снова ложно, и т. д.

Работа в парах. Придумайте два числа (меньших 20) и предложите своему соседу выполнить для них алгоритм «О» с ручной прокруткой. Проверьте решения друг друга.

При ручной прокрутке алгоритма мы видели, что величины a и b изменяются, им присваиваются новые значения. Такие величины в информатике называются *переменными.*

Переменная — это величина, значение которой можно изменять во время работы алгоритма.

Вместо записи «$a \leftarrow a - b$» часто используют другую форму: «$a := a - b$». Эта операция называется **присваиванием** нового значения переменной.

Языки программирования

Программы для компьютеров составляются на специальных языках, которые называются **языками программирования**. Этим занимаются люди, профессия которых — **программист**.

Любой компьютер (точнее, его процессор) понимает и умеет выполнять только команды в двоичном коде. Но так программы никто не пишет, потому что это очень утомительно, отнимает много времени и приводит к многочисленным ошибкам.

Ближе всего к языку процессора так называемые **языки низкого уровня**, или **языки ассемблера**. В них каждая команда может быть напрямую переведена в соответствующую команду процессора. Это делает специальная программа — **ассемблер** (сборщик).

На языке ассемблера обычно пишут программы, которые должны работать очень быстро и иметь небольшой размер, например драйверы устройств.

К сожалению, у каждого семейства процессоров свой язык ассемблера, поэтому программы, написанные для одного семейства, не подходят для других. Чтобы решить эту проблему, разработали **языки высокого уровня**, которые построены на основе естественного языка (чаще всего английского).

Вот так выглядит алгоритм «О», записанный в машинных кодах, на языке ассемблера и на языке высокого уровня — **школьном алгоритмическом языке**, который мы будем изучать:

Машинные коды:	*Язык ассемблера*:	*Школьный алгоритмический язык*:
10111000000111100000000	mov ax, 15	**цел** a, b
10111011000001000000000	mov bx, 4	a:=15
0011101111000011	m: cmp ax, bx	b:=4
0111110000000100	jl end	**нц пока** a>=b
0010101111000011	sub ax, bx	a:=a-b
1110101111111000	jmp m	**кц**
1100110100100000	end: int 20h	

Попробуйте по приведённым программам определить, какие числа обрабатывает этот алгоритм.

Языки высокого уровня никак не связаны с компьютером, на котором будут выполняться программы. Тогда возникает вопрос — как же разные компьютеры будут понимать программы на этих языках? Решение было найдено: для каждого типа процессоров (и операционных систем) создаётся **транслятор** (переводчик) — программа, которая переводит текст программы, написанной на языке программирования, в двоичные коды нужного процессора.

Первый язык программирования высокого уровня — **Фортран** — был разработан к 1957 году под руководством известного американского специалиста в области информатики Джона Бэкуса. Язык *Фортран* до сегодняшнего времени используется в научных расчётах.

В середине 1970-х годов Деннис Ритчи придумал язык **C** (читается как «си», название третьей буквы английского алфавита), который остаётся одним из самых популярных языков программирования и сегодня. Достаточно сказать, что на нём написаны практически все операционные системы (в том числе *UNIX, Windows, Linux*). На основе языка *C* разработано множество современных языков высокого уровня, в том числе **C++**, **C#** (читается как «Си шарп»), **Java**, **Javascript** и др. Именно эти языки чаще всего используются сегодня для создания программ.

Сейчас очень популярен язык **Python**, появившийся в 1991 году. Он широко используется в таких известных компаниях, как *Google* и *Яндекс*, применяется для программирования игр. Этот язык прост и содержит большую библиотеку алгоритмов для решения многих стандартных задач, с которыми встречаются программисты. Поэтому решение задачи на языке *Python* обычно занимает меньше времени, чем при использовании, например, языков *Фортран* или *C*.

Страницы современных сайтов в Интернете тоже содержат программы. Для этой цели применяют, как правило, языки **PHP** и **Javascript**.

Языки **Prolog** и **LISP** были разработаны в 1970-х годах как языки для решения задач искусственного интеллекта. Программа на **языке логического программирования** *Prolog* — это не алгоритм, а фактически только формулировка задачи. Нужно описать исходные данные, правила, которые используются для решения, и определить цель (что нужно найти). Дальше *Prolog*-машина сама на основе полученных правил решает задачу, т. е. пытается сама построить нужный алгоритм. К сожалению, это удается сделать не всегда. Сейчас эти языки используются редко, в основном в среде научных работников.

Существуют языки высокого уровня, которые придумали специально для обучения программированию. Долгое время в качестве первого языка программирования для начинающих использовали язык **Бейсик** (**BASIC**), но сейчас он вытесняется другими языками.

Один из популярных учебных языков — **Паскаль** — придумал в 1970 году швейцарец Никлаус Вирт. Он назвал новый язык в честь французского физика Блеза Паскаля. Мы начнём изучать язык Паскаль в следующем году.

Со **школьным алгоритмическим языком**, который разработал в 1980-х годах академик Андрей Петрович Ершов, мы уже начали знакомиться: он лежит в основе системы **КуМир**[1]. Алгоритмы, записанные для исполнителей в системе *КуМир*, мы будем называть программами.

Всего существует несколько тысяч языков программирования, но только 20–30 из них широко используются на практике.

Выводы

- Алгоритмы можно записывать в словесной форме, по шагам, в виде блок-схем или на языках программирования.
- Ручная прокрутка (трассировка) — это выполнения алгоритма человеком вручную.
- Переменная — это величина, значение которой можно изменять во время работы алгоритма.
- Программа хранится в памяти компьютера в двоичном коде (как цепочка битов).
- Языки программирования делятся на языки низкого уровня (машинные коды и языки ассемблера) и языки высокого уровня.
- Языки низкого уровня «привязаны» к командам конкретного процессора.
- Языки высокого уровня не зависят от компьютера, на котором выполняется программа.
- Для перевода программ на языке высокого уровня в команды конкретного процессора используют специальные программы — трансляторы.

[1] Это название расшифровывают как «Комплект учебных миров».

Интеллект-карта

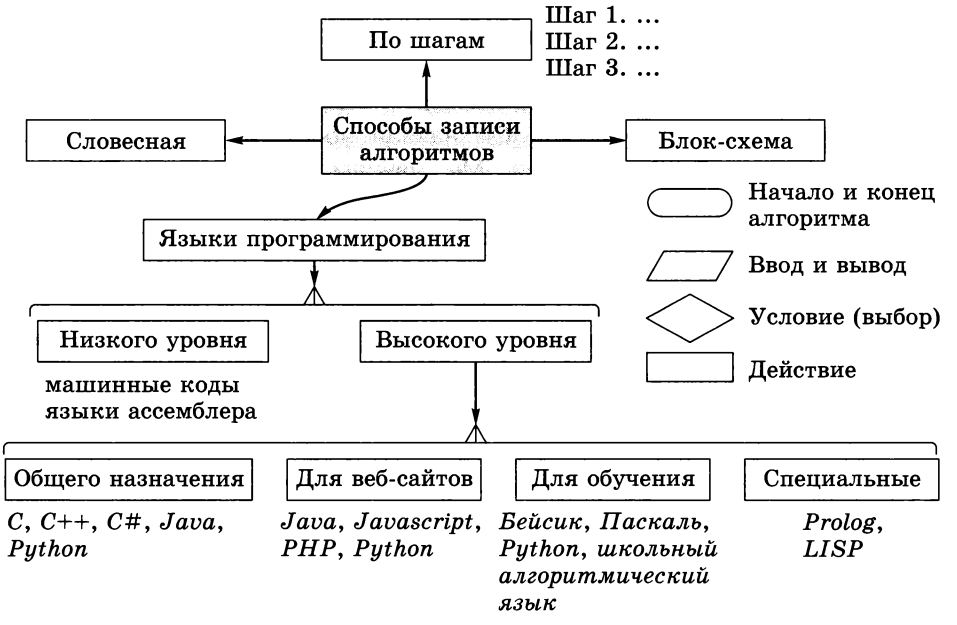


Рис. 6.6

Вопросы и задания

1. Сравните словесную запись алгоритмов в свободной форме, запись по шагам и блок-схему. Какие недостатки вы видите у каждой из этих форм записи?
2. Как вы думаете, почему блок-схемы не используются для записи сложных алгоритмов?
3. Как вы думаете, какими качествами должен обладать программист? Могла бы вам понравиться эта специальность?
4. Зачем нужны языки ассемблера? В чём их недостатки?
5. Правильно ли, на ваш взгляд, считать программу-ассемблер транслятором?
6. Как компьютер понимает программу, написанную на языке высокого уровня?
7. Как вы думаете, почему на практике широко применяются только несколько десятков языков программирования?
8. Какие достоинства и недостатки, на ваш взгляд, имеют языки программирования, разработанные специально для обучения?
9. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Какие бывают языки программирования?»
б) «Языки программирования с русскими командами»
в) «Академик А. П. Ершов»

Практическая работа

Выполните практическую работу № 19 «Алгоритм "О" в Кумире».

§ 31
Примеры исполнителей

Ключевые слова:

- Черепаха
- Удвоитель
- Шифровальщик

Вы уже знакомы с исполнителем Робот, который умеет выполнять четыре команды: `вверх`, `вниз`, `влево` и `вправо`. Теперь давайте познакомимся с другими исполнителями.

Исполнитель Черепаха

Черепаха работает на плоскости, оставляя след при движении. Она умеет выполнять такие команды:

- `вперед n` — передвинуться вперёд на `n` шагов (здесь и дальше *n* — целое число);
- `назад n` — передвинуться назад на `n` шагов;
- `влево n` — повернуться на `n` градусов влево (против часовой стрелки);
- `вправо n` — повернуться на `n` градусов вправо (по часовой стрелке).

Черепаха поворачивается относительно своего **текущего направления** (т. е. того направления, куда она смотрит в данный момент).

В программах для Черепахи мы будем использовать запись

```
повтори n [ ... ]
```

Она означает «повторить n раз все команды, записанные в квадратных скобках». Например, получив программу

```
повтори 3 [вперед 15  вправо 45]
```

управляющий компьютер последовательно выдаст Черепахе 6 команд:

```
вперед 15
вправо 45
вперед 15
вправо 45
вперед 15
вправо 45
```

Обсудите в классе, как Черепаха может нарисовать окружность.

Исполнитель Удвоитель

Исполнитель **Удвоитель** работает с целыми числами. Он умеет выполнять всего две команды:

1. прибавь 1
2. умножь на 2

Программа для Удвоителя — это последовательность номеров команд, записанная без пробелов. Например, программа 12211 означает, что сначала нужно выполнить команду 1, затем — дважды команду 2 и потом дважды команду 1.

Давайте попробуем понять, какие числа можно получить с помощью Удвоителя из некоторого целого числа x. Заметим, что число x может быть положительным, отрицательным или равным нулю. Рассмотрим эти три случая отдельно.

Если $x > 0$, то обе команды приводят только к увеличению числа. В частности, его можно увеличивать каждый раз на 1, т. е. можно получить любое натуральное число, большее, чем x.

Если $x = 0$, то получается то же, что и при положительном x, но при первом применении команды 2 к числу 0 оно не изменится. Вывод: с помощью Удвоителя из числа 0 можно получить любое натуральное число, большее или равное 0.

Если $x < 0$, то команда 1 увеличивает число, а команда 2, применённая к отрицательному числу, уменьшает его (например, из -10 мы получим -20). Поэтому Удвоитель может получить любое целое число, как положительное, так и отрицательное.

Напишите в тетради программу для Удвоителя, с помощью которой он из числа –5 получает число –38.

Пусть теперь нам известна некоторая программа для Удвоителя, например 1212. Выясним, какое значение будет получено из числа x после выполнения этой программы.

На первом шаге (выполнив команду 1) мы получаем число $x+1$, на втором шаге (командой 2) вычислим $2\cdot(x+1)$. Затем снова добавим 1 и умножим на 2, в результате получается число

$$y = 2\cdot(2\cdot(x+1)+1) = 2\cdot(2x+3) = 4x+6.$$

Таким образом, число $y - 6$ должно делиться на 4, другие числа мы получить не могли.

Проверим, можно ли было получить число 36 с помощью программы 1212: $36 - 6 = 30$, а число 30 не делится на 4, поэтому число 36 мы получить никак не могли. А вот число 34 – могли, потому что $34 - 6 = 28$ делится на 4.

Запишите в тетради все числа, меньшие 50, которые можно получить из натуральных чисел с помощью Удвоителя, выполнив программу 1212.

Исполнитель Шифровальщик

Исполнитель Шифровальщик работает с цепочкой символов (символьной строкой), составленной из прописных русских букв. Каждую букву строки он заменяет на следующую букву в алфавите[1]: «А» на «Б», «Б» на «В» и т. д. Последнюю букву алфавита («Я») Шифровальщик заменяет на первую («А»).

Шифровальщик обработал слово, состоящее из N символов. Какую длину имеет слово-результат? Как вы рассуждали?

Выводы

- Для того чтобы ввести исполнитель, необходимо определить его среду (множество обстановок, в которых он может работать) и систему команд исполнителя (СКИ) — перечень команд, которые он может выполнять.

[1] Шифр, который применяет Шифровальщик, называется шифром Цезаря.

Интеллект-карта

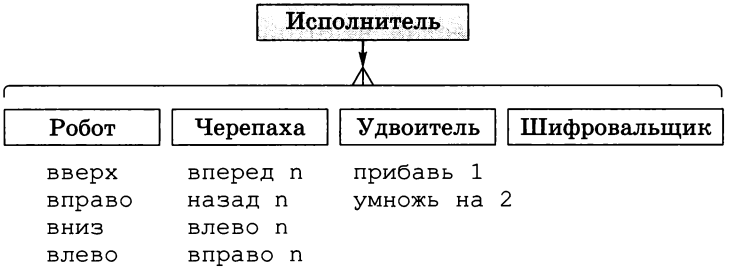


Рис. 6.7

Вопросы и задания

Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Исполнитель Кузнечик»
б) «Исполнитель Паркетчик»
в) «Шифр Цезаря»

Практическая работа

Выполните практическую работу № 20 «Программное управление Черепахой».

§ 32
Оптимальные программы

***Ключевые слова*:**

- оптимальная программа
- дерево вариантов

Задача. В системе команд исполнителя Удвоитель всего две команды:

1. `прибавь 1`
2. `умножь на 2`

Составьте самую короткую программу для Удвоителя, которая преобразует число 6 в 28.

Попытайтесь решить задачу самостоятельно. Сравните своё решение с решениями одноклассников. У кого получилось короче?

Возможно, в этой простой задаче вы сразу сообразили, какой будет самая короткая программа. Но в более сложных случаях это не так просто. Кроме того, нужно доказать, что это действительно самая короткая программа. Поэтому рассмотрим общий способ решения.

Самую короткую программу можно назвать *оптимальной по длине*.

Оптимальная программа — это самая лучшая программа по какому-то показателю.

В этой задаче нас интересует программа, содержащая меньше всего команд.

За одну команду мы можем из начального числа 6 получить только число 7 (командой 1) или число 12 (командой 2) — рис. 6.8.

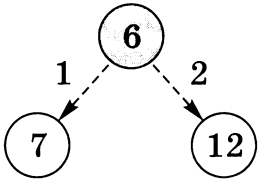
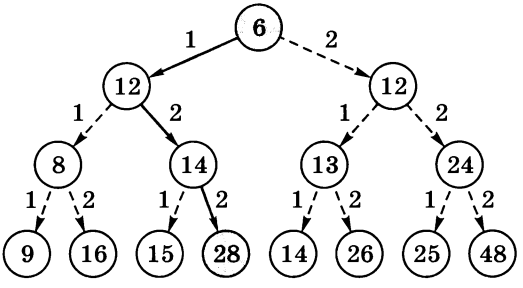


Рис. 6.8

Выясните, какие числа можно получить с помощью Удвоителя из начального числа 6 за два шага.

Проверим все возможные трёхшаговые программы. Мы обнаружим, что одна из программ, 122 (она выделена голубыми стрелками на рис. 6.9), приведёт к нужному нам значению 28.



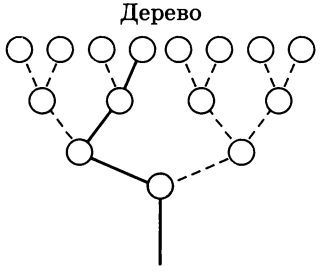


Рис. 6.9

До этого вы убедились, что за одну и даже за две команды число 28 получить невозможно, поэтому программа 122 из трёх команд — оптимальная, т. е. самая короткая.

Как видно из рис. 6.9, нет других программ из трёх команд, которые приводят к числу 28, поэтому оптимальная программа единственная. Все другие программы, с помощью которых можно получить число 28 из 6, длиннее, чем эта.

Схема, которую мы построили, называется **деревом возможных вариантов**. Действительно, мы рассмотрели *все возможные* программы, состоящие из одного, двух и трёх шагов. Если полученный рисунок развернуть «вверх ногами», он напоминает дерево (рис. 6.9, справа).

Построение полного дерева вариантов при большой длине программы (например, 5 и больше команд) — это очень утомительная работа. Во многих задачах бывает проще двигаться не от начального числа к результату, а наоборот.

Можно ли составить программу, которая на последнем шаге получает число 28 командой 1? Командой 2? Что можно сказать про число 27?

Возьмём конечное число 28 и подумаем, какой последней командой мы могли его получить. Так как 28 делится на 2, это могла быть как команда 1 (из 27 получаем 28), так и команда 2 (из 14 также получаем 28). Ни одно из этих чисел не совпадает с начальным (6), поэтому одной командой задачу не решить.

Делаем второй шаг — проверяем, из каких чисел мы могли получить 27 и 14. Число 27 не делится на 2, поэтому оно могло быть получено только командой 1 (из 26), а число 14 можно получить из 13 или из 7 (рис. 6.10).

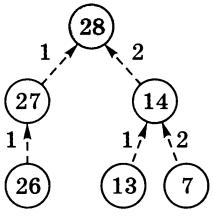


Рис. 6.10

До начального числа 6 снова добраться не удалось, поэтому решений из двух команд не существует. Делаем третий шаг (рис. 6.11).

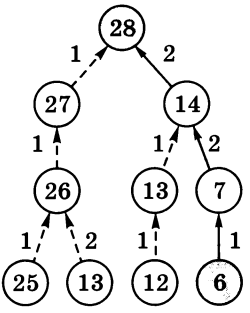


Рис. 6.11

Для того чтобы получить самую короткую программу, нужно пройти по стрелкам от начального числа к конечному, выписывая встречающиеся номера команд. Для нашей задачи получаем тот же ответ, что и раньше, — 122 (путь выделен голубыми стрелками).

Обратите внимание, что дерево при движении в обратном направлении, от конечного числа к начальному, получилось меньше. Нам удалось найти решение быстрее, рассматривая меньше вариантов.

Объясните, почему в некоторых случаях в дереве на рис. 6.11 не было разделения на две ветки.

Выводы

- Оптимальная программа — это самая лучшая программа по какому-то показателю, например содержащая меньше всего команд.
- Для того чтобы найти оптимальную программу для исполнителя, можно сначала рассмотреть все возможные результаты его работы за один шаг, затем — за два шага и т. д., пока на каком-то шаге не будет получен желаемый результат. Найденная программа для перехода из начального состояния в конечное будет оптимальной по длине (самой короткой).
- Схема, показывающая все возможные результаты работы исполнителя, называется деревом возможных вариантов.
- Если в СКИ исполнителя есть необратимые команды (например, любое целое число можно умножить на 2, но не любое можно разделить на 2 без остатка), лучше строить дерево возможных вариантов от конечного состояния к начальному.

Интеллект-карта

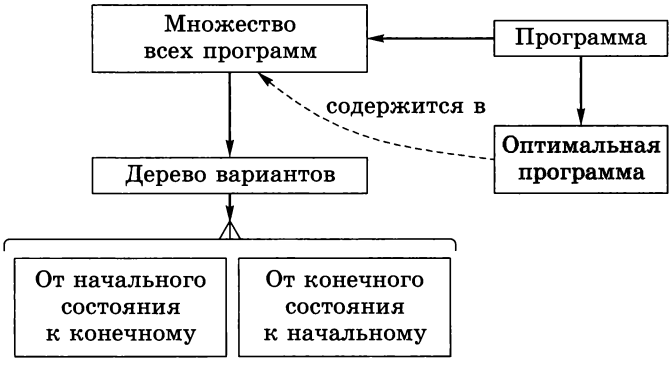


Рис. 6.12

Вопросы и задания

1. Может ли быть так, что задача для Удвоителя решается с помощью нескольких различных алгоритмов? Если да, приведите примеры.
2. Как можно доказать, что построенная программа для Удвоителя действительно самая короткая?
3. Какие числа можно (нельзя) получить из натурального числа N с помощью Удвоителя? Из нуля? Из отрицательного числа?
4. Как быстро построить самую короткую программу для получения некоторого числа N из нуля с помощью Удвоителя? Когда эта задача не имеет решений?
5. Исполнитель Калькулятор имеет команды
   1. `прибавь 1`
   2. `раздели на 2`

   Нужно составить самую короткую программу для Калькулятора, с помощью которой из числа *a* можно получить число *b*. Как лучше перебирать варианты программ, от начального числа к конечному или наоборот? Почему?
6. Выполните по указанию учителя задания в рабочей тетради.

§ 33
Линейные алгоритмы

Ключевые слова:

- линейный алгоритм
- программа
- алгоритмический язык
- служебные слова языка
- синтаксические ошибки
- логические ошибки
- отказы

Что такое линейный алгоритм?

В линейном алгоритме команды выполняются в том порядке, в котором они записаны.

В этом параграфе мы научимся составлять линейные алгоритмы для Робота. Кроме тех команд Робота, с которыми вы уже знакомы, в его СКИ есть команда

```
закрасить
```

Выполняя эту команду, Робот закрашивает клетку, в которой он стоит.

Клетки, которые нужно закрасить по заданию, мы будем обозначать точкой (рис. 6.13).

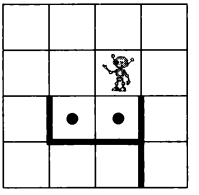


Рис. 6.13

Для того чтобы Робот смог выполнить программу, нужно оформить её так, чтобы она была понятна системе *КуМир*. Шаблон программы для решения задачи на рис. 6.13 выглядит так:

```
использовать Робот
алг Переход
нач
  ...
кон
```

В первой строке программы мы подключаем исполнителя Робот. Программа на алгоритмическом языке[1)] начинается словом **алг**, за которым записывают имя алгоритма. Имя выбирают так, чтобы можно было легко понять, для чего служит алгоритм.

Основная часть («тело») алгоритма начинается словом **нач** (сокращения от слова «*начало*») и заканчивается словом **кон** (сокращение от слова «*конец*»). Между словами **нач** и **кон** (вместо многоточия) нужно добавить команды алгоритма.

Слова **алг**, **нач** и **кон** — это **служебные (зарезервированные)** слова алгоритмического языка. Они всегда имеют единственное значение, которое задали им разработчики языка.

Служебные слова — это специальные слова языка программирования, имеющие единственное заранее определённое значение.

Какие могут быть ошибки?

При написании программ неизбежны ошибки. Это связано с тем, что возможности человека ограничены, и даже хороший программист редко может написать сразу без ошибок программу длиннее, чем 40–50 строк. Бояться ошибок не нужно, нужно научиться обнаруживать и исправлять их.

Есть два типа ошибок:

1) **синтаксические ошибки**, когда исполнитель не понимает команду, потому что она неверно записана;
2) **логические ошибки**, которые исполнитель обнаружить не может: он понимает все команды, но результат его работы не совпадает с тем, который нужен программисту.

В результате логической ошибки может произойти **отказ** — ситуация, когда исполнитель понимает команду, но в данных условиях не может её выполнить. Например, исполнитель Калькулятор не может делить на ноль, а Робот не может пройти через стену.

Для того чтобы обнаружить и исправить логические ошибки, можно выполнять программу в **пошаговом режиме** (в *КуМире* для этого служит клавиша *F8*). Это означает, что мы выполняем не все команды сразу, а по одной, останавливая исполнителя после каждой выполненной команды. Так мы сможем определить, на каком шаге исполнитель сделал не то, что мы предполагали.

[1)] Для краткости мы будем называть школьный алгоритмический язык просто «алгоритмический язык».

Вычислительные задачи

Как вы знаете, компьютеры были изобретены для того, чтобы ускорить вычисления. Посмотрим, как решать простую вычислительную задачу в системе *КуМир*.

Задача. Сколько километров проехал автомобиль за 2 часа, если его средняя скорость равна 60 км/ч?

Конечно, одну такую задачу несложно решить устно или на калькуляторе, но для быстрого решения большого количества задач одного типа с разными исходными данными лучше разработать компьютерную программу. Для этого алгоритм решения нужно записать «в буквах». Обозначим время через *t*, а скорость — через *v*. Результат — пройденное расстояние — обозначим буквой *S*. Программа должна вычислить результат по исходным данным (рис. 6.14).

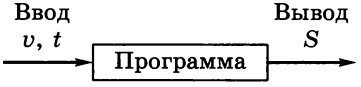


Рис. 6.14

Алгоритм решения задачи можно записать так:

Вход: v, t.
Шаг 1. S := v · t.
Результат: S.

Здесь нет ни одной команды перехода, которая направляет исполнителя на другой шаг. Поэтому порядок действий всегда одинаковый, какие бы исходные данные мы ни вводили.

Теперь остаётся оформить программу так, чтобы её можно было выполнить в системе *КуМир*. Она может выглядеть, например, так:

```
алг Путь
нач
  вещ v, t, S
  вывод "Введите скорость: "
  ввод v
  вывод "Введите время: "
  ввод t
  S:=v*t
  вывод "Расстояние: ", S
кон
```

Обратите внимание, что здесь нет никаких чисел, программа решает задачу «в буквах». Буквами (точнее, **именами**, которые могут состоять из нескольких сиимволов) обозначены изменяемые величины — **переменные**.

Команда

```
вещ v, t, S
```

объявляет переменные, т. е. сообщает компьютеру, что мы будем использовать переменные с именами `v`, `t` и `S`. Все они — вещественные, т. е. могут принимать не только целые, но и дробные значения, об этом говорит слово **вещ** в начале строки.

Команда `вывод` выводит данные в поле ввода и вывода программы (в нижней части окна). Это может быть просто строка символов, например

```
вывод "Введите скорость: "
```

или список того, что нужно вывести:

```
вывод "Расстояние: ", S
```

Здесь в списке два элемента: текст «Расстояние» и значение переменной величины *s*, они разделены запятой.

С помощью команды `ввод` пользователь (человек, работающий с программой) вводит исходные данные — скорость и время. Например, после вывода сообщения «Введите скорость: » компьютер выполняет команду

```
ввод v
```

Она означает «ждать, когда пользователь введёт число, присвоить это число переменной *v*». В поле ввода и вывода мигает курсор — это означает, что нужно ввести число и нажать клавишу *Enter*.

Выводы

- В линейном алгоритме команды выполняются в том же порядке, в котором они записаны.
- Служебные слова — это специальные слова языка программирования, имеющие единственное заранее определённое значение.
- Различают два типа ошибок в программах:
  - — синтаксические ошибки (ошибки типа «не понимаю») — неверное написание команды;
  - — логические ошибки — это ошибки, которые исполнитель обнаружить не может: он понимает все команды, выполняет их, но результат его работы не совпадает с тем, который нужен.
- В результате логической ошибки может произойти отказ — ситуация, когда исполнитель понимает команду, но не может её выполнить в данных условиях.

Интеллект-карта

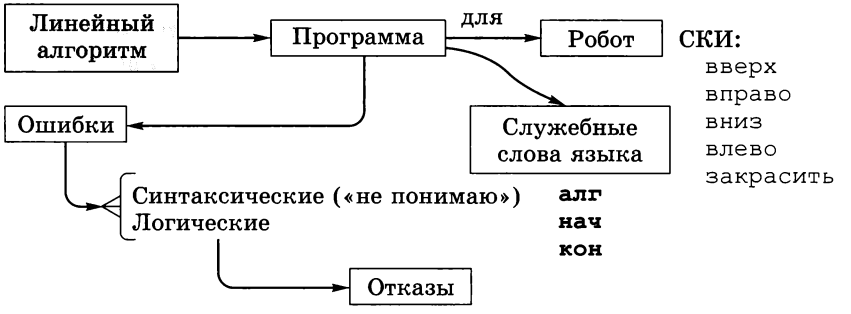


Рис. 6.15

Вопросы и задания

1. Верно ли, что для каждой задачи существует единственный алгоритм решения? Ответ обоснуйте.
2. Сталкивались ли вы уже с алгоритмами, которые нельзя назвать линейными?
3. Как вы думаете, какие задачи невозможно решить с помощью линейных алгоритмов?
4. Как можно сравнить два различных алгоритма решения одной и той же задачи? Как выбрать лучший их них?
5. Два друга по-разному ищут ошибки в программах. Кирилл, написав программу, сразу запускает её для того, чтобы транслятор обнаружил все синтаксические ошибки. Даниил же сначала внимательно изучает текст программы и пытается найти ошибки сам, а потом уже запускает её на выполнение. Чем хорош каждый из методов?
6. Чем различаются синтаксические и логические ошибки?
7. Как можно искать логическую ошибку в программе?
8. К какому типу ошибок относится случай, когда выполнение программы не останавливается (программа зацикливается)?
9. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Роботы вокруг нас»

Практические работы

Выполните практические работы:
№ 21 «Линейные программы для Робота»;
№ 22 «Вычислительные задачи».

§ 34
Вспомогательные алгоритмы

Ключевые слова:

- вспомогательный алгоритм
- процедура
- вызов процедуры
- возврат из процедуры
- метод последовательного уточнения

Что такое вспомогательный алгоритм?

В этой задаче Роботу нужно закрасить по три клетки в двух совершенно одинаковых помещениях, имеющих форму сапога (рис. 6.16).

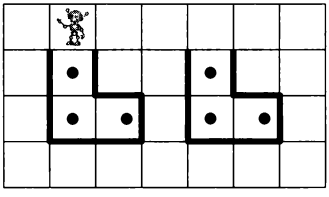


Рис. 6.16

Конечно, было бы хорошо, если бы у Робота была команда (например, `Сапог`), выполняя которую, Робот обрабатывает такое помещение и возвращается обратно. Тогда бы мы написали такую программу:

```
алг Два сапога
нач
  Сапог
  вправо; вправо; вправо
  Сапог
кон
```

Для сокращения длины программы мы записали несколько команд в одной строке, разделив их точкой с запятой.

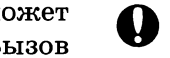
Обсудите в классе, какая ошибка возникнет, если запустить эту программу. Почему?

Но в системе команд Робота нет команды Сапог! Однако всё можно исправить, если объяснить Роботу, что ему нужно делать по команде Сапог. Для этого достаточно записать в конце программы (после служебного слова **кон**) вспомогательный алгоритм:

```
алг Сапог
нач
  вниз; закрасить
  вниз; закрасить
  вправо; закрасить
  влево; вверх; вверх
кон
```

Теперь программа запускается и работает.

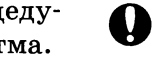
Вспомогательный алгоритм решает отдельную задачу и может быть использован при решении более сложных задач. Вызов вспомогательного алгоритма можно использовать так же, как команды из СКИ исполнителя.



Вспомогательный алгоритм часто называют **процедурой**.

Давайте разберёмся, что будет происходить при запуске этой программы. Алгоритм, записанный в самом начале программы (у нас это алгоритм Два сапога), называется **основной программой**. Компьютер выполняет основную программу. В первой же строке он встречает неизвестную команду Сапог, которая не входит в СКИ исполнителя Робот. Значит, это **вызов вспомогательного алгоритма** (*процедуры*).

Для того чтобы вызвать вспомогательный алгоритм (процедуру), нужно записать его название в теле другого алгоритма.



Если в основной программе нет вызова процедуры, эта процедура не выполнится ни разу, хотя и будет записана в тексте программы.

Вспомогательный алгоритм выполняется только тогда, когда он вызван.

Встретив вызов процедуры Сапог, компьютер ищет процедуру с таким названием ниже основной программы. Если она найдена, то выполняются все команды, записанные в теле этого вспомогательного алгоритма. Затем происходит *возврат из процедуры* — передача управления на ту команду в основной программе, которая записана сразу после команды Сапог.

После завершения работы процедуры управление передаётся обратно, к следующей команде вызывающей программы.

Для того чтобы увидеть, как передаётся управление, в КуМире можно использовать пошаговый режим с входом в процедуры (клавиша *F7*).

Конечно, эту задачу можно было бы решить и без процедуры, но решение с процедурой получилось короче. Нам не пришлось писать два раза одинаковые команды. Кроме того, если в какой-то совершенно другой задаче Роботу нужно будет закрашивать клетки в таком же помещении, мы сможем использовать готовый вспомогательный алгоритм Сапог.

Два метода составления программ

Мы только что использовали приём, который называется **проектированием «сверху вниз»** или методом **последовательного уточнения**. Задача разбивается на части (подзадачи), и сначала записывается основной алгоритм. Он использует вызовы ещё не написанных вспомогательных алгоритмов, каждый из которых решает свою подзадачу. Затем пишем вспомогательные алгоритмы, если нужно, так же разбивая их на части.

Есть и другой метод, который называется **проектированием «снизу вверх»**: сначала пишем все вспомогательные алгоритмы, а потом собираем из них, как из кубиков, основной алгоритм. Мы решим этим методом ещё одну задачу для Робота (рис. 6.17).

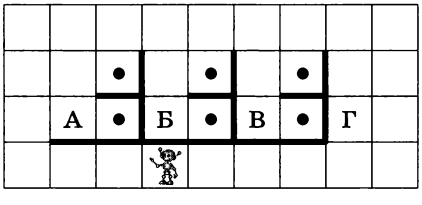


Рис. 6.17

Имеет ли смысл использовать здесь процедуру? Обоснуйте ответ.

На рисунке 6.17 мы видим три одинаковые группы из двух соседних клеток, разделённых стенкой. Поэтому можно оформить обработку такой пары клеток в виде процедуры. Выделим отдельную подзадачу, в которой Робот должен закрасить две клетки и прийти в клетку Б (рис. 6.18).

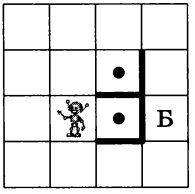


Рис. 6.18

Оформим решение подзадачи как вспомогательный алгоритм:

```
алг Пара
нач
  вправо; закрасить
  влево; вверх; вправо; закрасить
  вверх; вправо; вниз; вниз
кон
```

Теперь вернёмся к полной задаче на рис. 6.17. Видим, что сразу применить процедуру Пара нельзя, Робота нужно сначала привести в клетку А. Затем можно вызывать процедуру, причём после того, как она закончит работу, Робот остановится в клетке Б, так что можно снова вызывать нашу процедуру. Основная программа будет выглядеть так:

```
использовать Робот
алг ТриПары
нач
  влево; влево; вверх; вправо
  Пара; Пара; Пара
кон
```

Конечно, после неё нужно записать расшифровку процедуры.

Применяя проектирование «снизу вверх», мы строим решение основной задачи на основе уже готовых решений более мелких задач. Так же, например, в математике доказательство новой теоремы использует уже известные результаты.

Выводы

- Вспомогательный алгоритм решает отдельную задачу и может быть использован при решении более сложных задач. Вызов вспомогательного алгоритма можно использовать так же, как команды из СКИ исполнителя.
- Вспомогательные алгоритмы часто называют процедурами.
- Вспомогательный алгоритм выполняется только тогда, когда он вызван.
- Для того чтобы вызвать вспомогательный алгоритм (процедуру), нужно записать его название в тексте другого алгоритма.
- При вызове вспомогательного алгоритма выполняются все входящие в него команды.
- После завершения работы вспомогательного алгоритма управление передаётся обратно, к следующей команде вызывающей программы.
- При проектировании программы «сверху вниз» (методе последовательного уточнения) задача разбивается на подзадачи, решение каждой из них оформляется в виде процедуры.
- При проектировании программы «снизу вверх» программа собирается из заранее написанных процедур, как из кубиков.

Интеллект-карта

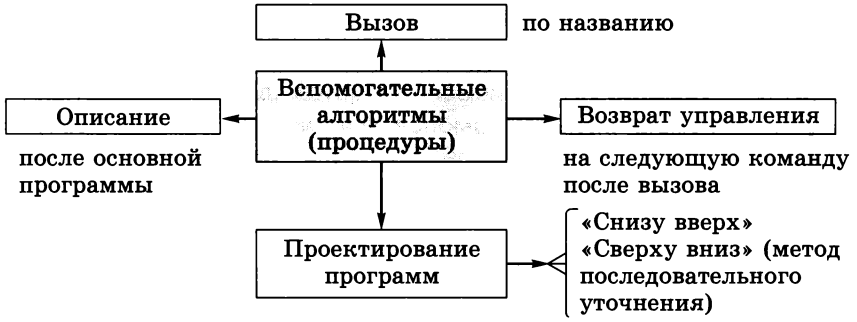


Рис. 6.19

Вопросы и задания

1. Какие задачи легче решаются с использованием вспомогательных алгоритмов?

2. Можно ли использовать процедуры, если в программе нет повторяющихся действий? Зачем это может быть нужно?
3. В каких ситуациях вы бы не рекомендовали использовать вспомогательные алгоритмы?
4. Что произойдёт, если исполнитель не обнаружит расшифровки новой команды?
5. Процедура есть в тексте программы, но не срабатывает. В чём может быть причина?
6. Куда передаётся управление, если процедура вызывается в самом конце основной программы (за ним нет других команд)?
7. Выполните по указанию учителя задания в рабочей тетради.

Практическая работа

Выполните практическую работу № 23 «Вспомогательные алгоритмы».

§ 35
Циклические алгоритмы

Ключевые слова:

- цикл
- циклический алгоритм
- тело цикла
- вложенный цикл
- комментарий

Что такое циклический алгоритм?

Задача. Роботу нужно закрасить все отмеченные клетки.

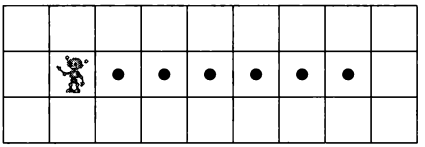


Рис. 6.20

Как записать решение этой задачи в виде линейного алгоритма? Как изменится алгоритм, если Роботу нужно закрасить не 6, а 1006 клеток подряд? Что вам хотелось бы улучшить в записи алгоритма?

Чтобы не повторять много раз одинаковые строчки в тексте программы, в языки программирования ввели специальные команды, которые сообщают исполнителю, сколько раз нужно повторить те или иные действия. Такие команды называют **циклами**, а алгоритмы — циклическими.

Циклический алгоритм — это алгоритм, в котором некоторая последовательность действий выполняется несколько раз.

В алгоритмическом языке есть цикл «*N* раз», который в нашем примере можно записать так:

```
нц 6 раз
  вправо
  закрасить
кц
```

Здесь **нц** обозначает «начало цикла», а **кц** — конец цикла.

Тело цикла — это команды, которые выполняются несколько раз.

В тело цикла не входят команды, которые служат для организации работы самого цикла, например проверка количества уже сделанных шагов цикла (или количества шагов, которые осталось сделать). В нашем случае тело цикла состоит из двух команд: `вправо` и `закрасить`.

Если нужно обработать не 6 клеток, а 6000, достаточно изменить число повторений после слова **нц**. При этом длина записи алгоритма не изменится, а количество выполняемых команд вырастет в 1000 раз! Таким образом, с помощью короткого алгоритма можно описать длинную последовательность команд.

Блок-схема циклического алгоритма содержит «петлю» (контур) — возврат к предыдущим командам. На рис. 6.21 петля показана штриховой линией.

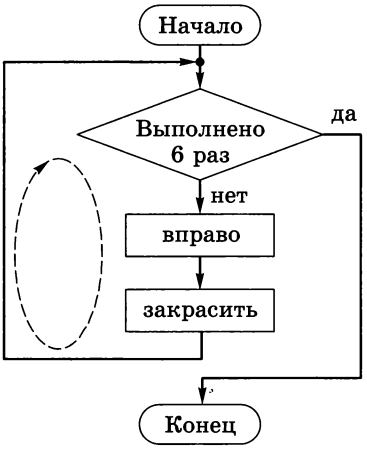


Рис. 6.21

Выбор начального положения

Очень важно правильно выбрать начальное положение Робота — клетку, в которой он стоит перед началом цикла.

В рассмотренной выше задаче выгодное начальное положение — это клетка Г (рис. 6.22), потому что Робот может сразу начать выполнение цикла. Можно также начать с клетки Д, при этом команды `вправо` и `закрась` в теле цикла нужно поменять местами. Если Робот стоит в другом месте, его лучше сначала привести в одну из этих клеток.

А		Б		В				
	Г	Д	•	•	•	•	•	
Е		Ж		З				

Рис. 6.22

Роботу нужно закрасить все клетки и прийти в клетку Б (База) — рис. 6.23. Выберите наилучшее положение Робота для начала цикла и напишите программу для решения задачи. Сравните свой вариант с решениями одноклассников. Как можно определить, какой вариант лучше?

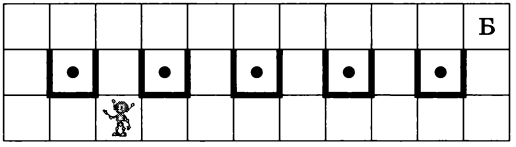


Рис. 6.23

Вложенные циклы

Рассмотрим задачу, в которой Роботу нужно закрасить поле размером 6×4 клетки (рис. 6.24).

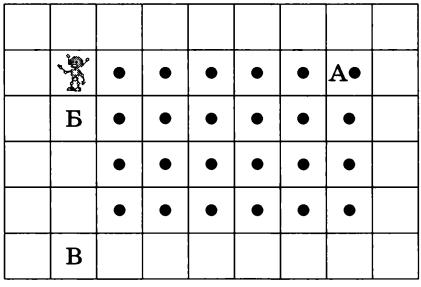


Рис. 6.24

Нам уже несложно написать цикл, который закрашивает верхний ряд:

```
| закрасить один ряд
нц 6 раз
  вправо
  закрасить
кц
```

Символ «|» начинает комментарий.

Комментарии — это пояснения для человека внутри текста программы.

Транслятор пропускает все комментарии, поэтому работа программы не изменится, если комментарии убрать. Комментарии помогают разобраться в том, как работает программа. Это очень помогает, если программа написана давно, и вы уже всё забыли, или её написал другой человек.

После выполнения цикла Робот остановится в клетке А. Для того чтобы такой же цикл можно было использовать ещё раз, нужно перевести Робота в клетку Б, например, так:

```
| перейти к следующему ряду
вниз
нц 6 раз влево кц
```

Предложите другой вариант перевода Робота из клетки А в клетку Б. Какой из них лучше? Как вы рассуждали?

У нас всего 4 ряда, поэтому можно использовать цикл

```
нц 4 раз
  | закрасить один ряд
  нц 6 раз
    вправо
    закрасить
  кц
  | перейти к следующему ряду
  вниз
  нц 6 раз влево кц
кц
```

Здесь фоном выделены два цикла, которые оказались внутри другого цикла. Такие циклы называются вложенными.

Вложенный цикл — это цикл, находящийся в теле другого цикла.

Что сделает Робот, если в последней программе заменить число 4 на 6 и число 6 на 15?
В какой клетке остановится Робот после выполнения программы?

Как решить задачу, показанную на рис. 6.24, используя процедуры `Ряд` и `Переход` (без вложенных циклов)? Сравните два решения.

Выводы

- Циклический алгоритм — это алгоритм, в котором некоторая последовательность действий (тело цикла) выполняется несколько раз.
- Цикл «N раз» выполняется заданное число раз.
- Комментарии — это пояснения для человека внутри текста программы. Комментарии не обрабатываются транслятором.
- Вложенный цикл — это цикл, находящийся в теле другого цикла.

Интеллект-карта

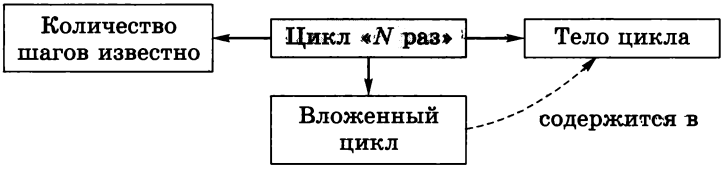


Рис. 6.25

Вопросы и задания

1. Как, по-вашему, Робот может определить, что он выполнил цикл нужно количество раз?
2. Как по блок-схеме определить, что алгоритм содержит цикл?
3. Юный программист Семён говорит, что никогда не пишет комментариев в программах, потому что это лишняя работа, а он и так всё помнит. Согласны ли вы с ним?
4. Выполните по указанию учителя задания в рабочей тетради.

Практические работы

Выполните практические работы:
№ 24 «Циклические алгоритмы»;
№ 25 «Вложенные циклы».

§ 36 Переменные

Ключевые слова:

- переменная
- присваивание
- объявление переменной
- процедура
- параметр

Зачем нужны переменные?

Попробуем решить задачу, показанную на рис. 6.26.

Сравните эту задачу с задачей, показанной на рис. 6.25. Чем они различаются?

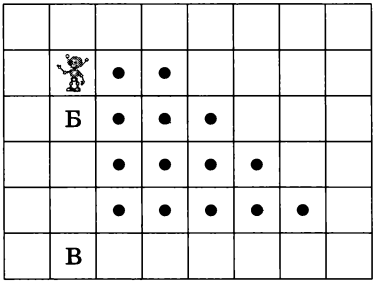


Рис. 6.26

Длины всех рядов тут разные: первый горизонтальный ряд состоит из двух клеток, а длина каждого следующего на одну клетку больше, чем предыдущего.

Обозначим длину очередного ряда буквой N. Тогда алгоритм закраски очередного ряда и возврата к началу следующего ряда выглядит так:

```
нц N раз
  вправо; закрасить
кц
вниз
нц N раз влево кц
```

Величина N должна изменяться во время работы программы. Как вы уже знаете, такая величина называется **переменной**.

Каково начальное значение переменной N? Как она должна изменяться при переходе к следующему ряду?

До начала основного цикла нужно присвоить этой переменной значение 2 (это длина верхнего ряда):

```
N:=2
```

Символы «:=» — это команда (**оператор**) **присваивания**, она присваивает переменной N значение, которое записано в правой части.

После обработки очередного ряда и возврата назад нужно увеличить значение N на 1, т. е. заменить N на N+1. Для этого тоже используется оператор присваивания:

```
N:=N+1
```

В результате получаем такую программу с вложенными циклами:

```
цел N
N:=2
нц 4 раз
   нц N раз
    вправо; закрасить
   кц
   вниз
   нц N раз влево кц
   N:=N+1
кц
```

Строка

```
цел N
```

называется **объявлением переменной**. Этой командой мы сказали, что в программе будет использована переменная, которая имеет имя *N* и может принимать только *целочисленные* значения (служебное слово **цел**).

Зачем нужно объявление переменной? При объявлении мы определяем:

- какие значения может принимать эта переменная (целые числа, десятичные дроби, строки символов и др.);
- какие операции можно выполнять с переменной;
- сколько места в памяти компьютера нужно выделить для хранения значения.

В некоторых языках программирования, например в языке *Python*, объявлять переменные не нужно. С одной стороны, это облегчает составление программы, но с другой — может привести к ошибкам, которые очень сложно обнаружить.

Процедуры с параметрами

Какое из двух высказываний по поводу вспомогательных алгоритмов (процедур), верно?

а) Процедура всегда выполняет одни и те же действия.

б) Процедура может выполнять разные действия.

Часто бывает нужно несколько раз выполнить похожие, но чем-то отличающиеся действия. В этом случае тоже можно составить процедуру, но более сложного типа.

Рассмотрим задачу, показанную на рис. 6.27.

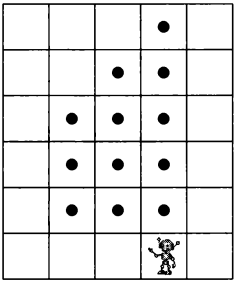


Рис. 6.27

Если бы все ряды были одинаковой длины (скажем, 4 клетки), мы могли бы написать процедуру (вспомогательный алгоритм) так:

```
алг Ряд
нач
  нц 4 раз
    вверх; закрасить
  кц
кон
```

Но длина ряда меняется, поэтому нужно сделать её *переменной*, обозначив каким-то именем. Кроме того, надо как-то передать значение длины ряда из основной программы в процедуру (иначе как исполнитель узнает, какая длина ряда нужна именно сейчас?). При вызове мы хотели бы записать, например, так:

```
Ряд (4)
```

Чтобы такая запись сработала, в заголовке процедуры нужно указать, что она принимает *параметр* — значение, от которого зависит работа алгоритма. В нашем случае параметр — это количество клеток, которые нужно закрасить:

```
алг Ряд (цел N)
нач
  нц N раз
    вверх; закрасить
  кц
кон
```

В заголовке процедуры

```
Ряд (цел N)
```

записано, что процедуре нужно передать (в скобках) целое число. Это значение будет записано во внутреннюю (**локальную**) переменную с именем `N`. Эту процедуру «знает» только процедура `Ряд`.

Параметры — это данные, которые передаются в процедуру. Каждый параметр имеет имя и тип.

Закончите программу для решения задачи, показанной на рис. 6.27.

Включите в процедуру команды для перехода в начало следующего ряда, так чтобы основную программу можно было записать так:

```
алг Трапеция2
нач
  Ряд(5)
  Ряд(4)
  Ряд(3)
кон
```

Числа, который записаны в скобках при вызове процедуры `Ряд`, — это **фактические значения параметров**, которые также называют **аргументами**.

В чём разница между результатами работы программ `Трапеция` и `Трапеция2`? Сравните:
а) какие клетки будут закрашены;
б) где остановится Робот.

Выводы

- Переменная — это величина, которая изменяется во время выполнения алгоритма (программы).
- Во многих языках программирования переменные нужно объявлять. Объявление нужно для того, чтобы определить, какие данные можно хранить в переменной; какие операции можно с ней выполнять, сколько памяти выделить для хранения значения.
- Параметры — это данные, которые передаются в процедуру. Каждый параметр имеет имя и тип.
- Процедуры с параметрами используются для того, чтобы одна процедура могла выполнять похожие, но чем-то отличающиеся действия.
- Фактические значения параметров, которые передаются в процедуру, называют аргументами.

Интеллект-карта

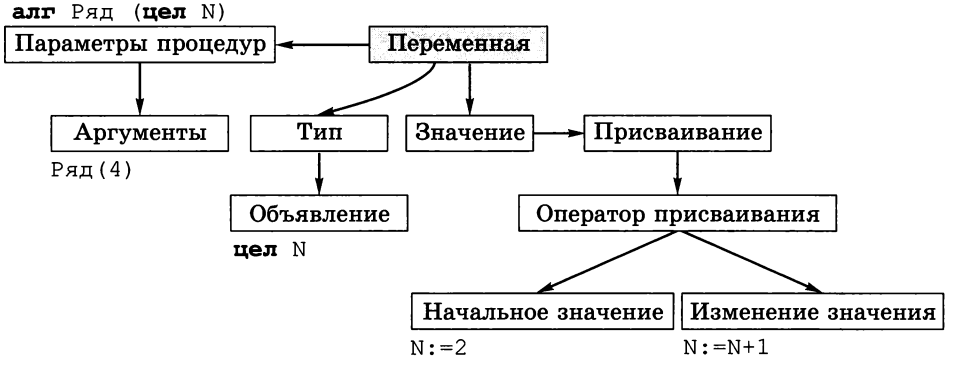


Рис. 6.28

Вопросы и задания

1. Можно ли решить задачу, показанную на рис. 6.26, не используя переменные?
2. Как, на ваш взгляд, компьютер может выполнить цикл «*N* раз» (как он запомнит, сколько раз уже выполнил тело цикла)?
3. Что означает запись `N:=N+N`?
4. Зачем нужно объявлять переменные?
5. Сколько различных процедур без параметров можно было бы написать для решения задачи, показанной на рис. 6.27?
6. Что значит передать данные в процедуру? Зачем это нужно?
7. Как по записи процедуры выяснить, принимает ли она параметры, и если да, то какие именно?
8. Выполните по указанию учителя задания в рабочей тетради.

Практические работы

Выполните практические работы:

№ 26 «Переменные»;

№ 27 «Процедуры с параметрами».

§ 37
Циклы с условием

Ключевые слова:

- условие
- цикл с условием
- логические команды
- обратная связь
- зацикливание
- вложенные циклы

Что такое цикл с условием?

Вспомним алгоритм вычисления остатка от деления, о котором мы говорили в начале § 30 («алгоритм "О"»):

Вход: два натуральных числа, a и b.
Шаг 1. Если $a < b$, перейти к шагу 4.
Шаг 2. Заменить a на $a - b$.
Шаг 3. Перейти к шагу 1.
Шаг 4. Стоп.
Результат: значение a.

Можно ли считать этот алгоритм циклическим? Как вы рассуждали? Можно ли сразу сказать, сколько раз выполнится цикл? От чего это зависит?
Можно ли запрограммировать такой алгоритм с помощью цикла «N раз»? Почему?

Определите условия, при которых: а) цикл закончится; б) цикл продолжает выполняться.

Цикл с условием — это цикл, который выполняется до тех пор, пока некоторое условие не станет ложным. Количество шагов такого цикла определяется исходными данными.

Логические команды Робота

Задача. Роботу нужно подойти к стене, которая находится слева от него, но неизвестно, на каком расстоянии (рис. 6.29).

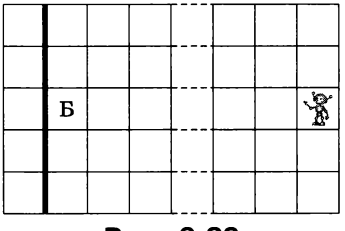


Рис. 6.29

Как бы действовали вы на месте Робота? Какое условие нужно проверить Роботу, чтобы определить, нужно ли двигаться дальше?

Человек может увидеть стену или почувствовать её, дотронувшись рукой. У Робота тоже есть «органы чувств» — датчики. В систему команд Робота входят логические команды:

сверху стена	сверху свободно
справа стена	справа свободно
снизу стена	снизу свободно
слева стена	слева свободно

Это команды-запросы, на которые Робот (с помощью своих датчиков) отвечает «да» или «нет».

Логическая команда — это запрос, на который исполнитель отвечает «да» или «нет».

С помощью логических команд (команд-запросов) Робот может проверять условия, связанные с окружающей обстановкой. Например, для того чтобы определить, можно ли двигаться в каком-то направлении, нужно выяснить, нет ли там стены.

Как решить задачу, показанную на рис. 6.29, если управлять Роботом с пульта? Робот может в любой момент получить ответ на любую логическую команду из его СКИ.

Логические команды также называют *командами обратной связи*, потому что информация передаётся в обратном направлении — не от управляющего устройства к исполнителю, а наоборот. С помощью этих команд Робот получает информацию об окружающей среде.

Обратная связь — это данные, которые передаются от датчиков к управляющему устройству.

Решение задачи на алгоритмическом языке запишется так:

```
алг До стены
нач
  нц пока слева свободно
    влево
  кц
кон
```

Вспомните, что обозначают служебные слова **нц** и **кц**.

После слова **пока** записывается условие: цикл выполняется, пока это условие истинно. Как только условие становится ложным, работа цикла заканчивается.

Что произойдёт, если слева от Робота нет стены? Почему?

Зацикливание — это ситуация, когда цикл выполняется бесконечно.

Решите задачу, показанную на рис. 6.30, используя цикл с условием. Проверьте ваше решение на компьютере.

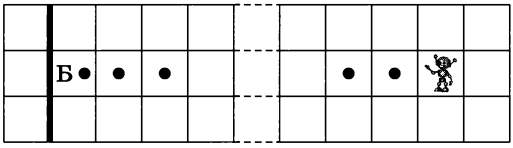


Рис. 6.30

Будет ли ваша программа работать правильно, если Робот стоит на расстоянии одной клетки от стены (рис. 6.31)?

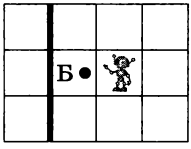


Рис. 6.31

Что произойдёт, если Робот стоит рядом со стеной (рис. 6.32)? Почему?

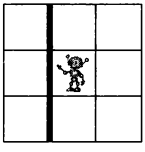


Рис. 6.32

Опишите все возможные обстановки (задачи), для которых ваша программа будет работать правильно.

Сравните цикл с условием и цикл «*N* раз» и ответьте на вопросы.

а) Если ли задачи, которые можно решить с помощью цикла с условием, но нельзя решить с помощью цикла «*N* раз»?

б) Если ли задачи, которые можно решить с помощью цикла «*N* раз», но нельзя решить с помощью цикла с условием?

в) Какой тип цикла позволяет решать большее количество задач?

Вложенные циклы

Задача. Роботу нужно закрасить 4 ряда клеток между двумя стенами, причём расстояние между этими стенами неизвестно (рис. 6.33).

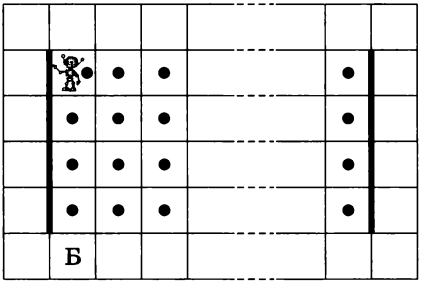


Рис. 6.33

Выделите в этой задаче две подзадачи, которые нужно выполнить несколько раз. Назовите их `Ряд` и `Переход`. Опишите в тетради эти подзадачи в словесном виде.

Найдите ошибку в решении подзадачи `Ряд`:

```
нц пока справа свободно
  вправо
  закрасить
кц
```

Какого типа эта ошибка? Как её исправить?

Найдите и исправьте логическую ошибку в решении подзадачи `Переход`:

```
вниз
нц пока слева свободно
  влево
кц
```

Подсказка: эта ошибка проявляется только после обработки последнего ряда.

Составьте полную программу решения задачи, показанной на рис. 6.33, и проверьте её работу на компьютере.

Теперь ещё усложним задачу — пусть рядов будет не четыре, а неизвестно сколько, причём эти ряды будут заканчиваться там, где заканчивается стена слева. Обратите внимание, что одна из этого множества задач — задача с четырьмя рядами, которую мы уже решили.

Можно ли для новой задачи использовать цикл «*N* раз»? Почему?

Какой цикл вы предлагаете использовать? Каково должно быть условие остановки цикла? Условие продолжения цикла?

Напишите программу, которая работает для любого количества рядов, и проверьте её на компьютере. Будет ли она правильно работать для задачи, показанной на рис. 6.33?

Выводы

- Цикл с условием — это цикл, который выполняется до тех пор, пока некоторое условие не станет ложным. Количество шагов такого цикла определяется исходными данными.
- Условие в заголовке цикла проверяется перед началом очередного шага цикла. Если условие в самом начале ложно, цикл не выполняется ни разу.
- Логическая команда — это запрос, на который исполнитель отвечает «да» или «нет».
- Обратная связь — это данные, которые передаются от датчиков исполнителя к управляющему устройству.
- Зацикливание — это ситуация, когда условие работы цикла никогда не становится ложным, и цикл выполняется бесконечно.
- Циклы с условием позволяют решать больше разнообразных задач, чем циклы «*N* раз».

Интеллект-карта

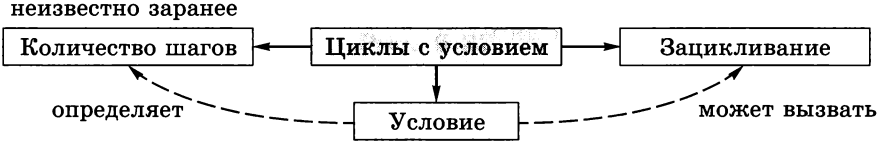


Рис. 6.34

Вопросы и задания

1. Объясните, в каких случаях нужно использовать цикл «N раз», а в каких — цикл с условием.
2. Какая серьезная логическая ошибка может встретиться при использовании цикла с условием?
3. Как по блок-схеме алгоритма определить, что он циклический?
4. Можно ли управлять исполнителем (например, Роботом) без обратной связи? В чём недостатки такого метода?
5. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Сравнение двух видов циклов»
б) «Циклы в природе и в технике»

Практическая работа

Выполните практическую работу № 28 «Циклы с условием».

§ 38
Разветвляющиеся алгоритмы

Ключевые слова:

- условие
- ветвление
- полная форма ветвления
- неполная форма ветвления
- разветвляющийся алгоритм
- вложенные ветвления

Что такое ветвление?

Рассмотрим две задачи для исполнителя Робота, в которых требуется перевести его в клетку, отмеченную буквой Б (рис. 6.35).

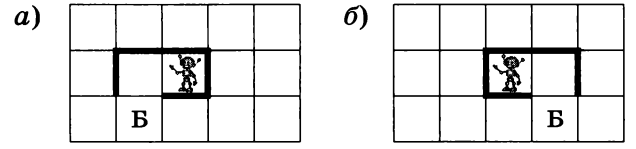


Рис. 6.35

Запишите в тетради решение каждой из этих задач в виде линейного алгоритма.

Можно ли написать такой линейный алгоритм, с помощью которого Робот решает обе задачи? Почему?

Сравните случаи а) и б) на рис. 6.35. Как Робот может отличить один случай от другого с помощью логических команд (команд-запросов)? Предложите разные варианты.

Алгоритмы, в которых действия исполнителя зависят от исходных данных (от обстановки), называются *разветвляющимися*. На блок-схеме такого алгоритма маршрут «расщепляется» на две ветви, появляются два пути выполнения алгоритма — **ветвление**.

Разветвляющийся алгоритм — это алгоритм, в котором последовательность действий изменяется в зависимости от выполнения некоторых условий.

Один из вариантов решения задачи, показанной на рис. 6.35, на алгоритмическом языке запишется так:

```
алг Выйти
нач
  если слева свободно то
    влево
  иначе
    вправо
  все
  вниз
кон
```

После слова **если** записано условие (команда-вопрос). Если условие слева свободно *истинно* (верно), то выполняются все команды между словами **то** и **иначе**. Если условие *ложно* (неверно), то исполнитель выполняет все команды между словами **иначе** и **все**.

Неполная форма ветвления

Запишите в тетради решения задач, показанных на рис. 6.36 (Роботу нужно прийти в клетку Б).

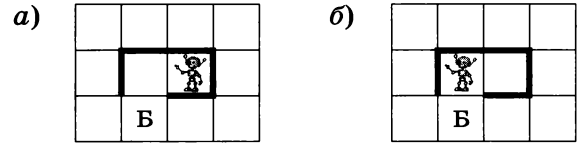


Рис. 6.36

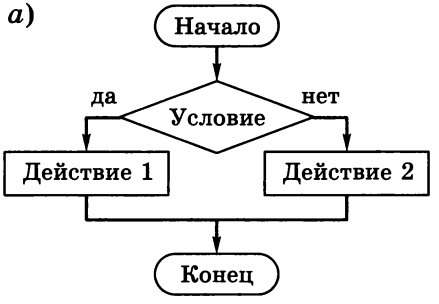
Сравните случаи *а* и *б* на рис. 6.36. Как Робот может отличить один случай от другого с помощью логических команд (команд-запросов)? Предложите несколько вариантов.

Исполнителю в любом случае нужно сделать шаг вниз, чтобы прийти в клетку Б. Но если слева нет стены, он должен ещё подготовиться — сделать шаг влево. А если слева уже есть стена, ничего дополнительно делать не нужно, поэтому ветвь «нет» на блок-схеме пустая! Такое ветвление называется **неполным**, а вариант, который мы использовали в начале параграфа, — **полной формой ветвления**.

Неполная форма ветвления отличается тем, что одна из веток на блок-схеме пустая.

Блок-схемы ветвлений в полной и неполной формах показаны на рис. 6.37.

Полная форма ветвления:



Неполная форма ветвления:

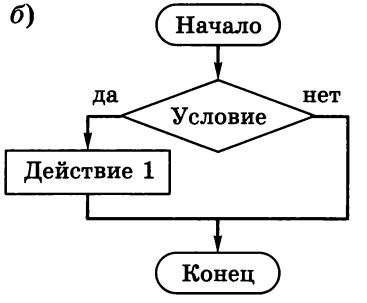


Рис. 6.37

Один из вариантов решения задачи, показанной на рис. 6.36, на алгоритмическом языке запишется так:

```
алг Выйти
нач
  если слева свободно то
    влево
  все
  вниз
кон
```

Обратите внимание, что в записи программы на Алгоритмическом языке нет слова **иначе**. Для ветвления в неполной форме второй блок команд (начинающийся словом **иначе**) не нужен, ведь он пустой.

Вложенные ветвления

Хватит ли одного ветвления для выбора одного из трёх вариантов? Сколько ветвлений нужно для того, чтобы выбрать один из *N* вариантов?

Напишем алгоритм, выполняя который Робот решает любую из трёх задач (рис. 6.38).

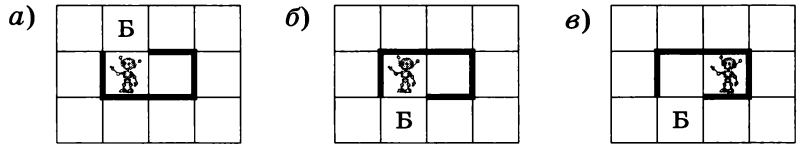


Рис. 6.38

Сравните три задачи на рис. 6.38. Чем отличатся задача *а* от двух остальных? С помощью какой команды-запроса Робот может определить, что имеет дело именно с задачей *а*?

Сравните задачи *б* и *в* на рис. 6.38. Чем они различатся? С помощью какой команды-запроса Робот может определить, что имеет дело именно с задачей *б*?

В результате получается алгоритм, который позволяет с помощью двух ветвлений выбрать один из трёх вариантов:

```
если сверху свободно то
   | работаем с задачей а
иначе
  если снизу свободно то
   | работаем с задачей б
  иначе
   | работаем с задачей в
  все
все
```

Часть алгоритма, обведённая штриховой линией, называется **вложенным ветвлением.** Вложенные ветвления используются для выбора более чем из двух вариантов действий.

Вложенное ветвление — это ветвление, которое расположено в одной из веток другого ветвления.

Закончите программу, заменив комментарии на команды из СКИ Робота. Проверьте её работу на компьютере.

Рассмотрим пример вычислительной задачи, которая решается с помощью вложенного ветвления.

Задача. Составьте алгоритм, который определяет знак числа. Результат его работы для некоторого числа x должен быть равен:

1, если число x положительное;
−1, если число x отрицательное;
0, если число x равно 0.

Сколько различных вариантов есть в этой задаче? Сколько ветвлений вам потребуется?

Выводы

- Разветвляющийся алгоритм — это алгоритм, в котором последовательность действий изменяется в зависимости от выполнения некоторых условий.
- Ветвление позволяет выбрать один вариант действий, если некоторое условие истинно, и другой — если это условие ложно.

- Неполная форма ветвления отличается тем, что одна из ветвей на блок-схеме — пустая (в одном из двух случаев никаких дополнительных действий делать не нужно).
- Вложенное ветвление — это ветвление, которое расположено в одной из ветвей другого ветвления.

Интеллект-карта

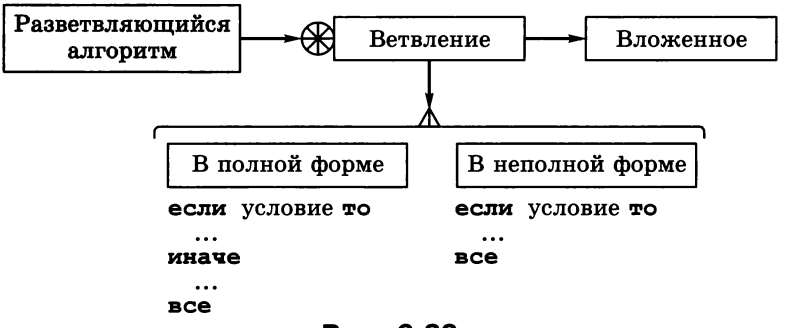


Рис. 6.39

Вопросы и задания

1. Как человек использует разветвляющиеся алгоритмы в жизни? Приведите примеры.
2. Почему во многих задачах невозможно обойтись без использования разветвляющихся алгоритмов?
3. Попытайтесь заменить полную форму ветвления на два неполных ветвления:

```
если слева стена то
  вправо
иначе
  влево
все
```

Всегда ли можно сделать такую замену?

4. Замените два ветвления в неполной форме на одно полное ветвление:

```
если слева свободно то
  влево
все
если слева стена то
  закрасить
все
```

В каких случаях такая замена возможна?

5. Можно ли заменить эти два ветвления в неполной форме на одно полное ветвление:

```
если слева свободно то
  влево
все
если справа стена то
  закрасить
все
```

Ответ обоснуйте.

6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Разветвляющиеся алгоритмы вокруг нас»

Практическая работа

Выполните практическую работу № 29 «Разветвляющиеся алгоритмы.

§ 39
Ветвления и циклы

Ключевые слова:

- ветвление
- цикл
- следование
- алгоритм Евклида
- диалоговая программа

В этом параграфе мы не будем изучать какие-то новые средства алгоритмического языка, а научимся строить более сложные программы, содержащие уже известные элементы: ветвления и циклы.

Пример задачи

Задача. Роботу нужно закрасить ряд клеток до клетки, отмеченной буквой Б (рис. 6.40).

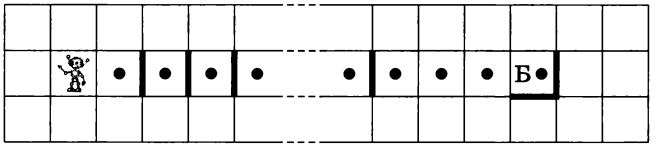


Рис. 6.40

На пути Робота в некоторых местах есть стены, которые можно обойти сверху. Если стены нет, Робот должен экономить энергию и идти прямо в соседнюю клетку.

Нужен ли в этой задаче циклический алгоритм? Почему?

Известно ли количество шагов цикла? Какой тип цикла нужно использовать?

Что отличает клетку Б, где Роботу нужно остановиться, от других клеток на пути? Запишите обратное условие, при выполнении которого Робот движется дальше.

Какие два варианта действий есть у Робота на каждом шаге?

С помощью какой логической команды Робот может определить, какой вариант применить?

Составьте в тетради блок-схему алгоритма решения задачи, изображённой на рис. 6.40. Найдите три основные алгоритмические конструкции:

- следование (последовательное выполнение, линейный алгоритм);
- ветвление (выбор одного из двух вариантов в зависимости от выполнения условия);
- повторение (цикл).

В середине XX века было строго доказано, что этих конструкций достаточно для того, чтобы записать любой алгоритм, который только можно придумать.

Алгоритм решения любой задачи можно составить с помощью трёх конструкций — **следования, ветвления** и **цикла.** Эти конструкции называют **базовыми алгоритмическими структурами.**

Блок-схемы эти структур показаны на рис. 6.41.

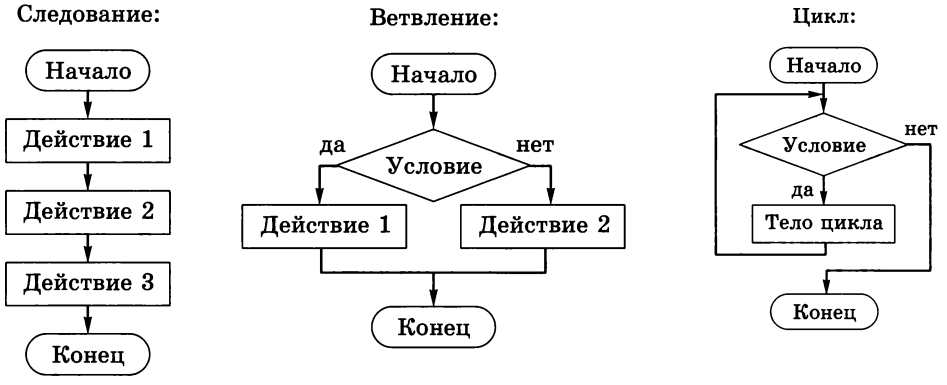


Рис. 6.41

Может ли быть так, что тело цикла на рис. 6.41 не выполнится ни разу? Когда это может произойти?

Цикл, показанный на рис. 6.41, — это **цикл с предусловием** (условием, которое проверяется в начале цикла). Иногда условие проверяют после выполнения цикла — если выполняется условие выхода из цикла, то цикл заканчивается. Такой цикл называется **циклом с постусловием**, т. е. с проверкой условия в конце тела цикла. Вот его блок-схема (рис. 6.42).

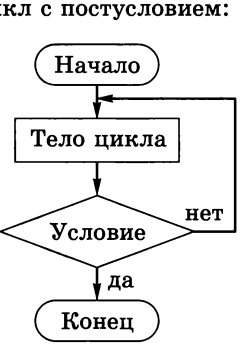


Рис. 6.42

Может ли быть так, что тело цикла с постусловием не выполнится ни разу? Когда это может произойти?

Сравните циклы на рис. 6.41 и рис. 6.42. В каком из них при выполнении условия в блоке «выбор» цикл продолжается, а в каком — заканчивается?

Исполнитель Раздвоитель

Рассмотрим ещё одного исполнителя, которого зовут Раздвоитель. СКИ Раздвоителя содержит две команды

1. `вычти 1`
2. `раздели на 2`

Какие команды будут обратными к командам 1 и 2? Сравните этого исполнителя с Удвоителем.

Исполнитель получает натуральное число. Может ли он в результате выполнения какой-либо программы получить число, большее, чем начальное?

Опишите всё множество чисел, которые может получить Раздвоитель из натурального числа `N` с помощью всевозможных программ.

Сравните два алгоритма для Раздвоителя.
Можно ли сразу сказать, какой результат получится из натурального числа `N` после выполнения этих алгоритмов? Одинаков ли этот результат для обоих алгоритмов? Как вы рассуждали?

Алгоритм 1:

```
нц пока N не ноль
  вычти 1
кц
```

Алгоритм 2:

```
нц пока N не ноль
  если N - чётное то
    раздели на 2
  иначе
    вычти 1
  все
кц
```

Алгоритм Евклида

Древнегреческий математик Евклид, живший в III веке до нашей эры, придумал замечательный алгоритм для вычисления наибольшего общего делителя (НОД) двух натуральных чисел. Этот алгоритм и сейчас используется, например в задачах шифрования. Напомним, что НОД — это наибольшее число, на которое два заданных числа делятся без остатка.

Алгоритм Евклида для натуральных чисел: заменять большее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число и есть их НОД.

Алгоритм Евклида можно записать более строго в виде последовательности шагов:

Вход: натуральные числа *a*, *b*.
Шаг 1. Если *a* = *b*, то перейти к шагу 7.
Шаг 2. Если *a* < *b*, то перейти к шагу 5.
Шаг 3. *a* := *a* – *b*.
Шаг 4. Перейти к шагу 1.
Шаг 5. *b* := *b* – *a*.
Шаг 6. Перейти к шагу 1.
Шаг 7. Стоп.
Результат: *a*.

Можно ли считать этот алгоритм циклическим? Почему?

Можно ли считать этот алгоритм разветвляющимся? Почему?

Нарисуйте в тетради блок-схему алгоритма Евклида. Найдите в этом алгоритме следование, ветвление и цикл.

Нарисованная вами блок-схема описывает достаточно простой алгоритм и уже выглядит довольно запутанно. Поэтому для описания сложных алгоритмов блок-схемы не используют — алгоритм записывают сразу на языке программирования или на **псевдокоде** — смеси естественного языка (русского, английского) и какого-нибудь языка программирования.

Программу на алгоритмическом языке можно записать так:

```
алг Евклид
нач
  цел a, b
  ввод a, b
  нц пока a<>b
    если a<b то
      b:=b-a
    иначе
      a:=a-b
    все
  кц
  вывод a
кон
```

Условие a<>b обозначает «a не равно b», оно записывается с помощью двух знаков, «<» и «>», без пробела между ними[1].

В этой программе вы увидели две новые команды алгоритмического языка, ввод и вывод. Они используются для ввода значений с клавиатуры и вывода результатов на экран, т. е. для диалога[2] компьютера с человеком. Такие программы называются диалоговыми.

Диалоговая программа — это программа, которая ведёт диалог с человеком.

Проверьте работу программы Евклид на компьютере.

Выводы

- Алгоритм решения любой задачи можно составить с помощью трёх конструкций — следования, ветвления и цикла. Эти конструкции называют базовыми алгоритмическими структурами.
- Ветвления позволяют выбрать нужный вариант действий на каждом шаге цикла.
- Алгоритм Евклида для натуральных чисел: заменять большее из двух заданных чисел на их разность до тех пор, пока числа не станут равны. Полученное число есть НОД этих чисел.
- Диалоговая программа — это программа, в которой исходные данные вводятся человеком с клавиатуры, а результаты работы выводятся на экран.

Интеллект-карта

Рис. 6.43

Вопросы и задания

1. Как можно найти НОД двух натуральных чисел, не используя алгоритм Евклида? Какой метод лучше? Сравните этот алгоритм с алгоритмом Евклида.
2. Как вы думаете, в чём достоинства диалоговых программ?
3. Выполните по указанию учителя задания в рабочей тетради.

[1] Так сделали потому, что клавиши «≠» на клавиатуре нет.
[2] Диалог — это общение двух объектов, например компьютера и человека.

Подготовьте сообщение

«Алгоритм Евклида»

Практическа работа

Выполните практическую работу № 30 «Ветвления и циклы».

§ 40
Компьютерная графика

***Ключевые слова*:**

- графический режим
- исполнитель Рисователь
- холст
- пиксель
- координаты
- оси координат
- переменная
- цикл по переменной

Что такое графический режим?

Много лет назад человек общался с компьютером только в **текстовом режиме** — вводил с клавиатуры числа и символы и получал ответ компьютера тоже в форме текста. Мониторы работали в текстовом режиме. Это значит, что весь экран был разбит на прямоугольники-символы (например, 25 строк по 80 символов в каждой) и программа могла управлять каждым символом. При этом было очень сложно что-то нарисовать на экране, потому что рисунок нужно было тоже строить из символов.

Современные мониторы работают в **графическом режиме**. Это значит, что программа может управлять отдельными точками на экране монитора, строить (по точкам) диаграммы и графики, а также выводить на экран растровые (точечные) изображения, например фотографии. Вы уже научились создавать рисунки в графических редакторах. Теперь мы будем составлять программы, которые рисуют без нашего участия, автоматически. Для этого можно использовать исполнитель Рисователь, который входит в систему КуМир.

Исполнитель Рисователь

Исполнитель Рисователь рисует на *холсте*.

Холст — это прямоугольная область экрана, доступная для рисования.

Если исполнитель попытается нарисовать что-то за пределами холста, эта часть рисунка будет потеряна.

Холст — это прямоугольник, состоящий из отдельных пикселей, т. е. **растровый рисунок.** Каждый пиксель имеет две координаты:

x — расстояние от пикселя до левой границы холста;

y — расстояние от пикселя до верхней границы холста.

Рисователь использует прямоугольную систему координат, показанную на рис. 6.44.

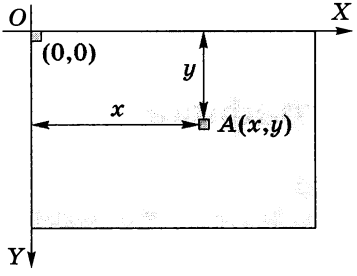


Рис. 6.44

Сравните систему координат на рис. 6.44 с системой координат, которую вы используете на уроках математики. Что у них общего и чем они различаются?

Как вы думаете, зачем «перевернули» ось *Y*?

Пиксель в левом верхнем углу холста имеет координаты (0,0). Нумерация с нуля часто используется в программировании. В нашем случае это значит, что пиксель находится одновременно на левой границе и на верхней границе холста (оба расстояния до границ равны нулю).

Напишем первую программу для Рисователя: создадим холст размером 500 × 400 пикселей (ширина — 500 пикселей, высота — 400 пикселей) и зальём его белым цветом:

```
использовать Рисователь
алг Холст
нач
  новый лист( 500, 400, белый )
кон
```

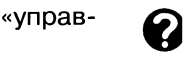
Сравните эту программу с программами для Робота. Что означает первая строка?

Единственная строка основной программы — вызов команды `новый лист`, входящей в СКИ Рисователя. Данные в скобках называются **аргументами**. Они задают свойства холста: первое число в скобках

(первый аргумент) означает ширину холста в пикселях, второе — высоту, а третье — название цвета, которым заливается холст. Вот все простые цвета, которые можно использовать:

белый, чёрный, серый, фиолетовый, синий,
голубой, зелёный, жёлтый, оранжевый, красный.

Управление пикселями

Какие свойства имеет пиксель? Как вы думаете, что значит «управлять пикселем»? 

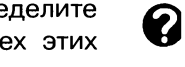
У Рисователя есть команда пиксель, которая изменяет цвет пикселя с заданными координатами. Например, закрасить синим цветом пиксель с координатами[1] (10,20) можно так:

```
пиксель(10, 20, синий)
```

Первые два аргумента команды пиксель — это x-координата и y-координата пикселя, а третий — нужный цвет.

Задача. Нарисовать горизонтальную линию (отрезок) синего цвета из точки с координатами (10,20) в точку с координатами (15,20).

Из каких элементов состоит отрезок на растровом рисунке? 

Какие пиксели нужно закрасить для решения задачи? Определите их координаты. Какая из двух координат одинакова для всех этих пикселей? Какая изменяется? 

Чтобы не писать много похожих команд, мы попытаемся построить цикл. Обозначим первую (изменяющуюся) координату пикселя именем X. Тогда нам нужно 6 раз выполнить команду

```
пиксель(X, 20, синий)
```

изменяя значение переменной X от 10 до 15. Это можно сделать, например, так:

```
цел X
X:=10
нц пока X<=15
  пиксель(X, 20, синий)
  X:=X+1
кц
```

В алгоритмическом языке есть ещё один вид цикла — **цикл по переменной**, — который позволяет записать то же самое задание для Рисователя проще:

[1] Координаты пикселей записываются в круглых скобках через запятую, сначала x-координата, потом y-координата.

```
цел X
нц для X от 10 до 15
  пиксель(X, 20, синий)
кц
```

В заголовке цикла мы сказали, что тело цикла нужно выполнить для всех целых значений X от 10 до 15 включительно. Всю работу по увеличению переменной X на каждом шаге цикла программа берёт на себя, нам не нужно об этом заботиться.

На блок-схеме цикл по переменной изображается так (рис. 6.45).

Цикл по переменной:

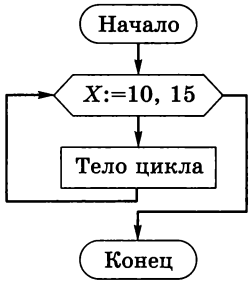


Рис. 6.45

Внутри блока ⬡ записывают имя переменной цикла, её начальное и конечное значения.

Напишите программу, которая тем же методом рисует вертикальный отрезок.

Какие сложности возникнут при рисовании наклонных линий? Окружностей?

Чтобы облегчить работу программистам, в СКИ графических исполнителей добавляют команды, которые рисуют сразу целые геометрические фигуры: линии, прямоугольники, окружности и др. Эти фигуры называются **графическими примитивами**. В следующем параграфе мы познакомимся с графическими примитивами в СКИ Рисователя.

Выводы

- Современные мониторы работают в графическом режиме. Это значит, что программа может управлять отдельными точками на экране монитора.
- Холст — это прямоугольная область экрана, доступная для рисования.

- В графическом режиме используется прямоугольная система координат. Начало координат — точка с координатами (0, 0) — находится в левом верхнем углу холста. Ось *X* направлена вправо, ось *Y* — вниз.
- Горизонтальные и вертикальные отрезки можно легко нарисовать, используя цикл по переменной.

Интеллект-карта

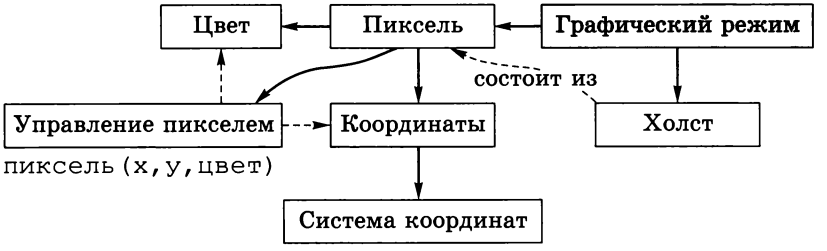


Рис. 6.46

Вопросы и задания

1. Чем различаются программы, работающие в текстовом и графическом режимах?
2. Как определяются координаты пикселя на холсте?
3. Чем необычна система координат, которая используется в графическом режиме? Как вы думаете, почему выбрана именно такая система координат?
4. Что будет, если мы вызовем команду `пиксель` так:

   ```
   пиксель (синий, 10, 20)
   ```

   В чём ошибка?
5. Чем отличается цикл по переменной от цикла «*N* раз» и от цикла с условием?
6. Можно ли зациклить программу, используя цикл по переменной?
7. Выполните по указанию учителя задания в рабочей тетради.

Практическая работа

Выполните практическую работу № 31 «Управление пикселями».

§ 41
Графические примитивы

***Ключевые слова*:**

- графический примитив
- линия
- прямоугольник
- окружность
- ломаная
- контур
- перо
- заливка
- кисть

Что такое графические примитивы?

Графический примитив — это геометрическая фигура, которая добавляется на рисунок с помощью одной команды.

Основные примитивы исполнителя Рисователь:

- пиксель;
- линия;
- прямоугольник;
- окружность.

Линии рисуются пером. Перо — это набор свойств линии: толщина и цвет. Для того чтобы нарисовать линию, нужно сначала выбрать её характеристики с помощью команды `перо`:

```
перо(1, синий)
```

Первый аргумент в скобках — это толщина линии, второй — её цвет. Обратите внимание, что эта команда сама по себе *ничего не рисует*, но устанавливает режим рисования для следующих команд. Такой приём значительно сокращает запись, когда для нескольких команд используются одинаковые свойства линии — их не нужно указывать в каждой команде рисования отдельно.

Теперь применим команду `линия`:

```
линия( 10, 20, 15, 20 )
```

Первые два аргумента — это координаты одного конца линии (точнее, отрезка), следующая пара — координаты второго конца. Здесь мы рисуем отрезок из точки с координатами (10, 20) в точку с координатами (15, 20).

Сравните рисование линии по пикселям и использование команды `линия`. Какой способ проще?

Рисователь умеет строить прямоугольники, но только такие, у которых стороны строго вертикальны и строго горизонтальны. Для того чтобы нарисовать такой прямоугольник, нужно знать координаты двух его противоположных углов, например верхнего левого и правого нижнего.

Координаты левого верхнего угла прямоугольника — (*a, b)*, а координаты правого нижнего — (*c, d*). Определите координаты остальных углов прямоугольника и его размеры.

Как можно было бы нарисовать прямоугольник, если бы в СКИ Рисователя не было команды `прямоугольник`?

Эта программа рисует прямоугольник, противоположные углы которого находятся в точках с координатами (20, 10) и (40, 30) — рис. 6.47 (и цветной рисунок на форзаце).

```
перо(1, синий)
кисть(красный)
прямоугольник(20, 10, 40, 30)
```

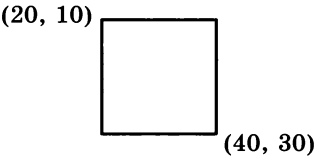


Рис. 6.47

В первой строке мы определяем свойства контура — толщину и цвет. Прямоугольник — это замкнутая фигура, поэтому для него можно задать ещё и цвет заливки. Это делает команда `кисть` во второй строке. Если нужно нарисовать только рамку (не заливая внутреннюю часть фигур), устанавливается прозрачный цвет кисти:

```
кисть(прозрачный)
```

Команды `перо` и `кисть` ничего не рисуют, а только устанавливают режимы рисования для команды `прямоугольник` в последней строке.

Используя рис. 6.47 и программу к нему, выясните, сколько аргументов имеет команда `прямоугольник`. В каком порядке они перечисляются?

Команда `окружность` предназначена для рисования окружностей и кругов. Ей нужно передать три аргумента: координаты центра и радиус. Окружность — замкнутая фигура, поэтому для неё можно задавать свойства пера и цвет заливки внутренней части (круга).

Эта программа рисует круг с центром в точке (50, 30) радиуса 20 пикселей (рис. 6.48 и цветной рисунок на форзаце).

```
перо(1, синий)
кисть(красный)
окружность(50, 30, 20)
```

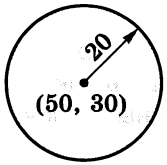


Рис. 6.48

Какие цвета будут иметь контур окружности и внутренняя часть круга после выполнения этой программы?

Как нужно вызвать команду `окружность` для того, чтобы нарисовать окружность радиуса *R* с центром в точке с координатами (*x*, *y*)?

Ломаные

Вспомните, из каких примитивов состоит ломаная.

Специальной команды для рисования треугольника нет (как вы думаете почему?). Но треугольник можно построить из отрезков, вызвав несколько раз команду `линия` (рис. 6.49).

```
линия(20, 30, 30, 10)
линия(30, 10, 40, 30)
линия(40, 30, 20, 30)
```

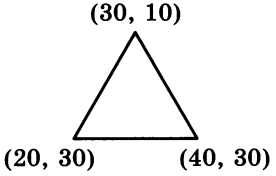


Рис. 6.49

Сравните первые две строки программы рисования треугольника. Какие данные в них повторяются? Почему?

Ломаную линию можно нарисовать, «не отрывая пера от холста», с помощью других команд, которые приведут к точно такому же результату:

```
в точку (20, 30)
линия в точку (30, 10)
линия в точку (40, 30)
линия в точку (20, 30)
```

Команда `в точку` переводит исполнитель в заданную точку холста, не оставляя следа, а команда `линия в точку` рисует линию из того места, где стоял исполнитель, в точку с новыми координатами.

Как по программе определить, что ломаная замкнута? Как нарисовать незамкнутую ломаную?

Заливка

Вспомните, как выполнялась заливка прямоугольников и окружностей.

Пусть контур фигуры уже готов, и нужно залить внутреннюю часть. Для этого в СКИ Рисователя есть команда `залить`, которая заливает одноцветную область, начиная с заданной точки:

```
кисть (красный)
залить (30, 20)
```

В этом примере заливка красным цветом начинается в точке с координатами (30, 20). Где же она остановится? Это зависит от цвета пикселя с координатами (30, 20). Если, допустим, этот пиксель был белым до начала заливки, то заливка остановится на границе белой области.

Пример

Используя только что изученные команды, нарисуем беседку (рис. 6.50 и цветной рисунок на форзаце).

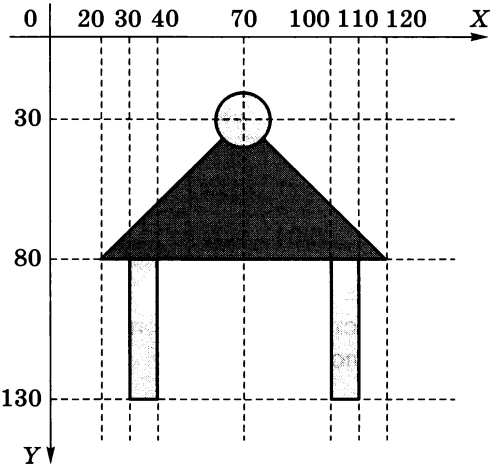


Рис. 6.50

Два столбика покрашены в серый цвет, крыша — в синий, а шарик на крыше — в красный.

Из каких геометрических фигур состоит рисунок 6.50? Для каких из них есть готовые примитивы?

Определите по рисунку 6.50:
а) координаты углов опор-прямоугольников;
б) координаты углов треугольной крыши;
в) координаты центра и радиус шарика.

В какой последовательности нужно рисовать эти фигуры? Возможны ли разные (правильные!) варианты? Как вы рассуждали?

Поскольку красный круг «накрывает» крышу, его нужно рисовать после крыши. Прямоугольники и треугольник можно рисовать в любом порядке. Вот один из вариантов программы (для удобства обсуждения строки пронумерованы в комментариях справа):

```
алг Беседка
нач
  новый лист(200, 200, белый)                | 1
  перо(1, черный)                            | 2
  кисть(серый)                               | 3
  прямоугольник(30, 80, 40, 130)             | 4
  прямоугольник(100, 80, 110, 130)           | 5
  в точку(20, 80)                            | 6
  линия в точку(70, 30)                      | 7
  линия в точку(120, 80)                     | 8
  линия в точку(20, 80)                      | 9
  кисть(синий)                               | 10
  залить(70, 40)                             | 11
  кисть(красный)                             | 12
  окружность(70, 30, 10)                     | 13
кон
```

Определите размер холста, который используется. Можно ли было взять другой размер холста? Какой?

В каких строках исполнитель рисует опоры? В какой строке определяется цвет крыши?

Какие соседние строки в программе можно поменять местами? Какие нельзя? Как вы рассуждали?

Можно ли строки 12 и 13 перенести в другое место программы? Если да, то куда именно?

Выводы

- Графический примитив — это геометрическая фигура, которая добавляется на рисунок с помощью одной команды.
- Основные графические примитивы — пиксель, линия (отрезок), прямоугольник, окружность.
- При рисовании отрезка нужно задать координаты его концов.
- При рисовании прямоугольника задаются координаты его противоположных углов.
- При рисовании окружности задаются координаты её центра и радиус.
- Ломаную линию можно нарисовать как последовательность отрезков.
- С помощью команды `залить` выполняется заливка области одного цвета, начиная с некоторой точки холста.

Интеллект-карта

Рис. 6.51

Вопросы и задания

1. Зачем в СКИ графических исполнителей добавляют команды для рисования примитивов?
2. Алгоритмы какого типа мы использовали в этом параграфе?
3. Как нарисовать замкнутую ломаную линию и залить её внутреннюю область? С какими проблемами вы можете столкнуться?
4. Как вы думаете, почему обычно в СКИ исполнителей нет команды «треугольник»?
5. Выясните, что произойдёт, если исполнитель будет рисовать за пределами холста.
6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Алгоритмы выполнения заливки»
б) «Команда эллипс»
в) «Цвет в модели RGB»
г) «Алгоритмы Брезенхэма»

Практическая работа

Выполните практическую работу № 32 «Графические примитивы».

§ 42
Применение процедур

***Ключевые слова*:**

- процедура
- параметры

Когда помогут процедуры?

Вспомните, зачем используются вспомогательные алгоритмы (процедуры).

Задача. Нарисовать три прямоугольных треугольника одинаковых размеров (рис. 6.52 и цветной рисунок на форзаце).

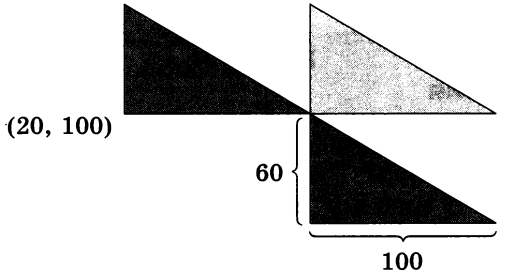


Рис. 6.52

Определите, что общего у трёх нарисованных фигур и чем они различаются.

Поскольку треугольники очень похожи, хочется нарисовать их с помощью одной процедуры. Однако треугольники всё-таки различаются, поэтому нам нужна **процедура с параметрами**.

Строим процедуру

Сколько координат точек треугольника нужно задать, чтобы в данной задаче построить весь треугольник?

Предположим, что мы знаем координаты (x, y) прямого угла в треугольнике и его размеры (w — ширина основания, h — высота). Определите координаты остальных углов, середины наклонной стороны и любой точки внутри треугольника (рис. 6.53).

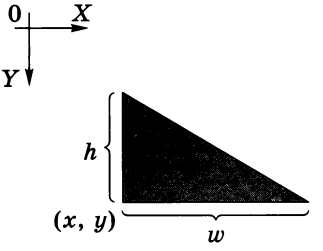


Рис. 6.53

Точку, которая задаёт положение фигуры на холсте, называют **опорной или базовой точкой**.

Базовая точка — это точка, через координаты которой вычисляют координаты всех остальных точек фигуры.

Процедура должна принимать три аргумента: координаты (x, y) одной из точек треугольника (например, прямого угла) и цвет заливки. Заготовка процедуры выглядит так:

```
алг треугольник(цел x, y, цвет ц)
нач
  ...
кон
```

Параметр с именем `ц` — это величина специального типа `цвет`, которая обозначает цвет заливки. Полный текст процедуры может выглядеть так:

```
алг треугольник(цел x, y, цвет ц)
нач
  в точку(x, y)
  линия в точку(x, y-60)
  линия в точку(x+100, y)
  линия в точку(x, y)
  кисть(ц)
  залить(x+10, y-10)
кон
```

В теле процедуры мы рисуем замкнутую ломаную линию, используя рассчитанные координаты углов треугольника, и затем заливаем его тем цветом, который будет передан в процедуру вызывающей программой. Все координаты зависят от значений *x* и *y*, т. е. процедура позволяет нам рисовать треугольники в любом месте холста.

Обратите внимание, что точка, откуда начинается заливка (в последней строке процедуры), может быть выбрана и по-другому, главное — попасть внутрь треугольника.

Используем процедуру

Посмотрите на рис. 6.52. Сколько вызовов процедур должно быть в основной программе?

По рисунку 6.52 определите координаты вершины прямого угла и цвет каждого треугольника.

Основная программа содержит только три вызова процедуры:

```
использовать Рисователь
алг Трио
нач
  треугольник(20, 100, синий)
  треугольник(120, 100, зеленый)
  треугольник(120, 160, красный)
кон
```

Проверьте работу программы на компьютере. Не забудьте после этой программы поместить текст процедуры. Для того чтобы разобраться, как работает программа, используйте пошаговый режим с заходом в процедуру (клавиша F7).

Выводы

- Процедуры можно использовать для рисования одинаковых и похожих фигур в графическом режиме.
- Обычно используют процедуры с параметрами, потому что фигуры различаются, по крайней мере, координатами.
- В параметры включают те величины, которыми различаются фигуры.
- При составлении процедур используют метод базовой точки. Базовая точка — это точка, через координаты которой вычисляют координаты всех остальных точек фигуры.

Интеллект-карта

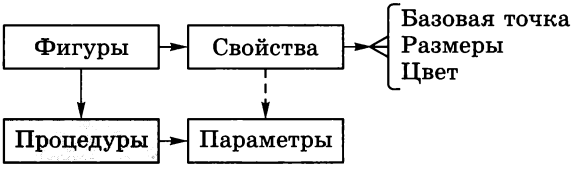


Рис. 6.54

Вопросы и задания

1. Почему процедуры для рисования фигур на холсте, как правило, имеют параметры?
2. Как определить, какие данные включать в список параметров процедуры?
3. Можно ли было включить в параметры процедуры `треугольник` ещё длины сторон (высоту и длину основания) треугольника? Какие достоинства и недостатки имеет это решение?
4. Если процедура `треугольник` записана ниже основной программы, но треугольники не появляются на холсте, то в чём может быть причина? Как вы будете искать ошибку?
5. Выполните по указанию учителя задания в рабочей тетради.

Практическая работа

Выполните практическую работу № 33 «Применение процедур».

§ 43
Применение циклов

***Ключевые слова*:**

- цикл
- узор
- процедура
- свойства по умолчанию

Узоры

Узор — это рисунок, основанный на повторении одинаковых элементов.

Задача 1. Построить ряд из пяти окружностей радиуса 5 пикселей (рис. 6.55).

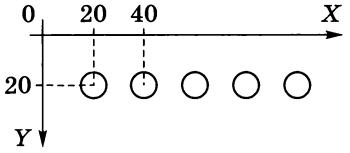


Рис. 6.55

Как можно нарисовать эти окружности с помощью отдельных вызовов команды `окружность`?

Какие виды циклов вы знаете? Какой из них лучше использовать в этой задаче?

Можно заметить, что у кругов на рис. 6.55 изменяется только *x*-координата центра. Её можно сделать переменной и назвать `x`. Эта переменная будет меняться от 20 до 100 с шагом 20, поэтому можно использовать такой цикл по переменной:

```
цел x
нц для x от 20 до 100 шаг 20
  окружность(x, 20, 5)
кц
```

Здесь в заголовке цикла появилась новая часть — `шаг 20`. Это значит, что изменение значения переменной происходит с шагом 20: 20, 40, 60, ... *По умолчанию* (т. е. если мы не указали в явном виде) шаг равен единице.

Задача 2. Построить три одинаковых ряда окружностей, расположенных на расстоянии 20 пикселей по высоте друг от друга (рис. 6.56).

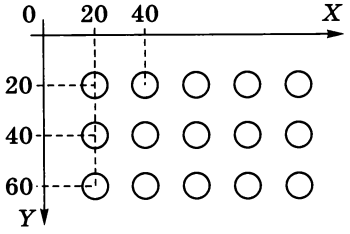


Рис. 6.56

Чем различаются три горизонтальных ряда на рис. 6.56?

Оформите рисование одного ряда окружностей в виде процедуры Ряд, которая принимает один параметр — *y*-координату центра окружностей.

Основная программа три раза вызывает процедуру Ряд:

```
алг Узор
нач
  Ряд(20)
  Ряд(40)
  Ряд(60)
кон
```

Найдите закономерность в изменении аргумента, передаваемого процедуре Ряд из основной программы: начальное значение, конечное значение, шаг изменения.

Можно записать основную программу и иначе, используя цикл по переменной:

```
алг Узор
нач
  цел y
  нц для y от 20 до 60 шаг 20
    Ряд(y)
  кц
кон
```

Использование процедур

Задача 3. Построить на экране такой узор из ромбов (рис. 6.57).

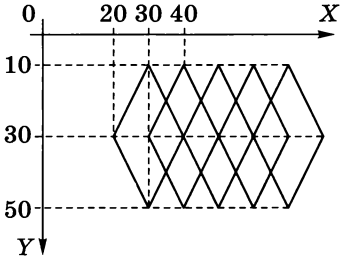


Рис. 6.57

По рисунку 6.57 определите размеры одного ромба. Выберите базовую точку для рисования ромба и обозначьте её координаты через (*x*, *y*). Определите через неё координаты всех углов ромба.

Сравните разные варианты выбора базовой точки. Какой из них лучше? Как вы рассуждали?

Процедуру, которая рисует контур ромба, можно написать так:

```
алг Ромб(цел x, y)
нач
  в точку(x, y)
  линия в точку(x+10, y-20)
  линия в точку(x+20, y)
  линия в точку(x+10, y+20)
  линия в точку(x, y)
кон
```

У процедуры два параметра — координаты левого угла ромба (*базовой точки*). По рис. 6.57 находим, что для первого ромба это координаты (20, 30), для второго — (30, 30), для третьего — (40, 30) и т. д.:

```
Ромб(20, 30)
Ромб(30, 30)
Ромб(40, 30)
Ромб(50, 30)
Ромб(60, 30)
```

Видим, что *x*-координата увеличивается с шагом 10, а *y*-координата не изменяется. Поэтому можно использовать цикл по переменной:

```
цел x
нц для x от 20 до 60 шаг 10
  Ромб(x, 30)
кц
```

Штриховка

Во многих задачах компьютерной графики нужно заштриховать какие-то области. Например, штриховкой обозначается сечение на чертежах и болота на картах местности. Штриховка строится из параллельных линий, которые удобно рисовать в цикле. Выполним вертикальную штриховку прямоугольника, разделив его на N полос (рис. 6.58).

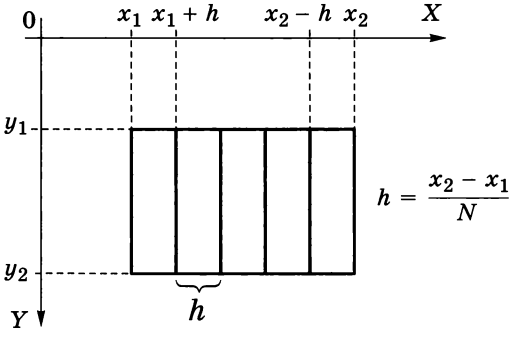


Рис. 6.58

Решим задачу в общем виде для любого прямоугольника. Будем считать, что его верхний левый угол находится в точке с координатами (x_1, y_1), а правый нижний — в точке с координатами (x_2, y_2). Сначала нарисуем контур прямоугольника:

```
цел x1=100, x2=300
цел y1=100, y2=200
прямоугольник(x1, y1, x2, y2)
```

Заметьте, что в первых двух строках мы не только объявляем переменные `x1`, `x2`, `y1` и `y2`, но и присваиваем им начальные значения. Конечно, вы можете выбрать и другие числа.

В результате штриховки нужно разделить прямоугольник на N полос, поэтому шаг штриховки (расстояние между соседними линиями) можно вычислить по формуле:

$$h = \frac{x_2 - y_1}{N}.$$

В нашей программе мы будем использовать только целые значения шага (в пикселях), поэтому величина h вычисляется с помощью **деления нацело**, которое в алгоритмическом языке записывается как вспомогательный алгоритм с именем `div`. Например, `div(a, b)` — это целая часть от деления `a` на `b`. В нашем случае шаг вычисляется так:

```
цел h
h:=div(x2-x1, N)
```

Какой шаг получится, если `x1` = 100, `x2` = 200 *и* `N` = 5?

Определите координаты концов первого слева, второго и последнего отрезков штриховки внутри прямоугольника.

У всех линий штриховки отличается только *x*-координата, которая изменяется от `x1 + h` до `x2 - h` с шагом `h`. Поэтому штриховку можно выполнить с помощью цикла:

```
цел x
нц для x от x1+h до x2-h шаг h
  линия(x, y1, x, y2)
кц
```

Составьте программу для Рисователя, которая рисует прямоугольник и выполняет его штриховку. Проверьте её работу на компьютере.

Выводы

- При выполнении узоров (повторяющихся рисунков) и штриховки удобно использовать цикл по переменной.
- Для того чтобы построить цикл, можно записать команды для рисования нескольких элементов узора, в том числе первого и последнего. После этого следует определить, какие величины изменяются и как именно (чему равны начальное и конечное значения, шаг).
- Штриховка обычно строится из параллельных линий, расположенных на одинаковом расстоянии друг от друга.
- Для вычисления шага штриховки нужно разделить длину отрезка, который разбивается на равные части, на количество полос. Для того чтобы выполнить деление целых чисел с отбрасыванием остатка, используют команду `div`.

Интеллект-карта

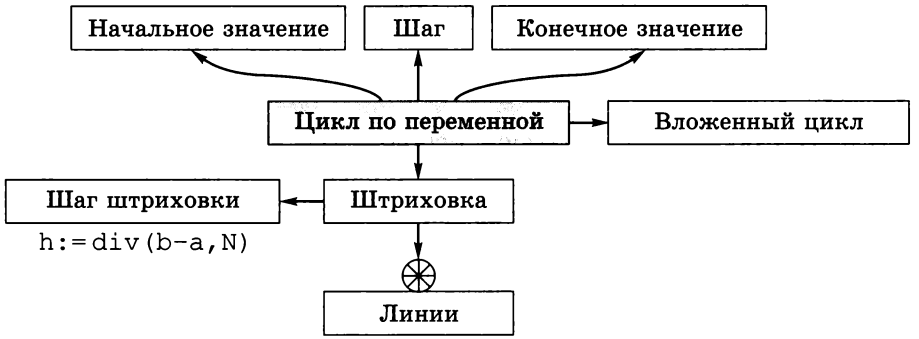
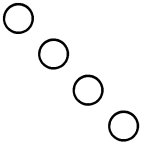


Рис. 6.59

Вопросы и задания

1. Что нужно изменить в первой программе из задачи 1, чтобы окружности выстроились по диагонали?



   Нужен ли для этого вложенный цикл?
2. Сравните два варианта решения задачи 2:
   а) вызов процедуры в цикле;
   б) вложенные циклы.
   Какой из них вам больше нравится? Почему?
3. Приведите примеры практических задач, где требуется штриховка. С какими из них вы встречались?
4. Что нужно изменить в программе, выполняющей вертикальную штриховку прямоугольника, чтобы сделать горизонтальную штриховку?
5. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Муаровый эффект»

Практические работы

Выполните практические работы:
№ 34 «Применение циклов»;
№ 35 «Штриховка».

---

§ 44
Анимация

Ключевые слова:

- анимация
- кадр
- смена кадров
- координаты объекта
- текущие координаты
- прозрачный цвет
- пауза

Принципы анимации

Слово «анимация» произошло от латинского слова *animatio*, что означает «оживление». При анимации быстрая смена мало отличающихся рисунков (**кадров**) создаёт иллюзию движения. Если кадры меняются чаще, чем 16 раз в секунду, человеческий глаз не успевает реагировать на каждое изменение и видит плавное движение.

Допустим, мы нарисовали четыре кадра одинакового размера (рис. 6.60) и меняем их на экране, скажем, через каждую секунду: сначала выводим кадр а), затем, через 1 с — кадр б), ещё через 1 с — кадр в) и т. д. Что мы увидим? Движение шарика слева направо. Конечно, оно не будет плавным, потому что выбран большой интервал времени. Но если менять кадры чаще, чем 16 раз в секунду, глаз перестаёт замечать смену изображений и человеку кажется, что движение непрерывное.

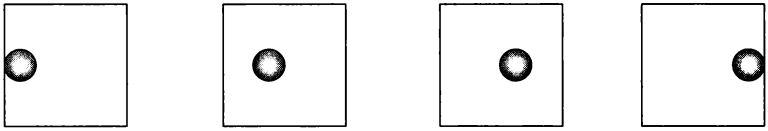


Рис. 6.60

Всегда ли можно заранее нарисовать все кадры анимации, например, в компьютерных играх? Почему?

Предположим, что нам нужно изобразить движение объекта (например, шарика) на каком-то фоне. Фон может быть одноцветный или в виде картинки. Для перемещения шарика нам нужно:

1) «стереть» шарик, т. е. восстановить фон в том месте, где сейчас нарисован шарик;
2) изменить положение шарика (его координаты на холсте);
3) запомнить участок фона, который будет испорчен при выводе шарика в новом месте;
4) вывести изображение шарика поверх фона.

Если фон одноцветный, то задача упрощается, потому что не нужно запоминать изображение, перекрытое шариком. Чтобы стереть шарик, достаточно залить это место цветом фона.

Используя этот подход, мы напишем программу для моделирования движения шарика на экране.

Анимация движения

Сначала создадим новый холст для анимации:

```
новый лист(200, 200, синий)
```

Вспомните, что означают аргументы, переданные команде `новый лист`.

Шарик будем изображать в виде круга жёлтого цвета. Его контур рисовать не будем, поэтому сразу установим прозрачный цвет пера:

```
перо(1, прозрачный)
```

Рисование и стирание шарика можно выполнять одной процедурой, которая принимает три параметра: пару координат (`x`, `y`) центра шарика и цвет заливки:

```
алг Шарик(цел x, y, цвет ц)
нач
  цел R=10
  кисть(ц)
  окружность(x, y, R)
кон
```

Что означает число 10 в тексте процедуры?

Как с помощью этой процедуры нарисовать шарик на синем фоне? Как стереть шарик, восстановив синий фон?

Для хранения *текущих* координат шарика (т. е. координат в данный момент) будем использовать переменные `x` и `y` основной программы. Шарик будем двигать слева направо по середине холста на уровне `y` = 100 (рис. 6.61).

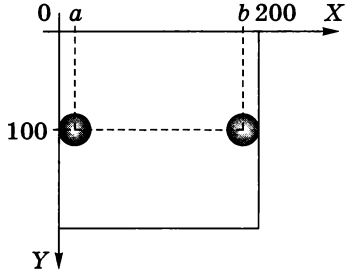


Рис. 6.61

Определите координаты центра шарика, когда он находится: а) у левой стенки; б) у правой стенки (рис. 6.61).

Какие начальные значения нужно установить для переменных *x* и *y*, чтобы шарик начал движение от левой стенки?

Если *x*-координата шарика стала больше, чем 200 – R, где R — радиус шарика, то шарик вышел за правую границу холста. Поэтому цикл движения должен продолжаться, пока x < 200 – R. При R = 10 получается такой цикл:

```
нц пока x < 190
  | движение шарика вправо
кц
```

В теле цикла сейчас только комментарий. Там нужно записать все повторяющиеся операции: вывод шарика на экран, скрытие шарика и его перемещение.

Как нужно вызвать процедуру Шарик, чтобы нарисовать шарик с координатами (x, y)? Стереть шарик с координатами (x, y)?

Как вы думаете, что произойдёт, если нарисовать шарик и сразу же его стереть? Обе эти операции компьютер выполняет очень быстро, поэтому мы даже не заметим, что на экране что-то было нарисовано. Чтобы все-таки увидеть шарик, после того как мы его нарисовали, нужно сделать паузу. Во время этой паузы шарик будет виден на экране. Пауза должна быть небольшая, например можно выбрать её длину 20 миллисекунд (1 миллисекунда = 1/1000 секунды). Добавить такую паузу в программу можно с помощью команды ждать:

```
ждать (20)
```

Остаётся понять, как сдвинуть шарик. Положение шарика определяется координатами его центра, которые хранятся в переменных *x* и *y*. Поэтому для перемещения шарика достаточно изменить значения этих переменных. Например, чтобы передвинуть шарик вправо на 2 пикселя, нужно добавить 2 к значению его *x*-координаты:

```
x:=x+2
```

Таким образом, мы можем написать тело цикла:

```
нц пока x<=190
  Шарик(x, y, жёлтый)
  ждать(20)
  Шарик(x, y, синий)
  x:=x+2
кц
```

Иннокентий и Данила написали свои циклы моделирования так:

Вариант Иннокентия:

```
нц пока x<=190
  Шарик(x, y, жёлтый)
  Шарик(x, y, синий)
  ждать(20)
  x:=x+2
кц
```

Вариант Данилы:

```
нц пока x<=190
  Шарик(x, y, жёлтый)
  ждать(20)
  x:=x+2
  Шарик(x, y, синий)
кц
```

Объясните, почему их программы работают неправильно.

Теперь остаётся «собрать» всю программу. Вы уже можете сделать это самостоятельно. Не забудьте, что после основной программы нужно поместить процедуру Шарик.

Выводы

- Анимация на компьютере состоит в том, что быстрая смена рисунков создаёт иллюзию движения. Каждый из таких рисунков называют кадром.
- Если менять кадры чаще, чем 16 раз в секунду, глаз перестаёт замечать смену изображений и человеку кажется, что движение непрерывное.
- Для перемещения объекта на некотором фоне нужно
  1) «стереть» объект, т. е. восстановить фон в том месте, где он сейчас нарисован;
  2) изменить координаты объекта;
  3) запомнить участок фона, который будет испорчен при выводе объекта в новом месте;
  4) вывести изображение объекта поверх фона.
- Для того чтобы увидеть объект на экране, между моментами его рисования и стирания, нужно сделать паузу.

Интеллект-карта

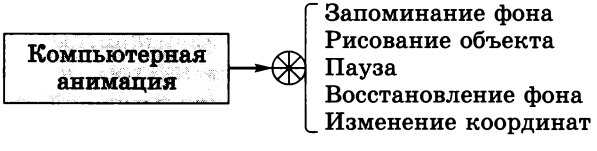


Рис. 6.62

Вопросы и задания

1. Почему в работах профессиональных аниматоров кадры сменяются не реже, чем 16 раз в секунду?
2. Почему обычно перерисовывают не всё изображение на экране, а только ту его часть, которая изменилась?
3. Какие проблемы возникают, если перемещать шарик на фоне картинки?
4. Что такое текущие координаты?
5. Как определить, когда шарик коснётся края холста?
6. Какие команды в программе определяют скорость перемещения шарика?
7. Как можно ускорить движение шарика по экрану? Предложите два метода.
8. Что нужно изменить в программе, чтобы шарик двигался справа налево? Сверху вниз? По диагонали?
9. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Компьютерная анимация в кино и рекламе»

Практическая работа

Выполните практическую работу № 36 «Анимация».

---

§ 45
Управление с помощью клавиатуры

Ключевые слова:

- интерактивность
- символические имена клавиш
- управление с ожиданием
- управление по требованию

Многие компьютерные игры основаны на том, что игрок управляет персонажем с помощью клавиатуры или мыши. Так обеспечивается **интерактивность** — взаимодействие между человеком и компьютером. В этом параграфе вы узнаете, как программа может обрабатывать нажатия клавиш на клавиатуре.

Работа с клавиатурой

Давайте подумаем, какие задачи возникают при управлении игрой с клавиатуры. Во-первых, надо уметь определять, была ли нажата какая-нибудь клавиша. В системе *КуМир* для этого используется логическая команда `сигнал клав` (сигнал клавиши). Результат выполнения этой команды проверки условия — это логическое значение[1]: ответ «да» или «нет». Например, можно использовать такое ветвление:

```
если сигнал клав то
  | что-то сделать
все
```

В этом случае программа будет реагировать на нажатие *любой* клавиши.

Вторая задача — определить, какая именно клавиша была нажата. Каждая клавиша на клавиатуре имеет свой числовой код. Для того чтобы узнать код нажатой клавиши, используется команда `код клав`:

```
цел с
с:= код клав
если с = КЛ_ВВЕРХ то
  | передвинуть объект вверх
все
```

Если ни одна клавиша не была нажата, при выполнении команды `код клав` программа ждёт, когда пользователь нажмёт клавишу.

Чтобы не запоминать числовые коды всех клавиш, для них введены символьные обозначения. Например, коды клавиш-стрелок обозначаются `КЛ_ВВЕРХ`, `КЛ_ВНИЗ`, `КЛ_ВПРАВО` и `КЛ_ВЛЕВО` (все буквы прописные). Это **константы** — неизменяемые величины, которым присвоены имена.

Верно ли, что при вводе символов «я», «Я», «z» и «Z» мы получим один и тот же код клавиши? Исследуйте, изменяется ли код клавиши, если одновременно нажать какие-то из клавиши сдвига: *Shift*, *Ctrl*, *Alt*.

Управление с ожиданием

Сначала научимся управлять каким-то объектом, например изображением шарика (см. предыдущий параграф), **в режиме ожидания**. Это значит, что программа ждёт нажатия клавиши-стрелки, определяет её код и после этого перемещает шарик на экране в нужную сторону.

---

[1] Так же как и для логических команд исполнителя Робот, например, для команды `справа свободно`.

Основной цикл программы можно написать так:

```
нц пока да
  Шарик (x, y, желтый)
  с:=код клав
  Шарик (x, y, синий)
  | переместить шарик
кц
```

Здесь используется цикл с условием **нц пока** да, причём условие (да) всегда истинно, поэтому цикл будет работать бесконечно, пока мы не остановим программу.

Вспомните, что означают два вызова процедуры Шарик в предыдущей программе.

В строке программы

```
с:=код клав
```

мы сказали, что исполнителю нужно:

1) ждать, пока не будет нажата какая-нибудь клавиша;
2) когда клавиша нажата, сохранить её код в переменной с.

Найдите и исправьте логическую ошибку в программе:

```
нц пока да
  Шарик (x, y, жёлтый)
  Шарик (x, y, синий)
  с:=код клав
  | переместить шарик
кц
```

Что произойдёт, если использовать именно этот вариант?

Какие величины, связанные с шариком, нужно изменить в программе для того, чтобы сдвинуть его влево? Вправо? Вверх? Вниз?

Для перемещения шарика нужно изменить его координаты — значения переменных x и y. Причём эти изменения будут зависеть от того, какую клавишу нажал пользователь. Например, можно написать четыре условных оператора:

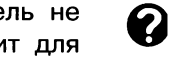
```
если с=КЛ_ВЛЕВО  то x:=x-5 все
если с=КЛ_ВПРАВО то x:=x+5 все
если с=КЛ_ВВЕРХ  то y:=y-5 все
если с=КЛ_ВНИЗ   то y:=y+5 все
```

В итоге получается такая основная программа:

```
использовать Рисователь
алг Управление клавишами
нач
  новый лист(200, 200, синий)
  цел x=100, y=100, c
  перо(1, прозрачный)
  нц пока да
    Шарик(x, y, жёлтый)
    c:=код клав
    Шарик(x, y, синий)
    если c=КЛ_ВЛЕВО  то x:=x-5 все
    если c=КЛ_ВПРАВО то x:=x+5 все
    если c=КЛ_ВВЕРХ  то y:=y-5 все
    если c=КЛ_ВНИЗ   то y:=y+5 все
  кц
кон
```

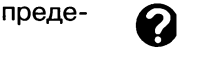
После неё нужно поместить процедуру Шарик из предыдущего параграфа.

Управление по требованию

Что происходило в предыдущей программе, если пользователь не нажимал никакую клавишу? Всегда ли такой вариант подходит для программирования игр? 

Во многих случаях нужен другой режим: события развиваются независимо от того, нажал игрок какую-либо клавишу или нет. Он вмешивается тогда, когда считает нужным. Это **управление по требованию**. Для того чтобы использовать такое управление, надо определить момент, когда пользователь нажимает клавишу, и реагировать на это нажатие. Основной цикл программы можно записать так:

```
нц пока да
  | если нажата клавиша, изменить направление движения
  | нарисовать шарик
  | ждать 20 мс
  | стереть шарик
  | переместить шарик
кц
```

В предыдущем пункте этого параграфа найдите, как можно определить, что пользователь нажал какую-то клавишу. 

Мы напишем программу, в которой шарик постоянно движется, изменяя свое направление при нажатии клавиш-стрелок. Если нажать на пробел (обозначение этой клавиши — КЛ_ПРОБЕЛ), шарик останавливается. Часть программы, где при нажатии клавиш изменяется направление движения, выглядит так:

```
если сигнал клав то
  с:=код клав
  | изменить направление движения
все
```

Что произойдёт, если команда сигнал клав вернёт значение нет (никакая клавиша не нажата)?

Как изменяются координаты шарика при движении в каждом из четырёх направлений?

Поскольку величины, на которые изменяются координаты, тоже изменяются, мы введём новые переменные:

dx — изменение *x*-координаты за один шаг;

dy — изменение *y*-координаты за один шаг.

Каждая переменная (независимо от другой) может быть положительной, отрицательной или равной нулю. Следующие строчки задают перемещение шарика:

```
x:=x+dx
y:=y+dy
```

Куда именно передвинется шарик, зависит от значений dx и dy. Поэтому для изменения направления движения при нажатии клавиши достаточно изменить dx и dy:

```
если с=КЛ_ВЛЕВО  то dx:=-5; dy:=0 все
если с=КЛ_ВПРАВО то dx:=5; dy:=0  все
если с=КЛ_ВВЕРХ  то dx:=0; dy:=-5 все
если с=КЛ_ВНИЗ   то dx:=0; dy:=5  все
если с=КЛ_ПРОБЕЛ то dx:=0; dy:=0  все
```

В последней строке мы останавливаем шарик при нажатии клавиши пробела. Эти команды нужно поставить в основной цикл вместо комментария «изменить направление движения». Полную программу вы можете собрать самостоятельно.

Выводы

- Интерактивность — это взаимодействие между человеком и компьютером.
- Основные операции при управлении с клавиатуры:
  — определить, что какая-то клавиша нажата;
  — определить код нажатой клавиши.
- Существуют два способа управления с клавиатуры:
  — управление с ожиданием, когда программа ждёт нажатия управляющей клавиши;
  — управление по требованию, когда пользователь вмешивается в работу программы только тогда, когда это нужно.
- При анимации движения удобно хранить изменения координат объекта за один шаг в отдельных переменных. Тогда изменение направления движения сводится просто к изменению значений этих переменных.

Интеллект-карта

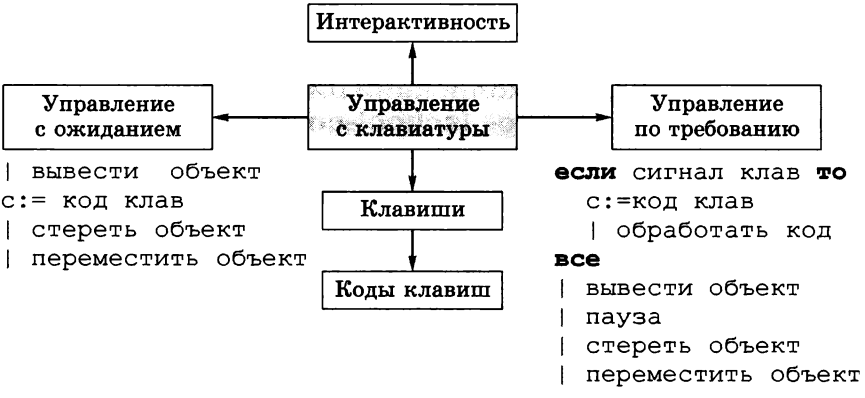


Рис. 6.53

Вопросы и задания

1. Какие две задачи нужно уметь решать для управления объектом с клавиатуры? Какие задачи невозможно решить, если не будет команды `сигнал клав`?
2. Что произойдёт, если при работе программы управления с ожиданием нажата какая-нибудь другая клавиша, кроме стрелок? Как можно устранить этот недостаток?
3. Как можно при нажатии какой-либо клавиши изменять радиус шарика?

4. Как можно при нажатии какой-либо клавиши изменять скорость движения шарика?
5. Зачем в программе управления по требованию использована команда `ждать`?
6. Можно ли в программе управления по требованию переставить оператор **если ... все**, который обрабатывает нажатие клавиши, в другое место в основном цикле? Ответ обоснуйте.
7. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

«Оператор `выбор` в алгоритмическом языке»

Практические работы

Выполните практические работы:
№ 37 «Управление в режиме ожидания»;
№ 38 «Управление по требованию».

ЭОР к главе 6 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Происхождение и определение понятия алгоритма
Исполнитель алгоритма
Алгоритм
Вспомогательные алгоритмы
Нисходящий и библиотечный методы построения сложных алгоритмов
Команда присваивания
Ветвление
Полное и неполное ветвление
Демонстрация алгоритма с вложенным ветвлением «Большее из трех» в среде «Конструктор алгоритмов»
Описание алгоритма Евклида
Покадровая анимация

Глава 7
МУЛЬТИМЕДИА

§ 46
Введение

Ключевые слова:

- мультимедиа
- интерактивность
- устройства мультимедиа
- технологии мультимедиа
- презентация
- слайд
- дизайн
- тема
- палитра
- цветовой круг

Понятие мультимедиа

Как вы знаете, компьютеры были изобретены в первую очередь для того, чтобы ускорить сложные вычисления — обработку числовых данных. Однако текст, рисунки, звуки и видео тоже можно представить как цепочки нулей и единиц. В результате в наше время компьютеры обрабатывают самые различные виды информации, причем в основном именно нечисловые.

Мультимедиа — это использование различных форм представления информации (текст, графика, анимация, звук, видео и т. д.) в одном документе.

Приведите примеры данных, в которых объединены:
а) текст и графика;
б) графика и анимация;
в) графика, анимация и звук.

Часто при использовании мультимедиа человек может влиять на показ материалов: перейти вперёд или вернуться назад, изменить настройки, выбрать один из предложенных вариантов и т. п. Такое взаимодействия человека и компьютера называют **интерактивностью** (взаимной активностью).

Приведите примеры мультимедиа с интерактивностью.

Слово «мультимедиа» также используется в словосочетаниях мультимедиа-компьютер, мультимедиа-носитель, устройства мультимедиа, технологии мультимедиа. К **устройствам мультимедиа** относят устройства, предназначенные для работы с графикой, звуками, видео:

- дисководы для работы с CD- и DVD-дисками;
- видеокарты, содержащие мощные процессоры и оперативную память;
- звуковые карты;
- звуковые колонки;
- микрофон;
- MIDI-клавиатуру для записи музыки в виде нот через специальный разъём звуковой карты;
- *тюнер* — устройство, с помощью которого можно принимать, просматривать и сохранять на компьютере телевизионные сигналы и радиосигналы;
- цифровые фотокамеры и видеокамеры.

Чтобы работать с устройствами, надо знать, как это делается. Тут на помощь приходят технологии.

Технология — способ изготовления некоторого продукта из исходных материалов.

Вот некоторые **технологии мультимедиа**:

- приём и обработка телевизионного сигнала;
- видеозахват — ввод, сохранение в цифровом виде и обработка видеосигнала;
- анимация — «оживление» изображения на экране;
- звуковые эффекты (созданные на компьютере или записанные с помощью микрофона);
- трёхмерная графика (3D-графика);
- виртуальная реальность[1)], позволяющая пользователю погрузиться в мир, созданный с помощью компьютера, и действовать в нём.

---

1) Слово «виртуальный» означает «действующий и проявляющий себя как настоящий».

К **программным средствам мультимедиа** относятся:

- мультимедийные приложения — энциклопедии, интерактивные обучающие курсы, компьютерные игры, тренажёры, рекламные ролики, компьютерные презентации и др.;
- средства создания мультимедийных приложений — редакторы изображений, звука и видеофильмов, программы для создания презентаций.

В этой главе мы научимся создавать мультимедийные компьютерные презентации, с помощью которых можно показывать рисунки и видео во время докладов и выступлений.

Что такое презентация?

Раньше, когда человек выступал перед публикой и хотел показать какие-то картинки, он рисовал большие плакаты и развешивал их вдоль стены. Потом стали использовать рисунки на прозрачной плёнке: с помощью аппарата, который назывался кодоскопом, эти рисунки можно было вывести на экран. Сейчас выступающие готовят набор плакатов на компьютере и выводят их на экран с помощью проектора. Такой набор электронных плакатов называется компьютерной презентацией, а сами плакаты — **слайдами**.

Компьютерная презентация — это набор изображений (слайдов), которые сменяют друг друга по команде человека или через заданные промежутки времени.

Чаще всего для подготовки слайдов используется программа **PowerPoint** из пакета *Microsoft Office* или бесплатная программа **OpenOffice Impress**.

Несколько десятилетий назад слайдом называли кадр фотоплёнки, вставленный в рамку. Слайды вставляли в специальный блок, откуда они выдвигались по одному в проектор и выводились на экран (рис. 7.1).

Рис. 7.1

Слайд в компьютерной презентации может содержать самую разную информацию: текст, рисунки, таблицы, диаграммы, анимацию, видеофильм. В презентацию можно добавлять звук, который проигрывается во время показа одного или нескольких слайдов.

К каждому слайду можно добавить заметки. В заметках вы можете записать примерный текст вашего выступления или комментарии. Во время показа презентации заметки не выводятся на экран.

Слайды можно распечатать (в том числе и в уменьшенном размере, по нескольку слайдов на странице), и тогда у вас будет раздаточный материал для слушателей. Им будет удобнее следить за выступлением и задавать вопросы.

Содержание презентаций

Цель любого докладчика — сообщить какую-то информацию и убедить слушателей в том, что это действительно важно. Для этого необходимо подать материал так, чтобы слушателям было интересно, чтобы они всё поняли и сделали нужные выводы. Презентация обычно состоит из трёх разделов:

- *введения*, где вы рассказываете о теме и её важности;
- *основной части*, содержащей главную информацию (например, предлагаемое решение проблемы, его достоинства и недостатки);
- *заключения*, где нужно подвести итоги и ещё раз сформулировать основные мысли доклада.

Для того чтобы правильно отобрать информацию для презентации, необходимо понять, какую задачу мы хотим решить.

Компьютерная презентация служит для иллюстрации устного выступления.

Разбейтесь на группы по 3–4 человека. Изучите слайд, показанный на рис. 7.2. Подготовьте краткое выступление от группы с ответами на следующие вопросы.

— Что следует говорить докладчику при выступлении по этому слайду?
— Успеют ли зрители прочитать текст во время выступления?
— Как вы оцениваете количество текста на слайде (много, достаточно, мало)?
— Как вы предлагаете улучшить содержание этого слайда?

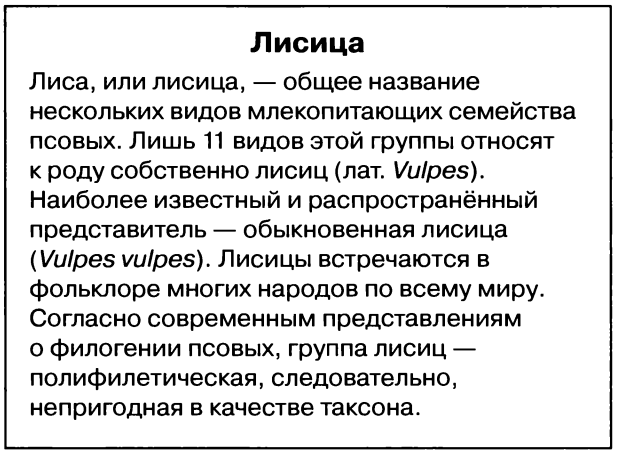


Рис. 7.2

Презентация не заменит электронный учебник, справочник, энциклопедию, веб-сайт, потому что это другой жанр. Не нужно выводить на слайды весь текст выступления. Если вы хотите, чтобы те, кто открыл презентацию, смогли прочитать текст, добавьте его в заметки к каждому слайду. Люди читают текст на слайдах быстрее, чем вы говорите, и если информация будет дублироваться, они перестанут вас слушать. Добавлять в презентацию много текста бессмысленно, его всё равно никто не будет читать.

Не пытайтесь включить в презентацию много информации. Наоборот, всё лишнее, что не «работает» на вашу цель, нужно убрать. Если вы думаете, что какая-то информация может понадобиться во время ответов на вопросы, сделайте дополнительные скрытые слайды. Они не будут показаны во время вашего доклада, но вы всегда сможете перейти к ним, если это будет необходимо.

Дизайн презентации

Для презентации важно не только содержание, но и оформление. На первых порах можно воспользоваться готовыми стилями **(темами)** оформления, которые разработаны профессиональными дизайнерами. В *PowerPoint* меню для выбора темы находится на вкладке *Дизайн* (рис. 7.3).

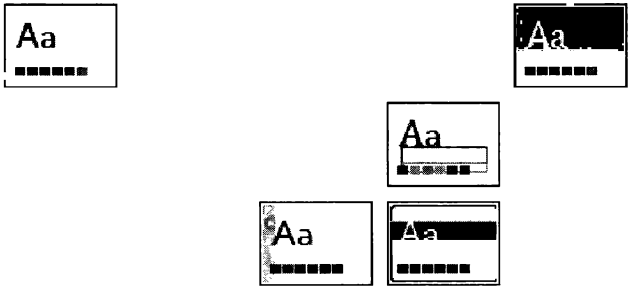


Рис. 7.3

Для каждой темы вы можете выбрать цветовую палитру, набор шрифтов, фон.

Опытные авторы презентаций редко используют готовые темы, а обычно создают свои. Одна из важных задач, которую при этом приходится решать, — выбор цветов для фона и текста. Гармоничные сочетания цветов изучает теория цвета. Для выбора палитры применяют цветовой круг (рис. 7.4 и цветной рисунок на форзаце).

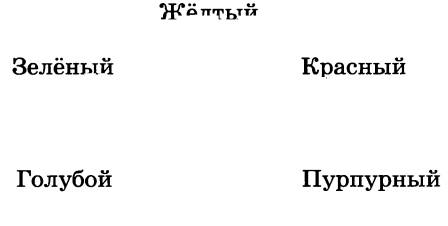


Рис. 7.4

Если нужно выбрать два цвета, обычно берут цвета, расположенные напротив друг друга, например красный и голубой, жёлтый и синий. Три цвета выбирают в вершинах равностороннего треугольника, например красный, синий и зелёный. Можно выбирать цвета в вершинах квадрата и других равносторонних фигур (пятиугольника, шестиугольника и т. п.).

Не забывайте, что есть ещё чёрный и белый цвета, которые очень хорошо сочетаются со всеми остальными. Дизайнер Роджер Блэк даже сформулировал такую мысль: «Первый цвет — белый, второй — чёрный, третий — красный». Действительно, чёрный и белый — это самый тёмный и самый яркий цвета, с ними отлично сочетается красный, его можно использовать для выделения. Хороший контраст также получается при использовании чёрного цвета рядом с жёлтым. Так, например, обозначается опасная зона (см. цветной рисунок на форзаце).

Работая в группах, сравните два слайда на рис. 7.5 (см. цветной рисунок на форзаце). Ответьте на следующие вопросы.

— Какую задачу решает выделение цветом в обоих случаях?
— На каком слайде информация воспринимается легче?
— Какие рекомендации вы можете дать по выбору цветов слайда?
— Как бы вы рекомендовали улучшить каждый из слайдов?

а)

б)

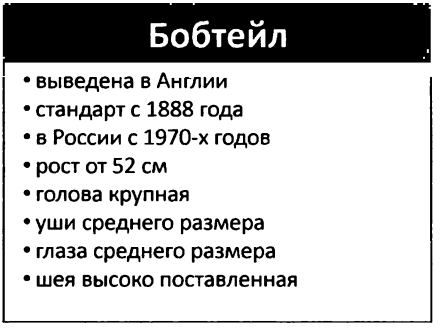


Рис. 7.5

Слайд не должен выглядеть разноцветным, как попугай. Желательно использовать не более трёх-четырех цветов (для фона, заголовков, обычного и выделенного текста).

Выводы

- Мультимедиа — это использование различных форм представления информации в одном документе. Примеры применения технологии мультимедиа — видеофильмы, компьютерные презентации, игры.
- Компьютерная презентация — это набор изображений (слайдов), которые сменяют друг друга по команде человека или автоматически через заданные промежутки времени.

- Компьютерная презентация служит для иллюстрации устного выступления. Главная задача презентации — донести информацию до слушателей.
- Слайд может содержать текст, рисунки, таблицы, диаграммы, анимацию, звук, видеофильм. К слайдам можно добавлять текстовые заметки.
- В презентации обычно используют не более 3–4 цветов. Одних из лучших трёхцветных наборов — чёрный, белый и красный.

Интеллект-карта

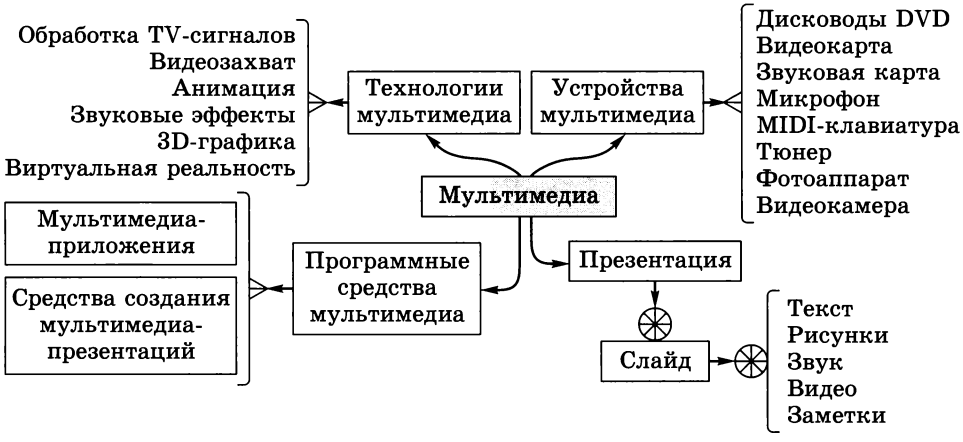


Рис. 7.6

Вопросы и задания

1. Как вы думаете, можно ли использовать компьютерные презентации как самостоятельные документы? Почему? Обсудите этот вопрос в классе.
2. Как вы считаете, нужно ли писать план (сценарий) презентации? Ответ обоснуйте.
3. Почему не рекомендуют размещать на слайдах много текста? Согласны ли вы с этим мнением?
4. Как можно использовать заметки к слайдам?
5. Предложите варианты использования скрытых слайдов.
6. Проверьте, как готовые темы оформления в вашей программе для подготовки презентаций будут смотреться через проектор. Нравится ли вам результат?

7. Найдите в Интернете бесплатные презентации на интересующую вас тему, обсудите в классе, как они оформлены.
8. Выберите тему, на которую вы хотели бы сделать презентацию. Составьте план презентации из 5–6 слайдов, записав названия слайдов.
9. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Эмоциональный и деловой стиль в презентациях»
б) «Презентации, которые делал Стив Джобс»
в) «Польза и вред презентаций»

Интересные сайты

color.adobe.com/ru/create/color-wheel/ — подбор цветовой палитры
paletton.com — подбор цветовой палитры

Практическая работа

Выполните практическую работу № 39 «Анализ презентаций».

§ 47
Работа со слайдом

Ключевые слова:

- слайд
- макет
- выравнивание
- шрифт
- список
- фон
- контрастность
- звук
- видео

Обычно презентации содержат несколько слайдов. Но мы начнём с простой задачи — сначала научимся грамотно оформлять один-единственный слайд. В любой момент можно проверить, как выглядит слайд в режиме просмотра (на полном экране), нажав на клавишу *F5*.

Макеты

При создании слайда можно использовать готовые макеты или делать всё вручную. **Макеты** — это готовые варианты размещения информации на слайде.

Разбейтесь на группы по 3–4 человека. Каждая группа после нескольких минут обсуждения представляет несколько вариантов расположения материала на слайде:

1) название презентации и имя автора;
2) заголовок слайда, текст и рисунок;
3) заголовок, текст и два рисунка.

В программах для создания презентаций есть готовые макеты, разработанные опытными специалистами. Вот такие макеты предлагает программа **Impress** (рис. 7.7).

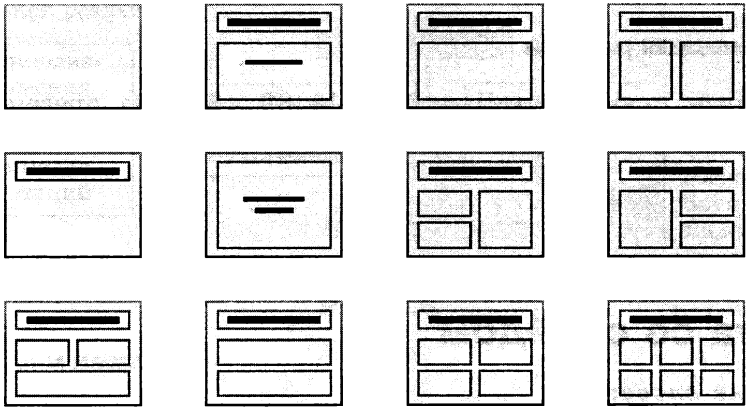


Рис. 7.7

Первый макет в верхнем ряду — пустой, если выбрать его, все элементы придётся добавлять вручную. На остальных явно выделяется заголовок и одна или несколько областей для элементов слайда. В каждую область можно вставить текст, рисунок, диаграмму, видео. Размеры областей можно менять, перетаскивая маркеры на углах рамок.

Выбрав один из макетов, мы увидим приглашение к вводу данных в каждой области (рис. 7.8).

Рис. 7.8

Текст, который мы сейчас видим, исчезнет, когда мы введём что-то своё.

Кроме текста в каждый блок (кроме заголовка) мы можем вставить таблицу, диаграмму, рисунок или видеофильм.

Размещение элементов слайда

Обсудите в классе, верны ли следующие высказывания.

а) Основная задача слайда — передать информацию.

б) Оформление слайда не влияет на восприятие информации на слайде.

в) Можно добавлять на слайд рисунки «для красоты».

Главная задача презентации — донести информацию.

Вы должны сделать всё, чтобы облегчить зрителям получение информации. Лишние элементы слайда – это информационный шум, он не улучшает восприятие, а только усложняет его.

Человек не может воспринимать много информации одновременно, поэтому обычно на слайд добавляют не более 7 элементов. Мелкие детали заставляют людей напрягать зрение, поэтому лучше их не использовать. Хорошо, если на слайде будут только три ведущих объекта. Если на слайде оказалось больше 7–9 объектов, лучше подумайте о том, чтобы сделать один дополнительный слайд.

Сравните два слайда на рис. 7.9 (см. цветной рисунок на форзаце) и ответьте на вопросы.
— Чем отличается расположение элементов?
— На каком слайде легче воспринимать информацию?
— В каком месте слайда нужно располагать самую важную информацию: в центре или по краям?

а)

б)

Рис. 7.9

Сравните два слайда на рис. 7.10 (см. цветной рисунок на форзаце) и ответьте на вопросы.
— Чем отличается расположение элементов?
— На каком слайде легче воспринимать информацию?
— Какие рекомендации вы можете дать?

а)

б)

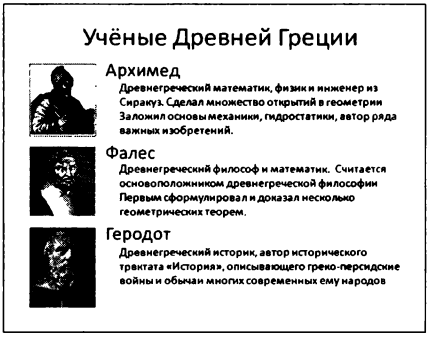


Рис. 7.10

Элементы на слайде не должны быть разбросаны в произвольном порядке, их нужно зрительно связывать друг с другом. Для этого используется выравнивание по вертикали и горизонтали. Выровненные элементы образуют в сознании человека единое

целое: невидимая линия «связывает» их, даже если они находятся на некотором расстоянии друг от друга. Для выравнивания вы можете использовать встроенные возможности программ для подготовки презентаций:

- *горизонтальное выравнивание*: по левой или правой границе объектов, по центру;
- *вертикальное выравнивание*: по верхней или нижней границе объектов, по середине;
- *распределение по горизонтали и вертикали* (на равных расстояниях друг от друга).

Оформление текста

Для того чтобы текст был виден с задних рядов аудитории, его размер делают не менее 24 пунктов (напомним, что 1 пункт = = 1/72 дюйма, а 1 дюйм = 2,54 см). Заголовки слайдов делают крупнее, чем подзаголовки и основной текст слайда, ведь их должны заметить первыми.

Размер шрифта для каждого типа элементов должен быть одинаковым на всех слайдах, например, 32 пункта для заголовков и 24 пункта для текста.

Для презентаций обычно выбирают шрифты без засечек (рубленые), например **Arial**, **Calibri**, **Helvetica**. Дело в том, что текст, набранный шрифтом с засечками (чёрточками на верхних и нижних концах букв), на расстоянии сливается в одну массу, и отдельные буквы трудно разобрать. В презентации обычно используют не более двух названий шрифтов (**гарнитур**).

Часто текстовая информация на слайде оформляется как список:

а)

Семейство псовые

Подсемейства:

- Волчьи
- Собачьи
- Большеухие лисицы

б)

Этапы моделирования

1. Постановка задачи
2. Разработка модели
3. Тестирование модели
4. Эксперимент с моделью.
5. Анализ результатов.

Рис. 7.11

Сравните два слайда на рис. 7.12[1)] и ответьте на вопросы.
— Чем отличается форматирование текста на слайдах?
— На каком слайде легче воспринимать информацию?
— Какие недостатки оформления вы увидели?
— Как можно улучшить форматирование слайдов?

а)

Задачи

Задача 1.
В бублике 1 дырка, а в кренделе в два раза больше. Насколько меньше дырок в 7 бубликах, чем в 12 кренделях?

Задача 2.
Площадь одного уха слона равна 10000 кв. см. Узнай в кв. м площадь ушей 12 одинаковых слонов.

б)

Задачи

Задача 1.
В бублике 1 дырка, а в кренделе в два раза больше. Насколько меньше дырок в 7 бубликах, чем в 12 кренделях?

Задача 2.
Площадь одного уха слона равна 10000 кв. см. Узнай в кв. м площадь ушей 12 одинаковых слонов.

Рис. 7.12

В книгах текст выравнивается по ширине (т. е. выравниваются левая и правая границы). Этот приём хорошо работает, если строки достаточно длинные. В презентациях мы используем более крупный шрифт, и при выравнивании по ширине получаются очень большие «дырки» между словами, поэтому нужно использовать выравнивание по левой границе.

Сравните две пары слайдов на рис. 7.13 и рис. 7.14. Для каждой пары ответьте на вопросы.
— Чем отличается форматирование текста на слайдах?
— На каком слайде легче воспринимать информацию?
— Какие недостатки оформления вы увидели?
— Как можно улучшить форматирование слайдов?

[1)] Задачи из книги: *Остер Г. Б.* Задачник. Наглядное пособие по математике.

а)

Белуха

Гора Белуха – наивысшая точка Горного Алтая. Ее высота – 4509 м. Здесь берёт своё начало река Катунь. Название «Белуха» происходит от обильного снега, покрывающего гору от вершины до самого основания.

б)

Белуха

Гора Белуха – наивысшая точка Горного Алтая. Ее высота – 4509 м. Здесь берёт своё начало река Катунь. Название «Белуха» происходит от обильного снега, покрывающего гору от вершины до самого основания.

Рис. 7.13

а)

Белуха

- наивысшая точка Горного Алтая
- высота – 4509 м
- начало реки Катунь
- «Белуха» – от обильного снега, покрывающего всю гору

б)

Белуха

- наивысшая точка Горного Алтая
- высота – 4509 м
- начало реки Катунь
- «Белуха» – от обильного снега, покрывающего всю гору

Рис. 7.14

Выравнивание по центру используют только для заголовков. Недопустимо выравнивать по центру длинные тексты и тем более списки. Дело в том, что ровная левая граница и маркеры служат для того, чтобы было легко найти начало следующей строки. При выравнивании по центру в каждой строке текст начинается в разных местах, и найти начало строки намного сложнее.

Сравните слайды на рис. 7.15 и ответьте на вопросы.

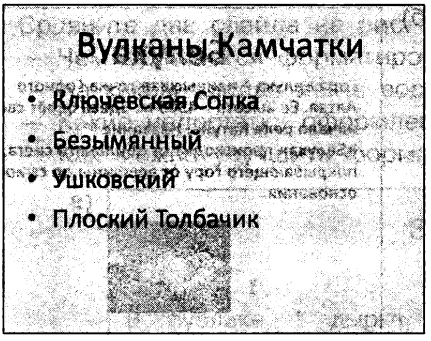
— Чем отличается форматирование текста на слайдах?

— На каком слайде легче воспринимать информацию?

— Какие недостатки оформления вы увидели?

— Как можно улучшить форматирование слайдов?

а)



б)

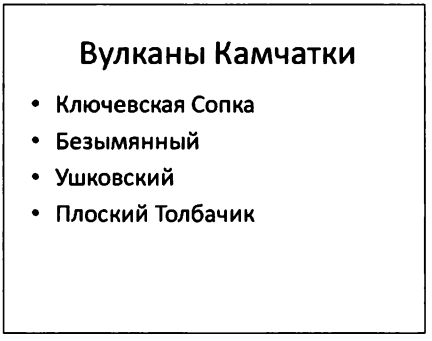


Рис. 7.15

Цвета фона и текста нужно выбирать так, чтобы они были **контрастными**, т. е. резко отличались друг от друга. Для того чтобы проверить контрастность цветов, можно перевести слайд в чёрно-белый вариант и сравнить тон пикселей текста и фона. Часто проектор искажает цвета и снижает контраст, поэтому получается, что на экране монитора текст виден хорошо, а на большом экране — плохо. Чтобы этого не произошло, контраст нужно выбирать «с запасом».

Сравните три слайда на рис. 7.16 и 7.17 (см. цветной рисунок на форзаце) и ответьте на вопросы.

— Чем различается дизайн этих слайдов?
— На каком слайде легче воспринимать информацию?
— Какие недостатки оформления вы увидели?
— Как можно улучшить форматирование слайдов?

а)

б)

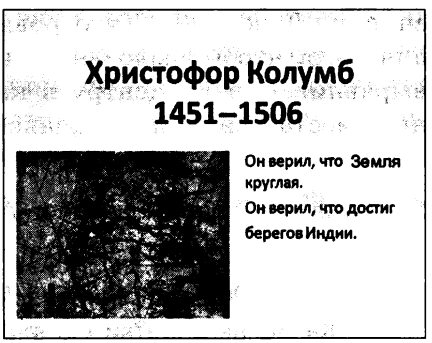


Рис. 7.16

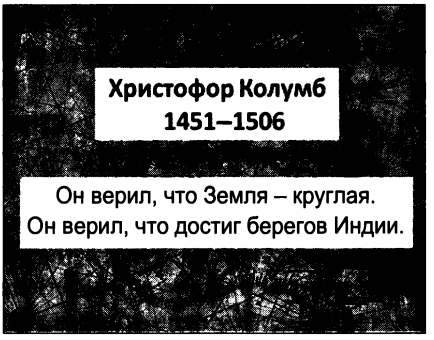


Рис. 7.17

Авторы многих презентаций любят делать фоном какую-нибудь картинку, и часто она мешает читать текст. Поэтому профессиональные дизайнеры, как правило, выбирают одноцветный фон. В крайнем случае, если для чего-то очень нужно оставить фоновый рисунок, можно подложить под текст так называемые «плашки» — одноцветные прямоугольники.

Как проверить, правильно ли оформлен ваш слайд? Задайте себе несколько вопросов и убедитесь, что на все эти вопросы ответ — «да».

— На слайде не более 7–9 объектов?
— На слайде есть поля?
— Элементы на слайде выровнены по вертикали и горизонтали?
— Текст хорошо читается издалека? Даже при показе через проектор?
— Рисунки и фон не мешают воспринимать информацию?

Добавление объектов

Часто элементов, которые уже есть в выбранном макете слайда, не хватает. Тогда приходится добавлять новые. В *PowerPoint* для этого используется вкладка *Вставка* (рис. 7.18).

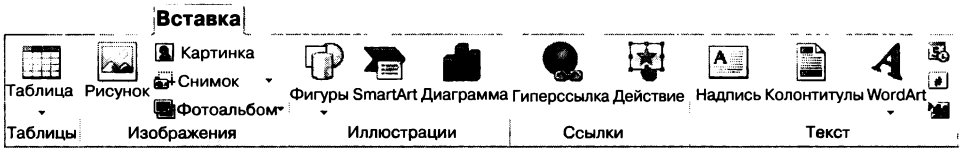


Рис. 7.18

С её помощью можно добавить на слайд таблицы, рисунки, клипы из коллекции, надписи, векторные фигуры, диаграммы, звуки, видео. В программе *Impress* для этой цели служит меню *Вставка*.

Работа с **таблицами** происходит так же, как и в текстовом процессоре. Таблицы, как и все объекты на слайде, можно перемещать за рамку в любое нужное место. Их размеры изменяются с помощью маркеров на рамке.

При вставке **рисунка** программа предложит выбрать файл на диске. Кроме того, можно вставить рисунок (а также текст и таблицу) через буфер обмена из другой программы.

При добавлении **диаграммы** на слайд в программе *PowerPoint* появляется окно программы *Excel*, где нужно ввести данные, по которым строится диаграмма. В программе *Impress* для изменения данных нужно щёлкнуть правой кнопкой мыши на диаграмме и выбрать пункт *Таблица данных диаграммы* в контекстном меню.

При вставке **звука** вы выбираете файл на диске и настраиваете свойства звука:

- способ запуска: автоматически (при показе слайда) или по щелчку;
- момент окончания: по щелчку, после этого слайда или после другого слайда (последний вариант позволяет сделать в *PowerPoint* музыкальное сопровождение на несколько слайдов).

В современных версиях *PowerPoint* можно сделать обрезку звукового фрагмента.

Возможности программы *Impress* значительно скромнее, там звук всегда запускается автоматически и заканчивается при переходе к новому слайду.

При добавлении **видео** можно настроить размеры области показа и переместить её в нужное место. В *PowerPoint* можно установить режим показа по щелчку и развернуть видео на весь экран. В программе *Impress* видео всегда запускается автоматически.

Если вы используете звук и видео, нужно помнить про две особенности. Во-первых, для того чтобы прослушать звук и просмотреть видеофайлы, нужно установить используемые в них **программы-кодеки** (кодировщики/декодировщики). Может случиться так, что на вашем компьютере нужные кодеки есть, а на компьютере, где вы будете показывать презентацию, — нет, и по-

этому слушатели ничего не услышат и не увидят. С этой точки зрения лучше всего использовать форматы WAV и MP3 для звука и форматы WMV и MPEG для видео. Эти кодеки установлены на большинстве компьютеров. Во-вторых, звуковые и видеофайлы не всегда сохраняются внутри файла с презентацией[1)]. Вместо этого программа устанавливает ссылки на их текущее место на диске и загружает их в память тогда, когда это необходимо. Поэтому при переносе презентации на другой компьютер необходимо скопировать не только файл с презентацией, но и все мультимедийные файлы. Лучше всего, если все эти файлы будут находиться в одном каталоге.

Выводы

- Главная задача презентации — донести информацию до слушателей.
- Макеты — это готовые варианты размещения элементов на слайде.
- Лучше всего размещать от 3 до 7 элементов на слайде.
- Со всех сторон слайда нужно оставлять поля.
- Элементы слайда должны быть выровнены по вертикали и/или по горизонтали.
- Для презентаций обычно выбирают шрифты без засечек (рубленые), потому что они лучше читаются издалека.
- Маркированный список используется для описания множеств, порядок перечисления элементов не важен. Нумерованный список показывает последовательность, например порядок действий, в нём порядок элементов изменять нельзя.
- Текст в узких колонках выравнивают по левой границе (не по ширине).
- Выравнивание по центру используют только для заголовков. Недопустимо выравнивать по центру длинные тексты и списки.
- Цвета фона и текста нужно выбирать так, чтобы они были контрастными. Фоновый рисунок не должен мешать чтению текста.
- На слайде можно размещать текст, таблицы, диаграммы, звук, видео.

[1)] В современных версиях *PowerPoint* звуковые и видеофайлы внедряются в презентацию.

Интеллект-карта

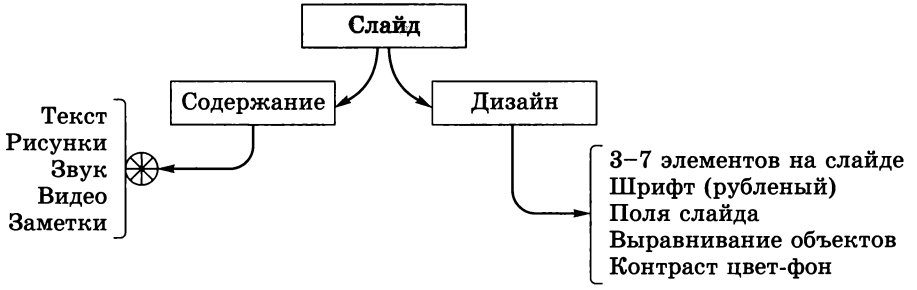


Рис. 7.19

Вопросы и задания

1. Предложите какой-нибудь алгоритм выбора макета для слайда. Этот алгоритм будет линейным, разветвляющимся или циклическим?
2. Обсудите достоинства и недостатки пустого макета, в котором всё приходится делать вручную. Почему его достаточно часто используют?
3. Как вы понимаете выражение «информационный шум»? Почему от него нужно избавляться?
4. Какие проблемы могут возникнуть при использовании звука и видео? Как их решать?
5. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Типичные ошибки в оформлении презентаций»
б) «Как люди воспринимают информацию на слайде?»

Интересные сайты

artlebedev.ru/kovodstvo/ — проект «Ководство» А. Лебедева (заметки о дизайне)
blog.powerlexis.ru — блог, посвящённый дизайну презентаций

Практическая работа

Выполните практическую работу № 40 «Визитная карточка».

Проект

Исследуйте, как зависит максимальное расстояние, на котором текст на экране хорошо виден, от размера шрифта. Сравните

рубленые шрифты и шрифты с засечками. Оформите полученные данные в виде таблицы. Будут ли зависеть результаты от размера экрана?

§ 48 Анимация

Ключевые слова:

- анимация
- последовательное появление
- вход
- выход
- выделение
- перемещение
- настройка анимации

Анимация — это «оживление» изображения на экране. В этом параграфе мы научимся *анимировать* (заставлять изменяться, двигаться) объекты в презентации и разберёмся, когда нужно и когда не нужно использовать анимацию.

Когда нужна анимация?

Обсудите в классе следующие утверждения.

а) Анимация — это всегда хорошо и интересно.

б) Анимация должна облегчать восприятие информации.

в) Анимация сильно отвлекает внимание слушателей.

Существует всего несколько ситуаций, когда анимация оправданна.

Последовательное появление элементов. Если слайд содержит много элементов, нет смысла показывать все элементы сразу. Лучше, если они будут появляться последовательно, как будто вы пишете и рисуете на доске во время выступления. Появляется новый объект, и вы начинаете про него рассказывать.

Представьте себе, что вы просите слушателей ответить на какие-то вопросы, и на экран выводятся по одному варианты ответов. После того как вы вместе выясните, что ответ неправильный, его можно перечеркнуть красной линией или крестом (рис. 7.20).

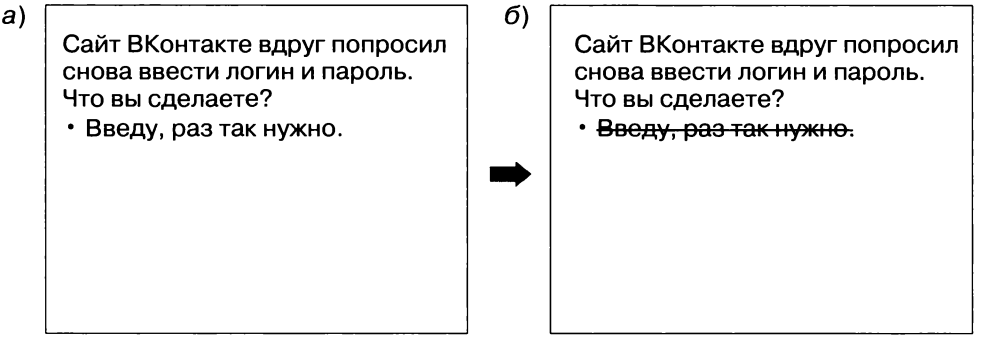


Рис. 7.20

Перечислите шаги анимации, показанной на рис. 7.20.

Иногда нужно временно вывести на экран какой-то вопрос или пример, а потом его убрать (рис. 7.21).

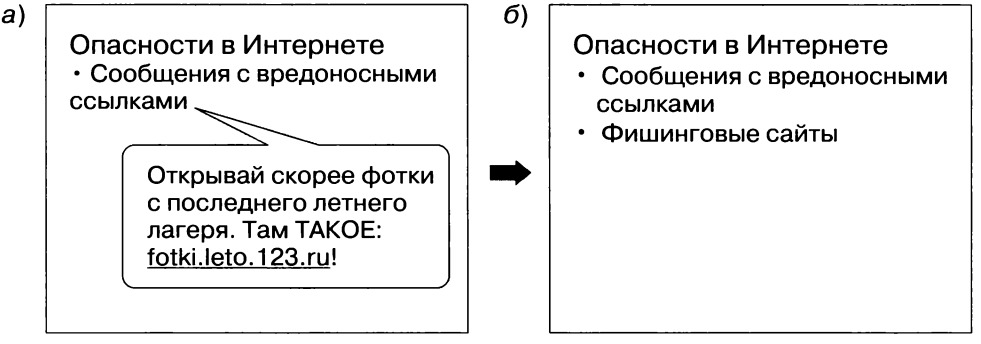


Рис. 7.21

Перечислите шаги анимации, показанной на рис. 7.21.

Установка элемента на своё место. Допустим, вы хотите крупно показать несколько рисунков на одну тему и затем оставить на экране их уменьшенные копии. В этом случае можно сначала выводить крупный рисунок, а затем перемещать его на постоянное место, одновременно уменьшая размеры (рис. 7.22).

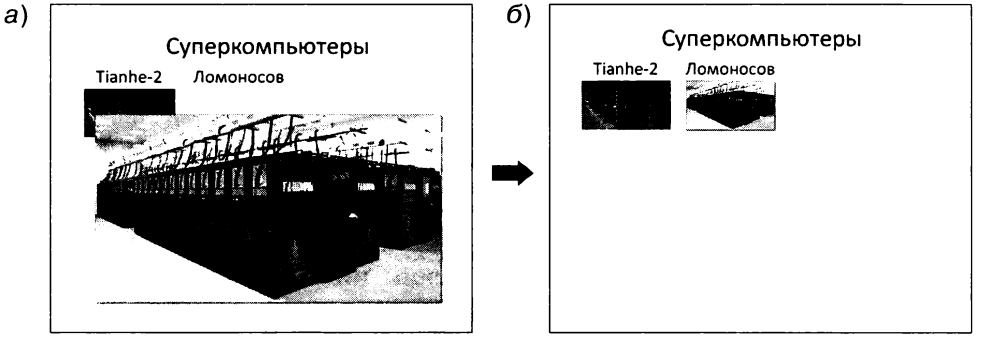


Рис. 7.22

Перечислите шаги анимации, показанной на рис. 7.22.

Иллюстрация процесса. Если вам нужно наглядно показать какие-то изменения, анимация поможет решить эту задачу. Например, вы рассказываете о том, как бильярдный шар отскакивает от стенки поля. Можно, конечно, просто нарисовать его путь в виде линий, но анимация позволит слушателям легче понять, что происходит.

Настройка анимации

В *PowerPoint* анимация выполняется с помощью панели *Настройка анимации* (вкладка ленты *Анимация*), а в программе *Impress* — с помощью панели *Эффекты*.

К каждому объекту можно применить несколько типов анимации (и по нескольку раз!):

- *вход* (появление объекта на экране);
- *выход* (исчезновение объекта);
- *выделение* (изменение свойств объекта);
- *перемещение* (по прямой или по нарисованной траектории).

На рис. 7.23 показана панель настройки анимации в *PowerPoint*.

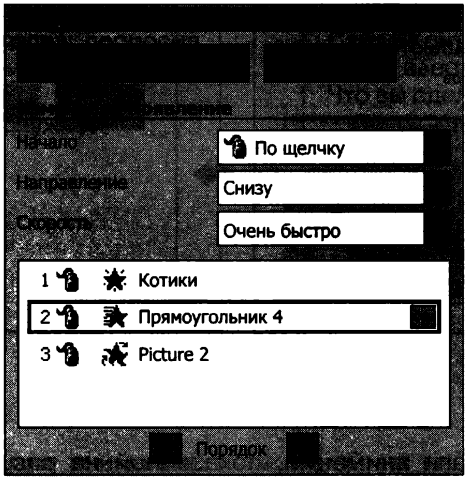


Рис. 7.23

Для каждого типа анимации предусмотрено множество эффектов. Для входа объектов часто используют эффекты *Растворение* (на месте) или *Появление* (постепенное прорисовывание в одном направлении, например, слева направо).

Для текста можно установить анимацию по абзацам, например чтобы выводить элементы списка по одному.

Исследуйте различные варианты анимации входа, выхода, выделения и перемещения.

Изучите назначение списков *Начало*, *Направление*, *Скорость*. Попробуйте выполнить несколько видов анимации одновременно.

Проверьте, может ли объект участвовать в анимации несколько раз.

Выводы

- Анимация должна помогать слушателям воспринимать информацию.
- Анимация сильно отвлекает внимание, поэтому нужно использовать её с осторожностью.
- Анимация полезна в особых случаях:
  - — последовательное появление элементов;
  - — установка элементов на своё место;
  - — иллюстрация процесса (например, движения).

- К каждому объекту можно применить несколько типов анимации (и по нескольку раз!):
  — вход (появление объекта на экране);
  — выход (исчезновение объекта);
  — выделение (изменение свойств объекта);
  — перемещение (по прямой или по нарисованной траектории).
- Следующий этап анимации может начинаться по щелчку мышью, автоматически после завершения предыдущего этапа или одновременно с предыдущим этапом анимации.

Интеллект-карта

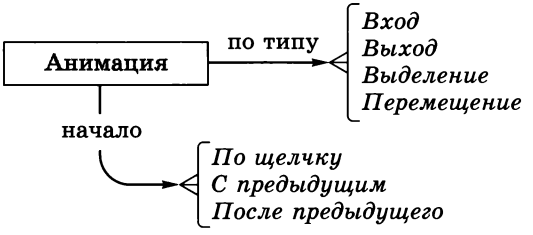


Рис. 7.24

Вопросы и задания

1. Предложите другие случаи, кроме описанных в параграфе, когда полезно использовать анимацию.
2. Придумайте ситуации, когда нужно использовать анимацию типа *Выделение*.
3. Приведите примеры неверного использования анимации, когда зрители только отвлекаются и ухудшается передача информации.
4. Приведите примеры, когда полезны режимы анимации *По щелчку*, *Вместе с предыдущим*, *После предыдущего*.
5. Найдите в Интернете презентации с анимацией. Обсудите в классе, насколько оправданно использование анимации в них.
6. Выполните по указанию учителя задания в рабочей тетради.

Подготовьте сообщение

а) «Когда анимация не нужна?»
б) «Триггеры в презентациях»

Практическая работа

Выполните практическую работу № 41 «Анимация».

Проект

Сделайте анимированный ролик на любую тему на одном слайде. Примените разные эффекты анимации. Обсудите достоинства и недостатки вашей работы с напарником или в классе.

§ 49 Презентации с несколькими слайдами

Ключевые слова:

- переход между слайдами
- скрытые слайды
- сортировщик слайдов
- показ презентации
- репетиция

Добавление нового слайда

Для добавления нового слайда (после текущего, т. е. того, с которым мы работаем) в *PowerPoint* используется кнопка *Создать слайд* на панели *Главная* (или клавиши *Ctrl+M*), а в программе *Impress* — пункт главного меню *Вставка* → *Слайд*.

В левой части окна программы находится панель *Слайды* (рис. 7.25).

Рис. 7.25

Создайте несколько простых слайдов, на каждый из них поместите надпись с номером. Используя окно *Слайды*, попробуйте:

- перетащить слайд мышью в другое место;
- перетащить слайд при нажатой клавише Ctrl (что получилось?);
- выделить слайд и нажать клавишу Delete (что получилось?).

Исследуйте контекстное меню окна *Слайды*. Какие операции можно выполнить с его помощью?

Некоторые слайды можно скрыть. Скрытые слайды не будут показаны во время просмотра презентации, но вы можете обратиться к ним при ответах на вопросы.

Все слайды презентации должны быть оформлены в одном стиле. Если оформление разных слайдов отличается, слушатели будут отвлекаться на то, чтобы рассмотреть новые цвета и элементы. Заголовки должны быть расположены в одних и тех же местах, нужно использовать на всех слайдах одинаковые шрифты (сохраняя их размеры для заголовков и основного текста).

Переходы между слайдами

Вы можете определить эффекты, которые происходят при замене одного слайда другим. Они называются **переходами**. В программе

PowerPoint для этого используется вкладка *Переходы* (рис. 7.26), а в *Impress* — панель *Смена слайдов*.

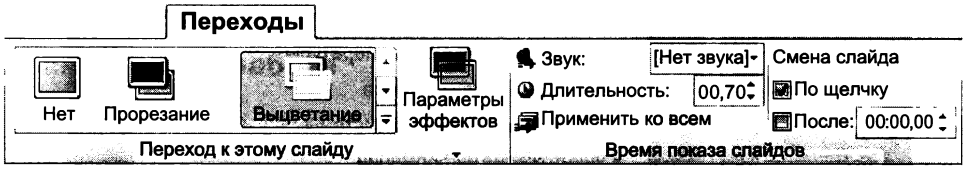


Рис. 7.26

Как видно из рис. 7.26, для выделенного слайда можно выбрать один из многочисленных эффектов перехода, сопровождающий звук, скорость перехода. Кнопка *Применить ко всем* позволяет установить этот режим перехода для всех слайдов.

Выясните, какие способы смены слайдов можно выбрать.

В каких ситуациях при выступлении докладчика полезно ручное переключение слайдов?

Сортировщик слайдов

Для того чтобы «охватить взглядом» всю презентацию, можно включить режим *Сортировщик слайдов* (рис. 7.27). В *PowerPoint* для этого нужно щёлкнуть на кнопке в правом нижнем углу окна, а в *Impress* — перейти на вкладку *Сортировщик слайдов*.

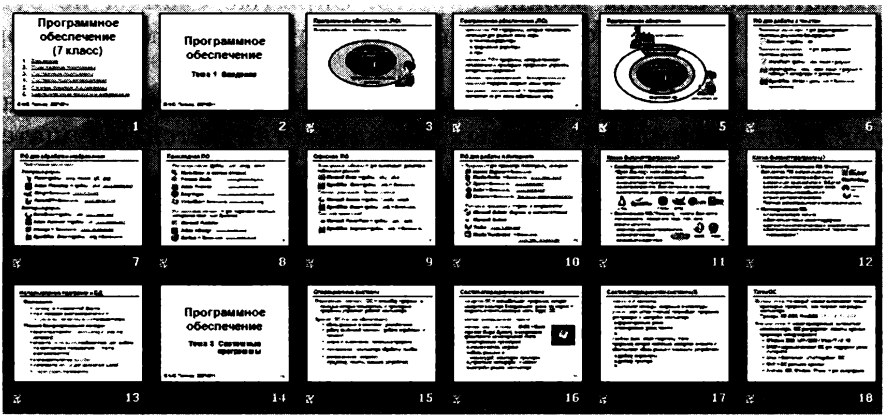


Рис. 7.27

Используя известные вам приёмы, в том числе контекстное меню, попробуйте в режиме *Сортировщик слайдов*:

- переместить слайд;
- скопировать слайд;
- удалить слайд;
- выделить несколько соседних слайдов (как в файловых менеджерах);
- выделить несколько слайдов, расположенных в разных местах.

Показ презентации

При нажатии кнопки *F5* презентация запускается с самого начала. Можно запустить её и с текущего (рабочего) слайда. Для этого в *PowerPoint* используются клавиши *Shift+F5* или кнопки на панели *Показ слайдов*. В программе *Impress* можно выбрать начальный слайд с помощью меню *Демонстрация* → *Параметры демонстрации*.

При желании можно сделать настраиваемую презентацию: выбрать отдельные слайды для показа и расставить их в нужном порядке, не меняя расположения слайдов в самой презентации.

Для управления ходом презентации используются клавиши и мышь:

- пробел или щелчок левой кнопкой мыши — просмотр следующего слайда;
- клавиши *PgUp* и *PgDown*, клавиши-стрелки — переходы вперёд и назад.

В обеих программах есть возможность провести *репетицию* презентации и записать время, необходимое для показа каждого слайда. Это удобно, когда нужно сделать презентацию с автоматической сменой слайдов по времени.

Выводы

- Все слайды презентации должны быть оформлены в одном стиле.
- Некоторые слайды можно скрыть. Скрытые слайды не будут показаны во время просмотра презентации, но вы можете обратиться к ним при ответах на вопросы.
- Смена слайдов выполняется по щелчку мышью или автоматически через установленный промежуток времени.
- Переходы — это эффекты при замене одного слайда другим.

Интеллект-карта

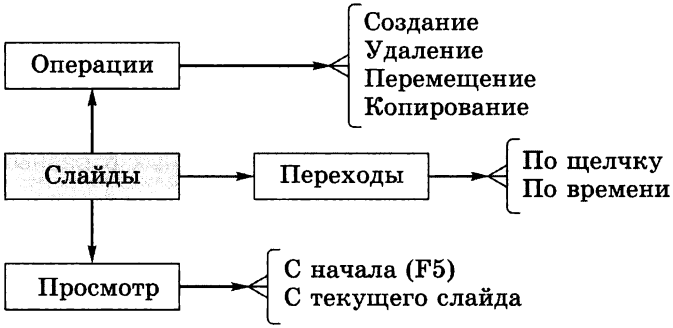


Рис. 7.28

Вопросы и задания

1. Каким инструментом, на ваш взгляд, удобнее пользоваться для управления слайдами: сортировщиком слайдов или панелью *Слайды* слева от области редактирования слайда? Обсудите достоинства и недостатки обоих инструментов.
2. Выполните по указанию учителя задания в рабочей тетради.

Практическая работа

Выполните практическую работу № 42 «Презентация. Проект».

Проекты

1. Самостоятельно или в составе группы создайте презентацию из нескольких слайдов на выбранную тему. Обсудите в классе результаты вашей работы. Исправьте обнаруженные недостатки.
2. Постройте слайд-шоу из своих фотографий (например, сделанных во время каникул). Презентация должна работать в автоматическом режиме.

ЭОР к главе 7 из Единой коллекции цифровых образовательных ресурсов (school-collection.edu.ru)

Интерфейс программы PowerPoint
Создание новой презентации в PowerPoint
Создание слайда в PowerPoint
Изменение оформления слайдов в PowerPoint
Работа с объектами в PowerPoint
Настройка анимации и звука в PowerPoint
Демонстрация презентации в PowerPoint

ОГЛАВЛЕНИЕ