№3 Издание основано в 1995 г. inf.1september.ru
УЧЕБНО-МЕТОДИЧЕСКИЙ ЖУРНАЛ ДЛЯ УЧИТЕЛЕЙ ИНФОРМАТИКИ

ИНФОРМАТИКА

20 лет

электронная версия журнала
дополнительные материалы в Личном кабинете на сайте www.1september.ru

издательский дом 1september.ru
Первое сентября
март 2015
ИНФОРМАТИКА Подписка на сайте www.1september.ru или по каталогу «Почта России»: 79066 — бумажная версия, 12684 — CD-версия

НА ОБЛОЖКЕ

ЗЕМЛЯ ГУЛЛИВЕРА

В марте исполняется 20 лет одной из самых известных интернет-компаний Yahoo!. Предыдущее предложение не случайно заканчивается двумя знаками препинания, поскольку восклицательный знак входит в название. При учреждении Yahoo! его пришлось добавить — название “Yahoo” было уже занято торговой маркой соуса для кетчупа. По поводу же самого слова имеются разные точки зрения, но авторская — точка зрения основателей компании — заключается в том, что название посвящено народу “йеху”, земли которого посетил в своих путешествиях Гулливер.

Ken Wolter / Shutterstock.com

В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Уважаемые подписчики бумажной версии журнала!

Дополнительные материалы к номеру и электронная версия журнала находятся в вашем Личном кабинете на сайте www.1september.ru.

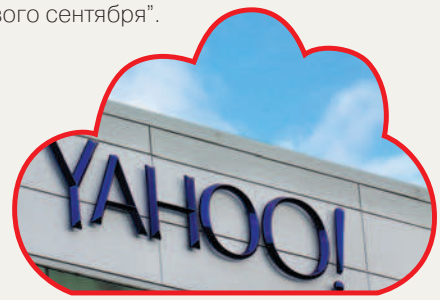
Для доступа к материалам воспользуйтесь, пожалуйста, кодом доступа, вложенным в № 1/2015.

Срок действия кода: с 1 января по 30 июня 2015 года.

Для активации кода:

- зайдите на сайт www.1september.ru;
- откройте Личный кабинет (создайте, если у вас его еще нет);
- введите код доступа и выберите свое издание.

Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.



ЭЛЕКТРОННЫЕ МАТЕРИАЛЫ

Презентации к статьям номера

ИНФОРМАТИКА

ПОДПИСНЫЕ ИНДЕКСЫ
по каталогу “Почта России”: 79066 — бумажная версия, 12684 — электронная версия

http://inf.1september.ru

Учебно-методический журнал для учителей информатики
Основан в 1995 г.
Выходит один раз в месяц

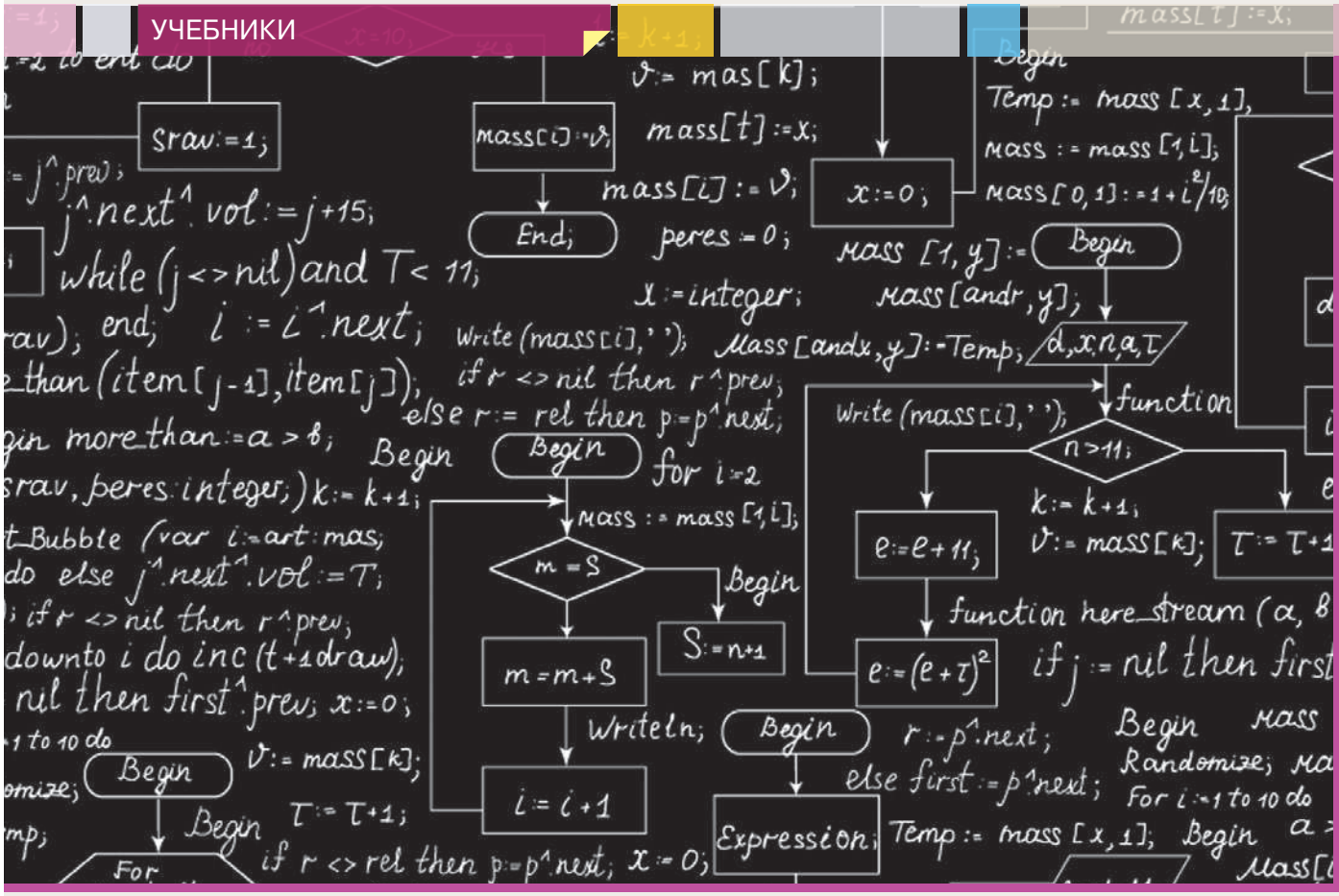
РЕДАКЦИЯ:
гл. редактор С.Л. Островский
редакторы
Е.В. Андреева,
Д.М. Златопольский (редактор вкладки “В мир информатики”)
Дизайн макета И.Е. Лукьянов
верстка Н.И. Пронская
корректор Е.Л. Володина
секретарь Н.П. Медведева
Фото: фотобанк Shutterstock
Журнал распространяется по подписке
Цена свободная
Тираж 16 000 экз.
Тел. редакции: (499) 249-48-96
E-mail: inf@1september.ru
http://inf.1september.ru

ИЗДАТЕЛЬСКИЙ ДОМ “ПЕРВОЕ СЕНТЯБРЯ”
Главный редактор:
Артем Соловейчик (генеральный директор)
Коммерческая деятельность:
Константин Шмарковский (финансовый директор)
Развитие, IT и координация проектов:
Сергей Островский (исполнительный директор)
Реклама, конференции и техническое обеспечение Издательского дома:
Павел Кузнецов
Производство:
Станислав Савельев
Административно-хозяйственное обеспечение:
Андрей Ушков
Педагогический университет:
Валерия Арсланьян (ректор)

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА “ПЕРВОЕ СЕНТЯБРЯ”
Английский язык – А.Громушкина
Библиотека в школе – О.Громова
Биология – Н.Иванова
География – и.о. А.Митрофанов
Дошкольное образование – Д.Тюттерин
Здоровье детей – Н.Сёмина
Информатика – С.Островский
Искусство – О.Волкова
История – А.Савельев
Классное руководство и воспитание школьников – М.Битянова
Литература – С.Волков
Математика – Л.Рослова
Начальная школа – М.Соловейчик
Немецкий язык – М.Бузоева
ОБЖ – А.Митрофанов
Русский язык – Л.Гончар
Спорт в школе – О.Леонтьева
Технология – А.Митрофанов
Управление школой – Е.Рачевский
Физика – Н.Козлова
Французский язык – Г.Чесновицкая
Химия – О.Блохина
Школа для родителей – Л.Печатникова
Школьный психолог – М.Чибисова

УЧРЕДИТЕЛЬ:
ООО “ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»”
Зарегистрировано ПИ № ФС77-58447 от 25.06.2014 в Роскомнадзоре
Подписано в печать:
по графику 15.01.2015,
фактически 15.01.2015
Заказ №
Отпечатано в ОАО “Первая Образцовая типография”
Филиал “Чеховский Печатный Двор”
ул. Полиграфистов, д. 1,
Московская область,
г. Чехов, 142300
Сайт: www.chpd.ru
E-mail: sales@chpk.ru
Факс: 8 (495) 988-63-76

АДРЕС ИЗДАТЕЛЯ:
ул. Киевская, д. 24,
Москва, 121165
Тел./факс: (499) 249-31-38
Отдел рекламы:
(499) 249-98-70
http://1september.ru
ИЗДАТЕЛЬСКАЯ ПОДПИСКА:
Телефон: (499) 249-47-58
E-mail: podpiska@1september.ru



Алгоритмизация и программирование

К.Ю. Поляков,
д. т. н., Санкт-Петербург,
http://kpolyakov.spb.ru,

Е.А. Еремин,
к. ф.-м. н., г. Пермь

Графические примитивы

Ключевые слова:
- графический примитив
- линия
- прямоугольник
- окружность
- ломаная
- контур
- перо
- заливка
- кисть

Что такое графические примитивы?

Программа строит изображение на экране с помощью графических примитивов — элементарных фигур. Каждая из таких фигур рисуется на экране с помощью одной команды.

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

Основные примитивы исполнителя Рисователь:
- пиксель
- линия
- прямоугольник
- окружность

Одну команду — **пиксель** — мы уже знаем, теперь изучим остальные.

Линия рисуется пером. “Перо” — это набор свойств линии: толщина и цвет. Для того чтобы нарисовать линию, нужно сначала выбрать ее характеристики с помощью команды **перо**:

```
перо( 1, синий )
```

Первый аргумент в скобках — это толщина линии, второй — ее цвет.

Окончание. Начало см. в № 2/2015.

Обратите внимание, что эта команда сама по себе ничего не рисует, но устанавливает режим рисования для следующих команд.

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 (10,20) в точку (15,20).

Вспомните, что ранее мы решили эту задачу с помощью команды **пиксель** и цикла. Решение с помощью команды **линия** значительно проще. Кроме того, она позволяет так же легко рисовать любые отрезки на холсте, в том числе и наклонные. Конечно, эти отрезки также состоят из пикселей, но расчет координат очередной точки берет на себя команда **линия**.

Теперь научимся рисовать прямоугольники. Рисователь умеет строить только прямоугольники, у которых стороны строго вертикальны и строго горизонтальны. Для того чтобы нарисовать прямоугольник, нужно знать координаты двух его противоположных углов — верхнего левого и нижнего правого. Так как две стороны вертикальны, а две горизонтальны, по этим данным можно определить координаты двух оставшихся углов.

Эта программа рисует прямоугольник, противоположные углы которого находятся в точках (20,10) и (40,30):

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

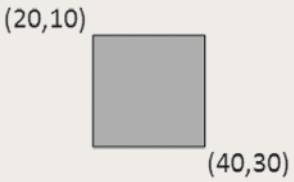


Рис. 64

В первой строчке мы определяем свойства контура, как для линий. Прямоугольник — это замкнутая фигура, поэтому для него можно задать еще и цвет заливки. Это делается с помощью команды **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. Этой команде передаются четыре аргумента, они разделены запятыми. Первая пара чисел — это координаты левого верхнего угла, вторая — координаты правого нижнего.

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

Эта программа рисует круг с центром в точке (50, 30) радиуса 20 пикселей:

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

Рис. 65

Сама окружность будет синего цвета, а внутренняя часть круга заливается красным цветом.

Ломаные

Как мы увидели, примитивов в СКИ Рисователя не так много. Например, нет специальной команды для рисования треугольника (как вы думаете, почему?). Но треугольник можно построить из отрезков, вызвав несколько раз команду **линия**:

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

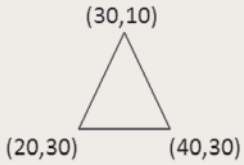


Рис. 66

Заметим, что данные в этих строчках повторяются — следующий отрезок начинается в той же точке, где заканчивается предыдущий. Это означает, что мы рисуем ломаную линию, “не отрывая пера от холста”. Чтобы не повторяться, в таких случаях удобно использовать другие команды, 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 исполнителя 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. В первой строке программы мы выводим исполнителя в вершину треугольника (20,30), а затем тремя отрезками рисуем три стороны треугольника, замыкая контур.

С помощью этих команд можно построить и незамкнутую ломаную линию. Для этого не нужно строить последний отрезок, соединяющий конец ломаной с ее началом.

Заливка

Как же залить треугольник? Вспомните, что заливка прямоугольников и кругов происходила при

вызове команд рисования, одновременно с рисованием контура. Здесь контур уже готов, и нужно залить его 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 начинается в точке (30,20), область заполняется красным цветом. Где же остановится заливка? Это зависит от цвета пикселя (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:

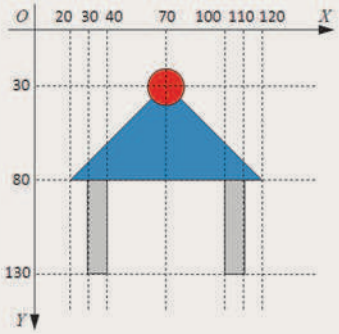


Рис. 67

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

Рисунок состоит из двух прямоугольников, треугольника и круга. Чтобы нам легче было составить программу, мы нанесли на рисунок линии сетки, которые показывают координаты всех угловых точек графических примитивов: прямоугольников, круга и отрезков, из которых строится треугольник-крыша.

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 объяснения строки пронумерованы):

```
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
```

Сначала создаем холст размером 200 × 200 пикселей (строка 1). Затем устанавливаем черный цвет контура (строка 2) и серый цвет заливки (строка 3). В строках 4 и 5 рисуем столбики-прямоугольники с этими свойствами.

Дальше рисуем крышу из трех отрезков (строки 7–9). После того как контур готов, устанавливаем синий цвет кисти (строка 10) и закрашиваем треугольник (строка 11).

Круг с красной заливкой рисуем в последнюю очередь (строки 12–13).

Контрольные вопросы

1. Зачем в СКИ графических исполнителей добавляют команды для рисования примитивов?
2. Как определить размеры прямоугольника по координатам углов?
3. Алгоритмы какого типа мы использовали в этом параграфе?
4. Как нарисовать замкнутую ломаную линию и залить ее внутреннюю область? С какими проблемами вы можете столкнуться?
5. Как вы думаете, почему обычно в СКИ исполнителей нет команды **треугольник**?
6. Что произойдет, если исполнитель будет рисовать за пределами холста?

Задачи

1. Определите размеры прямоугольников, которые нарисованы с помощью команд
   а) **прямоугольник(10, 10, 50, 20)**
   б) **прямоугольник(110, 120, 50, 20)**
   в) **прямоугольник(110, 20, 50, 120)**
   г) **прямоугольник(10, 130, 150, 20)**
2. Посмотрите, как нарисованы кнопки в окне какой-нибудь программы. Нарисуйте с помощью Рисователя изображения кнопки в нажатом и отпущенном положениях.
3. Нарисуйте с помощью Рисователя стандартное окно программы в вашей операционной системе.
4. Придумайте и нарисуйте с помощью Рисователя свой рисунок, в котором есть прямоугольники, окружности и ломаные.
5. Напишите программы для Рисователя, которые строят такие рисунки:

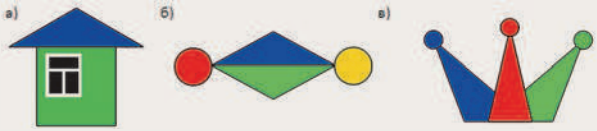


Рис. 68

6. Напишите программы для Рисователя, которые строят изображения трехмерных объектов:

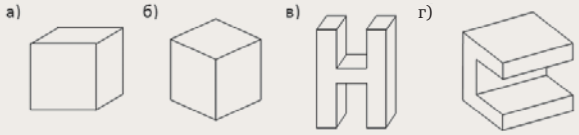


Рис. 69

Темы для сообщений:
а) “Алгоритмы выполнения заливки”
б) “Команда **эллипс**”
в) “Цвет в модели RGB”

Применение процедур

Ключевые слова:
• процедура
• параметры

Когда помогут процедуры?

Как вы знаете, процедуры (вспомогательные алгоритмы) служат для того, чтобы не писать несколько раз одинаковые серии команд. Например, нам нужно нарисовать три прямоугольных треугольника одинаковых размеров:

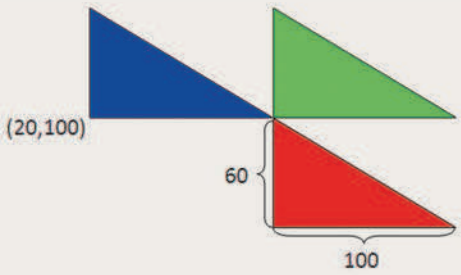


Рис. 70

Хотя треугольники и похожи, они не совсем одинаковы: у них разные
• цвета заливки;
• расположение (координаты углов на холсте).

Эти данные нужно как-то передать в процедуру из вызывающего алгоритма. Таким образом, нужна *процедура с параметрами*.

Сразу понятно, что один из параметров — это цвет заливки. Подумаем, какие данные нужны для того, чтобы определить место каждого из треугольников на холсте. Конечно, можно задать координаты всех трех вершин, это шесть чисел (три пары координат). При этом мы сможем рисовать любые (!) треугольники. Но в нашей задаче ситуация проще: размеры сторон известны, все треугольники прямоугольные, и самое важное — стороны прямого угла параллельны сторонам холста. Поэтому координаты любого угла позволяют вычислить координаты всех остальных углов и построить треугольник.

Строим процедуру

Предположим, что мы знаем координаты прямого угла. Как найти координаты остальных углов? Построим один треугольник:

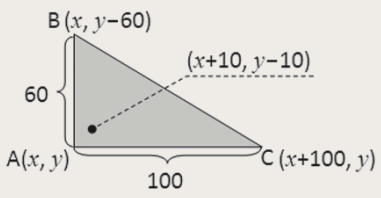


Рис. 71

Пусть (x, y) — координаты прямого угла А. Так как отрезок АВ идет строго вертикально, угол В расположен на таком же расстоянии от левой границы, что и угол А, поэтому их x-координаты совпадают. В то же время y-координата угла В будет меньше y-координаты А на 60 (на высоту треугольника). Поэтому угол В имеет координаты $(x, y – 60)$.

Так как отрезок АС идет строго горизонтально, угол С расположен на таком же расстоянии от верхней границы, что и угол А, поэтому их y-координаты совпадают. В то же время x-координата угла С будет больше y-координаты А на 100 (на длину основания треугольника). Поэтому угол С имеет координаты $(x + 100, y)$.

Для того чтобы залить треугольник, нам понадобится любая точка внутри него, например, точка $(x + 10, y – 10)$.

Теперь можно написать текст процедуры:

```
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
```

Процедура принимает три параметра:
• два целых значения **x** и **y** — координаты прямого угла;
• цвет заливки, обозначенный именем **ц**; эта величина относится к особому типу **цвет**.

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. Обратите внимание, что все координаты зависят от значений **x** и **y**, то есть процедура позволяет нам рисовать треугольники в любом месте холста.

Используем процедуру

Теперь составим основную программу. Она будет содержать три вызова процедуры “треугольник” с разными значениями параметров. Нам нужно знать координаты прямых углов всех треугольников. Для первого (синего) треугольника эти координаты нам известны — (20, 100). Для зеленого треугольника x-координата будет на 100 больше, то есть его прямой угол имеет координаты (120, 100). Красный треугольник находится ниже зеленого на 60 пикселей, поэтому координаты его прямого угла — (120, 160). Теперь можно записать основную программу:

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

Не забудьте, что после этой программы нужно поместить текст процедуры, иначе Рисователь не сможет выполнить неизвестную команду “треугольник”.

При первом вызове процедура выполняется для значений **x** = 20, **y** = 100 и **ц** = синий. Поэтому фактически будут выполнены команды

```
в точку( 20, 100 )
линия в точку( 20, 40 )
линия в точку( 120, 100 )
линия в точку( 20, 100 )
кисть( синий )
залить( 30, 90 )
```

После этого процедура вызывается еще два раза, с другими значениями параметров. Поэтому треугольники будут нарисованы в других местах холста.

Контрольные вопросы

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** длины сторон треугольника? Какие достоинства и недостатки имеет это решение?
4. Процедура **треугольник** записана ниже основной программы, но треугольники не появляются на хосте. В чем может быть причина? Как вы будете искать ошибку?

Задачи

1. Перепишите процедуру **треугольник**, взяв за точку отсчета правый угол основания треугольника (считайте, что он имеет координаты (*x*, *y*), которые передаются как параметры). Измените основную программу так, чтобы получился точно такой же рисунок, как и раньше.
2. Напишите программы для Рисователя, которые строят такие рисунки с помощью одной процедуры:

Рис. 72

3. Напишите программы для Рисователя, которые строят изображения трехмерных объектов с помощью одной процедуры:

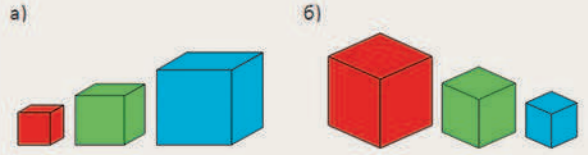


Рис. 73

Применение циклов

Ключевые слова:
- цикл
- узор
- процедура
- свойства по умолчанию

Часто в рисунках встречаются одинаковые элементы, которые удобно рисовать с помощью циклов. В этом параграфе вы узнаете, как грамотно составлять такие циклы.

Узоры

Узор — это рисунок, основанный на повторении одинаковых элементов. Эти элементы могут быть различной сложности, начиная с примитивов — кругов, прямоугольников и др.

Начнем с простой задачи: построим ряд из пяти кругов радиуса 5 пикселей:

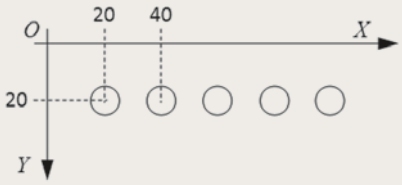


Рис. 74

Конечно, можно рисовать их отдельно, используя пять вызовов команды **окружность**:

```
окружность(  20, 20, 5)
окружность(  40, 20, 5)
окружность(  60, 20, 5)
окружность(  80, 20, 5)
окружность( 100, 20, 5)
```

Можно заметить, что в этих вызовах изменяется только *x*-координата центра. Ее можно сделать переменной и назвать **x**. Эта переменная будет меняться от 20 до 100 с шагом 20, поэтому можно использовать такой цикл с 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 переменной происходит с шагом 20: 20, 40, 60, … По умолчанию (то есть если мы не указали иначе) шаг равен единице.

Теперь построим три одинаковых ряда кругов, расположенных на расстоянии 20 пикселей по высоте друг от друга:

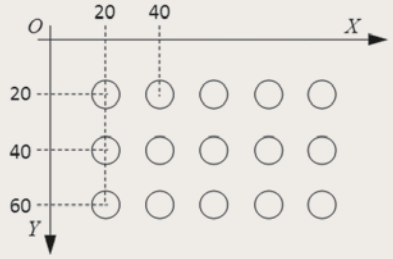


Рис. 75

Один ряд отличается от другого только *y*-координатой, поэтому можно оформить вспомогательный алгоритм (процедуру):

```
алг Ряд ( цел y )
нач
  цел x
  нц для x от 20 до 100 шаг 20
    окружность(x, y, 5)
  кц
кон
```

а затем в основной программе вызвать ее три раза

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

Снова замечаем, что *y*-координата, которая передается процедуре **Ряд**, изменяется с постоянным шагом 20, поэтому можно обозначить ее как переменную **y** и использовать цикл:

```
цел y
нц для y от 20 до 60 шаг 20
  Ряд( y )
кц
```

Вызов процедуры можно заменить на вложенный цикл, в котором изменяется переменная **x**:

```
цел x, y
нц для y от 20 до 60 шаг 20
  нц для x от 20 до 100 шаг 20
    окружность(x, y, 5)
  кц
кц
```

Использование процедур

Вместо команды **окружность** можно вызывать и свою процедуру. Например, построим на экране такой узор из ромбов:

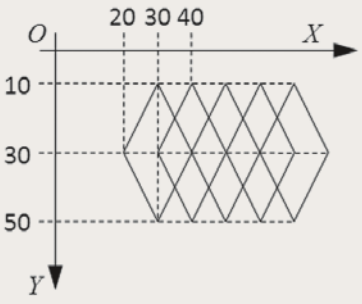


Рис. 76

Чтобы составить процедуру, нарисуем один ромб, определим по рисунку его размеры и обозначим через (*x*, *y*) координаты одного из углов. Это позволит рассчитать координаты всех остальных углов так же, как мы делали ранее.

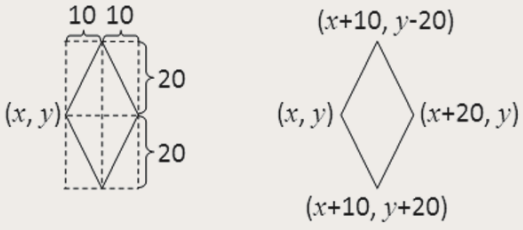


Рис. 77

Составляем процедуру, которая рисует контур ромба:

```
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. По схеме *рис.* 76 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:

```
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 состоит из параллельных линий, которые удобно рисовать в цикле. Выполним вертикальную штриховку прямоугольника:

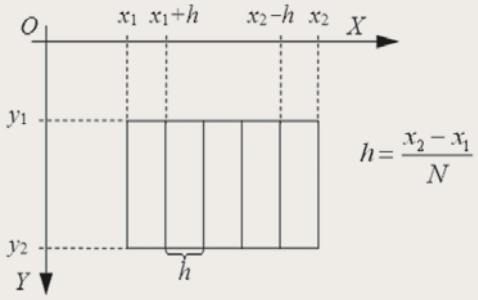


Рис. 78

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 строчках 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, так что шаг штриховки (расстояние между соседними линиями) можно вычислить по формуле

$$h = \frac{x_2 - x_1}{N}.$$

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**:

```
цел h
h := div( x2 - x1, N )
```

Например, если $x_1 = 100$, $x_2 = 200$ и $N = 5$, мы получим $h = 20$.

Как видно из *рис.* 78, первая линия штриховки идет из точки $(x_1 + h, y_1)$ в точку $(x_1 + h, y_2)$:

```
линия( x1 + h, y1, x1 + h, y2 )
```

У второй линии x-координата обоих концов отрезка увеличивается на h, у третьей — еще на h и т.д. Для обоих концов последней линии x-координата равна $x_2 - h$, т.к. $(x_2 - x_1)$ в нашем примере делится на N. Можно обозначить эту изменяющуюся величину как переменную **x** и записать такой цикл штриховки:

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

Приведем полную программу, которая строит прямоугольник и выполняет штриховку:

```
использовать Рисователь
алг Штриховка
нач
  цел N = 5
  цел x1 = 100, x2 = 300
  цел y1 = 100, y2 = 200
  цел h, x
  прямоугольник( x1, y1, x2, y2 )
  h := div( x2 - x1, N )
  нц для x от x1 + h до x2 - h шаг h
    линия( x, y1, x, y2 )
  кц
кон
```

Контрольные вопросы

1. Подумайте, что нужно изменить в первой программе, чтобы круги выстроились по диагонали:

Нужен ли для этого вложенный цикл?

2. Сравните два варианта решения второй задачи:

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 последней программе, чтобы сделать горизонтальную штриховку?

Задачи

1. Постройте изображение шахматной доски размером 8 × 8 клеток.

2. Постройте узоры:

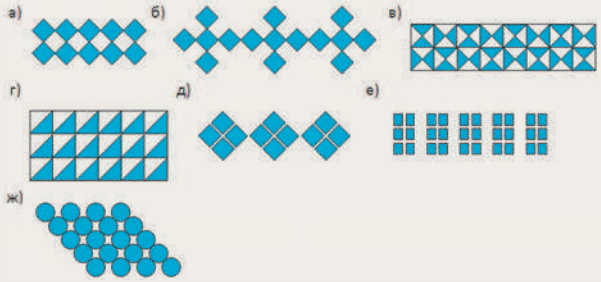


Рис. 79

3. Постройте следующие рисунки (число линий храните в переменной **N**):

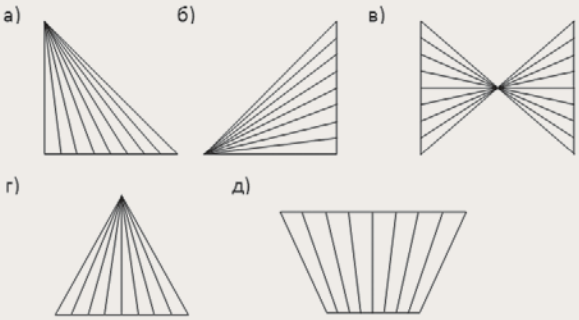


Рис. 80

4. В программе штриховки мы использовали целочисленное деление, чтобы шаг (интервал между линиями) получился целый. В чем недостаток такого подхода? Попробуйте изменить программу так, чтобы он был явно виден. Предложите вариант решения проблемы.

5. Выполните штриховку (число линий храните в переменной **N**):

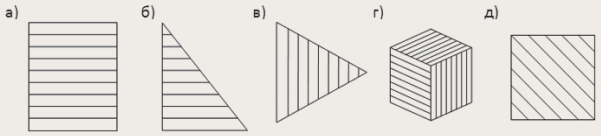


Рис. 81

Тема для сообщения:
“Муаровый эффект”

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”. Анимация на компьютере состоит в том, что простая смена рисунков создает иллюзию движения. Каждый из таких рисунков называют кадром. Допустим, мы нарисовали четыре рисунка одинакового размера:

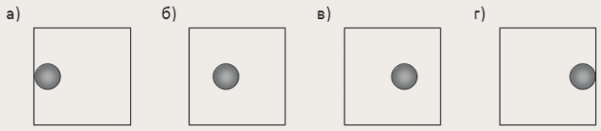


Рис. 82

и меняем их на экране, скажем, через каждую секунду: сначала выводим рисунок а, затем, через 1 с, — рисунок б, еще через 1 с — рисунок в и т.д. Что 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 перестает замечать смену изображений и видит непрерывное движение.

Однако не всегда можно нарисовать заранее все кадры анимации. Например, в компьютерных играх игрок управляет персонажем с помощью клавиатуры, мыши или джойстика, и поэтому нельзя предсказать, какое изображение возникнет на экране в следующий момент.

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

- “стереть” шарик, то есть восстановить фон в том месте, где сейчас нарисован шарик;
- изменить положение шарика (его координаты на холсте);
- запомнить участок фона, который будет испорчен при выводе шарика в новом месте;
- 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 и закрасим его синим (или каким-нибудь другим) цветом:

```
новый лист( 200, 200, синий )
```

Размер этого холста — 200 × 200 пикселей.

Шарик будем изображать в виде круга желтого цвета, его радиус — 10 пикселей. Контур рисовать не будем, поэтому в начале программы 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. Для того чтобы нарисовать шарик, процедуре нужно передать цвет “желтый”, а для стирания — “синий”:

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

Теперь подумаем, как использовать эту процедуру. Для хранения текущих координат шарика (то есть координат в данный момент) будем использовать переменные **x** и **y** основной программы. Шарик будет двигаться слева направо по середине холста на уровне $y = 100$ (*рис.* 83).

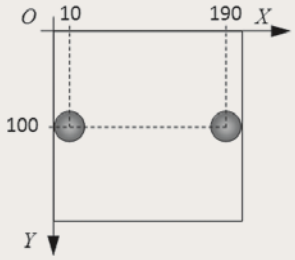


Рис. 83

В начальный момент он находится у левой границы, то есть x-координата его центра равна радиусу (10 пикселей). Шарик выходит за границы экрана, когда расстояние от центра до правой границы холста станет меньше, чем радиус, то есть когда x-координата его центра станет больше, чем 200 – 10 = 190. В этом случае цикл нужно закончить. Следовательно, цикл выполняется “пока $x <= 190$”. Таким образом, основная программа может выглядеть так:

```
использовать Рисователь
алг Анимация
нач
  новый лист ( 200, 200, синий )
  цел x = 100, y = 100
  перо( 1, прозрачный )
  нц пока x <= 190
    | движение шарика
  кц
кон
```

В теле цикла сейчас только комментарий. Там нужно записать все повторяющиеся операции: вывод шарика на экран, скрытие шарика и его перемещение. Для того чтобы нарисовать шарик, нужно просто вызвать процедуру **Шарик**:

```
Шарик ( x, y, желтый )
```

Шарик двигается на одноцветном фоне, поэтому для стирания достаточно нарисовать его в том же

месте, задав синий цвет кисти (совпадающий с цветом фона):

```
Шарик ( x, y, синий )
```

Как вы думаете, что произойдет, 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 ее длину 20 миллисекунд (1 миллисекунда = 1/1000 секунды). Добавить паузу в программу можно с помощью команды “ждать”:

```
ждать ( 20 )
```

В скобках записано время в миллисекундах.

Остается 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 два 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, желтый )
  ждать( 20 )
  Шарик( x, y, синий )
  x := x + 2
кц
```

Обратите внимание, что изменять координаты шарика нужно тогда, когда он скрыт. Иначе мы будем стирать его не в том месте, где до этого нарисовали.

Теперь остается “собрать” всю программу. Вы уже можете сделать это самостоятельно. Не забудьте, что после основной программы нужно поместить процедуру **Шарик**.

Контрольные вопросы

1. Почему в работах профессиональных аниматоров кадры сменяются не реже, чем 16 раз в секунду?
2. Почему обычно перерисовывают не все 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 коснется 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. Найдите ошибки в основном цикле анимации:

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

Что будет происходить на экране в этом случае? Проверьте свой ответ экспериментально.

Задачи

1. Измените программу, приведенную в этом параграфе, так, чтобы шарик двигался справа налево.
2. По холсту перемещаются два шарика разного цвета и одинакового радиуса, один — слева направо, второй — справа налево. Скорости движения шариков одинаковые. Разработайте программу для выполнения этой анимации.
3. По холсту перемещаются два шарика разного цвета и разного радиуса, один — слева направо, второй — сверху вниз. Скорости движения шариков разные. Разработайте программу для выполнения этой анимации.
4. *По холсту перемещается шарик, который отскакивает от границ холста. Сначала он движется слева направо, потом, оттолкнувшись от правой границы холста, начинает движение влево и т.д. Разработайте программу для выполнения этой анимации. Используйте бесконечный цикл вида

```
нц пока да
  ...
кц
```

5. *По холсту размером 400 × 200 пикселей перемещается шарик, он движется по диагонали и отскакивает от краев холста. Разработайте программу для выполнения этой анимации.

Тема для сообщения:
“Компьютерная анимация в кино и рекламе”

Управление с помощью клавиатуры

Ключевые слова:
- интерактивность
- скан-код клавиши
- 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 мы узнаем, 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 среде КуМир[9] для этого используется логическая команда **клавиша нажата**. Результат выполнения этой команды — это логическое значение[10]: ответ “да” или “нет”. Например, можно использовать такое ветвление:

```
если клавиша нажата 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 (так называемый скан-код). Для того чтобы узнать скан-код нажатой клавиши, используется команда **код клавиши**:

```
цел к
к := код клавиши
если к = КЛ_ВВЕРХ то
  | передвинуть объект вверх
все
```

Чтобы не запоминать числовые коды всех клавиш, для них введены символьные обозначения. Например, коды клавиш-стрелок обозначаются как **КЛ_ВВЕРХ**, **КЛ_ВНИЗ**, **КЛ_ВПРАВО** и **КЛ_ВЛЕВО** (все буквы заглавные).

Управление с ожиданием

Сначала научимся управлять каким-то объектом, например изображением шарика (см. предыдущий раздел), в режиме ожидания. Это значит, что программа ждет нажатия на клавишу, определяет ее код и после этого перемещает шарик на экране в нужную сторону.

Основной цикл программы на псевдокоде (смеси русского и алгоритмического языков) можно написать так:

```
нц пока да
  Шарик ( x, y, "желтый" )
  к := код клавиши
  Шарик ( x, y, "синий" )
  | переместить шарик
кц
```

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

Для рисования и стирания шарика используется процедура **Шарик**, с которой мы работали в предыдущем параграфе. В строчке

```
к := код клавиши
```

мы сказали программе, что нужно

1) ждать, пока не будет нажата какая-нибудь клавиша;

2) когда клавиша нажата, сохранить ее код в переменной **к**.

Обратите внимание, что мы стираем шарик после того, как клавиша нажата (во время ожидания шарик виден на экране).

Остается раскрыть комментарий “переместить шарик”. Как вы уже знаете, нам нужно изменить координаты — значения переменных **x** и **y**. Причем эти изменения будут зависеть от того, какую клавишу нажал пользователь. Например, можно написать четыре условных оператора:

```
если к = КЛ_ВЛЕВО  то x := x – 5 все
если к = КЛ_ВПРАВО то x := x + 5 все
если к = КЛ_ВВЕРХ  то y := y – 5 все
если к = КЛ_ВНИЗ   то y := y + 5 все
```

Получается такая основная программа:

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

После нее нужно поместить процедуру **Шарик** из предыдущего параграфа.

Управление по требованию

В только что написанной программе исполнитель ожидал, когда мы нажмем на клавишу, и в это время никаких действий не выполнял. Часто при программировании игр нужен другой режим: события развиваются, но игрок может вмешаться тогда, когда считает нужным. Это управление по требованию. Для того чтобы использовать такое управление, надо определить момент, когда пользователь нажимает на клавишу, и реагировать на это нажатие.

Мы напишем программу, в которой шарик постоянно движется, изменяя свое направление при нажатии клавиш-стрелок. Основной цикл программы выглядит так:

```
нц пока да
  если клавиша нажата
    к := код клавиши
    | изменить направление движения
  все
  Шарик( x, y, желтый )
  ждать( 20 )
  Шарик( x, y, синий )
  | переместить шарик
кц
```

[9] Здесь используются названия команд для работы с клавиатурой, которые приняты в тестовой версии КуМир 2x.

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

Пока мы еще не решили, как изменять направление движения и как перемещать шарик с учетом направления (эти строчки заменены на комментарии).

Чтобы задать направление движения шарика, нужно определить, как изменяется каждая из его координат на каждом шаге. Будем хранить изменение *x*-координаты в переменной **dx**, а изменение *y*-координаты — в переменной **dy**. Тогда перемещение шарика записывается в две строчки:

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

Куда именно передвинется шарик, зависит от значений **dx** и **dy**. Если за один шаг шарик смещается на пять пикселей, получаем такие значения для разных направлений:

Направление	dx	dy
влево	−5	0
вправо	5	0
вверх	0	−5
вниз	0	5
стоп	0	0

Тогда для изменения направления движения при нажатии клавиши достаточно изменить значения **dx** и **dy**:

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

В последней строке мы останавливаем шарик при нажатии на пробел. Эти команды нужно поставить в основной цикл вместо комментария “изменить направление движения”. Полную программу вы уже можете собрать самостоятельно.

Контрольные вопросы

1. Какие две задачи нужно уметь решать для управления объектом с клавиатуры? Какие задачи невозможно решить, если не будет команды **клавиша нажата**?
2. Что такое скан-код? Верно ли, что при вводе символа “Я” и символа “Z” мы получим один и тот же скан-код клавиши?
3. Что произойдет, если при работе первой программы нажата какая-нибудь другая клавиша, кроме стрелок? Как можно устранить этот недостаток?
4. Как можно при нажатии клавиши изменять радиус шарика?
5. Как можно при нажатии клавиши изменять скорость движения шарика?
6. Зачем во второй программе использована команда **ждать**?
7. Можно ли во второй программе переставить оператор **если** … **все**, который обрабатывает нажатие клавиши, в другое место в основном цикле? Ответ обоснуйте.

Задачи

1. Дополните первую программу так, чтобы шарик не мог выйти за пределы холста.
2. Добавьте во вторую программу возможность изменения с клавиатуры
   а) скорости движения шарика;
   б) радиуса шарика.
3. Добавьте во вторую программу отталкивание шарика от стенок холста.

Тема для сообщения:

“Оператор **выбор** в алгоритмическом языке”

Выводы из прочитанного:

- Алгоритм — это точное описание порядка действий, которые должен выполнить исполнитель для решения задачи. Основные свойства алгоритма — дискретность, понятность и определенность.
- Исполнитель — это человек, животное или машина, которые могут понимать и выполнять некоторые команды. Система команд исполнителя (СКИ) — это набор команд, который понимает и умеет выполнять исполнитель.
- Формальный исполнитель — это исполнитель, который одну и ту же команду всегда выполняет одинаково.
- Алгоритм можно записать в словесной форме, в виде блок-схемы или в виде программы на языке программирования.
- Программа — это алгоритм, записанный на языке конкретного исполнителя. Часто используют псевдокод — смесь естественного языка и языка программирования.
- Ключевые слова — это специальные слова языка программирования, имеющие единственное заранее определенное значение.
- Комментарий — это пояснение к программе. Комментарии не обрабатываются исполнителем.
- Различают три типа ошибок в программах: синтаксические ошибки, отказы и логические ошибки. Для того чтобы найти логические ошибки, используют ручную прокрутку, выполняя алгоритм без исполнителя.
- Для хранения данных используют переменные — величины, значения которых можно изменять во время работы алгоритма.
- Алгоритм решения любой задачи можно составить с помощью следования (линейных алгоритмов), ветвлений (разветвляющихся алгоритмов) и циклов (циклических алгоритмов).
- В линейном алгоритме команды выполняются в том же порядке, в котором они записаны.
- Вспомогательный алгоритм (процедура) — это новая команда, которой мы дополняем СКИ исполнителя. Для того чтобы процедура выполнилась, нужно вызвать ее из основной программы. После завершения работы процедуры управление передается обратно, к следующей команде вызывающей программы.
- Параметры — это данные, которые передаются в процедуру. Каждый параметр имеет имя и тип.
- Существует два метода разработки программ: “снизу вверх” и “сверху вниз”.
- При использовании метода “снизу вверх” сначала составляются вспомогательные алгоритмы, а затем из них строится основная программа.
- В методе “сверху вниз” (методе последовательного уточнения) задача разбивается на подзадачи, каждая из подзадач оформляется в виде вспомогательного алгоритма. Сначала составляется основная программа, а затем все вспомогательные алгоритмы.
- Циклический алгоритм — это алгоритм, в котором некоторая последовательность действий (тело цикла) выполняется несколько раз. Существует два вида циклов: циклы с известным числом шагов и циклы с условием.
- Цикл с условием — это цикл, который выполняется до тех пор, пока некоторое условие не станет ложным. Количество шагов такого цикла определяется исходными данными. Если в цикле с условием сделана ошибка, программа может зациклиться.
- Вложенный цикл — это цикл, расположенный внутри другого цикла.
- Логическая команда — это вопрос, на который исполнитель отвечает “да” или “нет”.
- Разветвляющийся алгоритм — это алгоритм, в котором последовательность действий изменяется в зависимости от выполнения некоторых условий.
- Диалоговая программа — это программа, в которой исходные данные вводятся человеком с клавиатуры, а результаты работы выводятся на экран.
- Программы, работающие в графическом режиме, могут управлять отдельно каждым пикселем области рисования — холста.
- Графический примитив — это геометрическая фигура, которая добавляется на рисунок с помощью одной команды. При рисовании примитивов свойства линии определяются объектом `перо`, а свойства заливки — объектом `кисть`.
- Анимация — это создание иллюзии движения на экране. Компьютерная анимация — это быстрая смена рисунков (кадров). Для того чтобы переместить изображение объекта на фоне, нужно скрыть его, изменить координаты и снова вывести на холст. Если фон — это картинка, перед рисованием объекта нужно запомнить часть холста, которая будет изменена.
- Для управления с клавиатуры используют две функции. Одна из них дает ответ на вопрос “нажата ли какая-нибудь клавиша”, а вторая определяет код нажатой клавиши. Каждая клавиша на клавиатуре имеет свой код, который называют скан-кодом.