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

ИНФОРМАТИКА

Augusta Ada King Byron, Countess of Lovelace: 200 лет со дня рождения

```
with Ada.Text_IO;
procedure Hello is
  use Ada.Text_IO;
begin
  Put_Line("Hello, world!");
end Hello;
```

электронная ВЕРСИЯ ЖУРНАЛА в Личном кабинете на сайте www.1september.ru

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

НА ОБЛОЖКЕ

В декабре исполняется 200 лет со дня рождения Авгу́сты А́ды Кинг (урожденной Ба́йрон), графини Ла́влейс (Augusta Ada King Byron, Countess of Lovelace), более известной как Ада Лавлейс. Она известна прежде всего созданием описания вычислительной машины, проект которой был разработан Чарльзом Бэббиджем. Составила первую в мире программу (для этой машины). Ввела в употребление термины “цикл” и “рабочая ячейка”, считается первым программистом в истории. В честь Ады Лавлейс был назван язык программирования Ада.

В НОМЕРЕ

В ЛИЧНОМ КАБИНЕТЕ

Облачные технологии от Издательского дома “Первое сентября”

Все подписчики журнала имеют возможность получать электронную версию, которая является полной копией бумажной. Для получения электронной версии:

1) Откройте Личный кабинет на портале “Первое сентября” (www.1september.ru).

2) В разделе “Газеты и журналы / Получение” выберите свой журнал и кликните на кнопку “Я — подписчик бумажной версии”.

3) Появится форма, посредством которой вы сможете отправить нам копию подписной квитанции.

После этого в течение одного рабочего дня будет активирована электронная подписка на весь период действия бумажной. Справки: podpiska@1september.ru или через службу поддержки на портале “Первого сентября”.

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 18 000 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)

Школа цифрового века: facebook.com/School.of.Digital.Age

ЖУРНАЛЫ ИЗДАТЕЛЬСКОГО ДОМА “ПЕРВОЕ СЕНТЯБРЯ”
Английский язык – Е.Богданова
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 16.09.2015,
фактически 16.09.2015
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

volkova natalia / Shutterstock.com

Под грифом “малоизвестный”

В 2015 году исполняется 40 лет с момента начала разработки языка программирования Ada, названного в честь Ады Лавлейс. Даже многие профессионалы в области программирования бывают удивлены тому, что язык этот не просто жив: регулярно выходят новые версии языка (последний новый стандарт датирован 2012 годом) и средства разработки. Более того, имеет место поразительный факт: специалисты по языку Ада требуются, и они в весьма серьезном дефиците. Это вызвано тем, что курсы по языку мало где читаются, а программных систем на нем работает много. Правда… весьма специфических программных систем…

В свое время Чарльз Хоар, который не принадлежал к числу поклонников языка, заметил, что опасается “армады ракет, летящих не туда из-за не обнаруженной вовремя ошибки в компиляторе Ады”. Хоар имел в виду именно основное назначение языка — то, для чего он изначально был создан и по большей части применяется: программирование встроенных систем преимущественно военного назначения, работающих в реальном времени. Именно Пентагон в 1975 году стал инициатором и заказчиком разработки Ады.

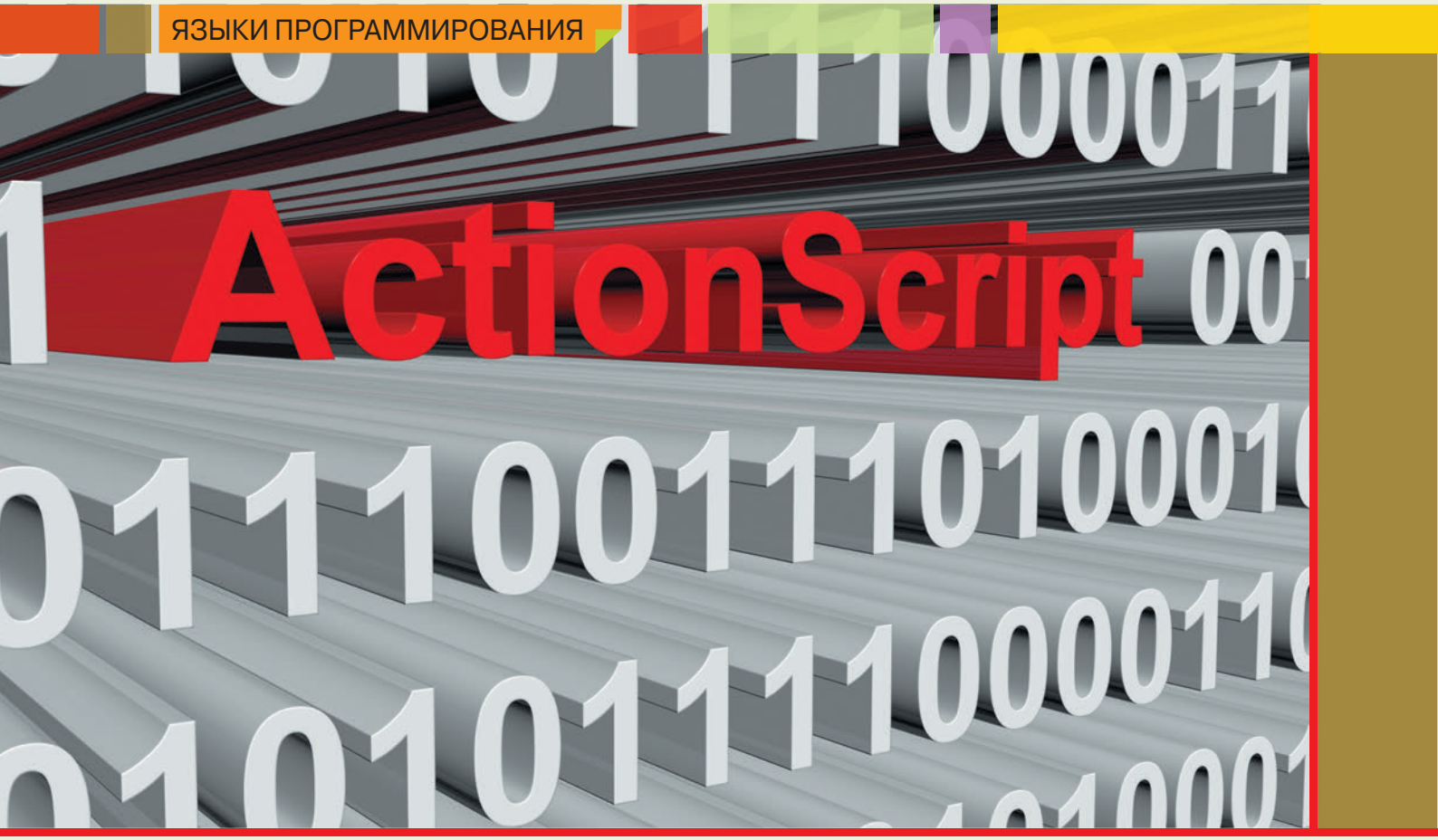
В СССР имелось немало специалистов по Аде — надо же было изучать функционирование систем вероятного противника. Компиляторы Ады были разработаны для большинства компьютеров, которые использовались на территории Союза. В открытых источниках называется и наиболее известный уже российский проект промышленного использования Ады — для ПО самолета-амфибии Бе-200.

На что похож язык Ада? Когда он только проектировался, был организован конкурс проектов языка. По условиям конкурса разработчики должны были базироваться на одном из трех языков: Паскаль, Алгол-68 или PL/1. На конкурс было представлено 15 проектов. На второй этап конкурса прошли четыре проекта, все основанные на Паскале. Как и Паскаль, Ада довольно поздно получил средства объектно ориентированного программирования — они появились только в стандарте 1995 года. Некоторые разработчики считают, что, как и в Паскале, они получились не вполне “родными” для языка и дополнительно усложнили его. А сложности Аде и так хватало. Одним из наиболее последовательных критиков Ады с этой точки зрения был Дейкстра:

“Если Ada собирается выдать стандарт, желательно, чтобы он был недвусмысленно документирован. По меньшей мере две группы попытались сделать это; в результате обе выдали около 600 страниц формального текста. Это гораздо больше, чем необходимо, чтобы удостовериться в невозможности хотя бы твердо установить, что оба документа определяют один и тот же язык. Ошибка очевидной неуправляемости этих двух документов кроется не в двух группах, составивших их, не в принятом ими формализме, а лишь в самом языке: сами не обеспечив формального определения, могут ли его разработчики скрыть, что они предлагают неуправляемого монстра. То, что Ada уменьшит проблемы программирования и увеличит надежность наших разработок до приемлемых границ, — это лишь одна из тех сказок, в которые могут поверить только люди с военным образованием”.

Тем не менее Ада жив. Язык используется и развивается. Специалисты по нему требуются. Может, и кого-то из наших учеников заинтересует такая перспективная с точки зрения работы ниша?

С.Л. Островский,
гл. редактор, *so@1september.ru*



Практикум программирования на ActionScript

Д.Ю. Усенков,
г. Москва

Предлагаемый цикл практических занятий предназначен для учащихся различных возрастов — от 8-х до 11-х классов, уже освоивших основы работы (рисование при помощи векторного графического редактора и создание анимаций традиционными средствами) в среде программы — редактора флеш-анимаций Macromedia/Adobe Flash[1] и желающих продолжить обучение, чтобы освоить основы программирования на языке ActionScript (скриптовом языке, “встроенном” в редактор флеш-анимаций).

Изучение приемов программирования на ActionScript производится на наглядных практических примерах, интересных для учащихся; соответствующий теоретический материал (команды, события, объекты и их свойства, методы, алгоритмические конструкции и т.д.) вводится по мере появления необходимости в нем. Таким образом, изложение ведется “от практики к теории”, что обеспечивает более высокий уровень мотивации учащихся к освоению материала.

Представленные разработки уроков (кроме самого первого занятия,

[1] Использована более старая версия Macromedia Flash Professional 8, которая с большей вероятностью может иметься в школе, чем новые версии, например, CS. Напомним также читателям, что технология создания флеш-анимаций была разработана компанией Macromedia, а позже приобретена компанией Adobe. Поэтому авторы считают более корректным для этой технологии название “Macromedia Flash”, тогда как инструментальная программная среда для создания анимаций может иметь название как “Macromedia Flash”, так и “Adobe Flash”, в зависимости от версии. — **Прим. авт.**

которое носит в основном ознакомительный характер) вначале содержат краткую теоретическую информацию, требуемую на данном занятии. Далее приводится одно или несколько практических заданий с пошаговым разбором необходимых для создания флеш-анимации действий; предполагается, что эти практические задания должны выполнить все учащиеся. После этого учащимся предлагается самостоятельно выполнить индивидуальные мини-проекты с использованием приемов программирования на ActionScript, изученных как на данном занятии, так и ранее (темы и содержание проектов — на усмотрение учителя). Созданные проекты могут затем демонстрироваться классу, в том числе с элементами соревновательности (например, с голосованием учащихся и выбором лучшего проекта).

Организация занятий и их почасовое планирование могут быть различными в зависимости от конкретной ситуации: это могут быть внешкольные занятия в рамках кружка или в доме детского творчества, внеклассный элективный курс по выбору учащихся либо элемент профильного курса информатики (в последнем случае самостоятельная проектная работа выносится на домашнее выполнение, а сам курс может иметь различную продолжительность в зависимости от имеющегося времени на его изучение, т.е. может ограничиваться только некоторым количеством первых занятий представленного цикла, а также возможно выполнение не всех предлагаемых практических заданий). Однако желательно выделить на курс не менее 36 часов, чтобы учащиеся могли выполнить все “обязательные” практические задания и выполнять индивидуальные проекты, консультируясь с учителем.

Представленный цикл практических занятий был успешно апробирован во Дворце детского творчества “Преображенское” (г. Москва) для учебной группы с возрастом обучаемых, соответствующим 10–11-му классу.

УРОК 1. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА ACTIONSCRIPT

Язык ActionScript

ActionScript — это событийно-управляемый язык, встроенный в инструментальную среду Macromedia/Adobe Flash. С его помощью анимации можно сделать интерактивными — выполняющими те или иные заданные действия при нажатии клавиш на клавиатуре, по командам мыши, при проигрывании определенного кадра анимации и т.д.

Принципы работы с языком ActionScript аналогичны принципам использования других языков программирования (таких, как JavaScript при создании HTML-страниц). Однако язык ActionScript во многом проще других языков программирования, для работы с ним не обязательно знать о нем все — достаточно использовать только те его возможности, которые необходимы в конкретной ситуации.

Основы работы с языком ActionScript

Прежде всего рассмотрим основные понятия, связанные с языком ActionScript.

Конструкции языка

Команды, или действия (Actions) — команды языка ActionScript (его название как раз и означает “сценарий действий”).

Переменные (Variables) — как и в других языках программирования, используются для хранения значений различных типов (числовых, строчных и т.д.), обозначенных тем или иным именем (*идентификатором*).

Операторы (Operators) — элементы языка, позволяющие выполнять вычисления, исходя из одного или более аргументов (констант и/или переменных). Это операторы сложения, вычитания, умножения, деления и т.д. При помощи операторов составляются *выражения*.

Выражения (Expressions) — записи, содержащие константы и/или переменные, соединенные операторами и позволяющие вычислять требуемые значения. В выражение могут также входить *функции*.

Функции (Functions) — программные модули (блоки программного кода), которые нацелены на выполнение определенных действий и могут выполняться многократно. Каждая функция имеет собственное имя, по которому осуществляется ее вызов, принимает один или несколько аргументов — конкретных значений (констант, переменных, выражений или других (вложенных) функций) и после выполнения вычислений с этими аргументами возвращает полученный результат. В ActionScript имеется большое число готовых (*встроенных*) функций, но можно и создавать собственные функции. Пример встроенной функции: `gotoAndPlay()` (“перейти и воспроизвести”) — в качестве аргумента она использует номер кадра анимации и предписывает перейти на этот кадр и начать с него воспроизведение анимации.

Идеология объектов, классов и экземпляров

Объект (Object) — некоторая целостная сущность, с которой (или над которой) можно выполнять те или иные действия. Так, объектами являются кнопки, клипы (MovieClip) и т.д., созданные в анимации. Каждый объект имеет *свойства* — определенные характеристики, которые можно менять. Среди таких свойств можно назвать размеры объекта (его ширину и высоту), координаты его “точки привязки” на экране, угол поворота, длитель-

ность (для звука) и т.д. Объект также может иметь особые функции, называемые *методами*, — характерные для него действия, позволяющие управлять объектом. Например, объект “звук” имеет методы **start** и **stop**, позволяющие запустить или остановить его воспроизведение, методы **getVolume** и **setVolume**, позволяющие получить текущее значение громкости или установить новое значение громкости, и т.д.

Класс (Class) — это некоторая “типовая” разновидность объектов с характерным для нее набором свойств и методов. Так, например, одним из классов является “кнопка” (Button). При этом нужно понимать, что класс — это некая абстрактная (теоретическая) сущность. Если обратиться к аналогиям, то можно рассматривать в качестве одного из классов “дерево”. Такой класс обладает таким набором свойств, как “глубина корней”, “толщина ствола”, “количество ветвей”, “форма листьев”, при этом его характерная особенность — одиночный твердый одревеснев-ший ствол. В ActionScript есть предопределенные классы (подобные имеющимся в JavaScript), но можно создавать и собственные классы или модифицировать существующие.

Экземпляр (Instance) — конкретный объект, созданный “по образцу” того или иного класса. Например, для класса “дерево” экземплярами будут объекты “береза”, “клен” или “дуб”. Экземпляр — это реальный объект, воплощение, реализация класса. Экземпляр может иметь конкретное имя, позволяющее обращаться к методам и свойствам объекта. Обычно при создании экземпляра класса получающийся объект-экземпляр *наследует* типовые свойства и методы своего класса, а вот значения свойств для каждого экземпляра будут своими.

Идеология событий

События (Events) — различные ситуации, которые могут происходить в ходе воспроизведения флеш-анимации. События могут быть предопределенными — например, когда заканчивается загрузка некоторого кадра либо когда воспроизводится конкретный кадр, или в какой-то мере случайными (заранее “непредсказуемыми” для программы на языке ActionScript), — например, когда пользователь нажимает ту или иную клавишу на клавиатуре или наводит на объект курсор мыши.

Обработчик (Handler) — специальная команда, обрабатывающая то или иное событие. Например, **onClipEvent** — это обработчик, связанный с конкретным символом (Symbol), созданным в анимации: этот обработчик включается, если щелкнуть мышью на соответствующем символе. Обработчик может быть “встроенным” или созданным самим программистом.

Когда происходит то или иное событие, выполняется его *перехват*: срабатывает предусмотренный в программном коде ActionScript программный модуль, в заголовке которого записана команда **on()** с указанием в скобках требуемого события. Например, программный модуль

```
on(press) {
. . .
}
```

срабатывает при нажатии (событие — **press**) кнопки (Button), к которой “привязан” этот программный модуль. Соответственно, для каждого объекта и каждого происходящего с ним события, для которых нужно предусмотреть реакцию компьютера, нужно написать свой обработчик, реализующий требуемые действия. (На события, для которых данному объекту не назначены обработчики, никакой реакции не происходит.)

Панель действий (Actions Panel)

Панель действий служит для работы с языком ActionScript — для ввода и отображения ActionScript-программ (*рис.* 1).

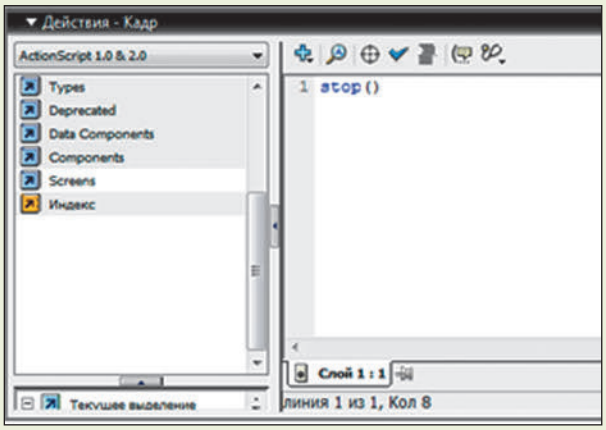


Рис. 1. Панель действий

При работе с панелью действий можно вводить команды ActionScript непосредственно с клавиатуры в расположенном справа поле либо выбирать их из списка элементов языка (слева). При этом в поле ввода программы, начиная с текущей позиции курсора, может быть вставлено как выбранное слово (например, название свойства), так и целая программная конструкция (например, при выборе обработчика), которую затем нужно “наполнить” — вписать в нее те или иные команды. То же можно сделать, нажав на кнопку “+” в панели над полем ввода текста программы и выбрав желаемый пункт многоуровневого меню (по структуре оно совпадает со списком слева). Кроме того, можно использовать “горячие клавиши”, записанные в меню кнопки “+” справа от некоторых пунктов меню. Например, добавить функцию **stop()** можно комбинацией клавиш Esc + **st**. Редактировать и удалять команды ActionScript можно в поле их ввода обычным способом (как в любом текстовом редакторе).

УРОК 2. УПРАВЛЕНИЕ ВОСПРОИЗВЕДЕНИЕМ АНИМАЦИИ: “СЛАЙД-ШОУ”

События

on (*событие*) { *программный код* } — обработчик событий, назначаемый для какого-либо объекта (например, кнопки) для реализации реакции компьютера на событие, совершаемое над этим объектом. События для этого обработчика:

press — нажатие кнопки мыши, когда курсор находится в пределах кнопки;

release — отпускание кнопки мыши, когда курсор находится в пределах кнопки.

Функции управления работой анимации

stop() — останов воспроизведения анимации на данном кадре;

play() — запуск воспроизведения анимации с данного кадра;

gotoAndStop(*номер кадра*) — переход на кадр с указанным номером и останов воспроизведения анимации на данном кадре;

gotoAndPlay(*номер кадра*) — переход на кадр с указанным номером и запуск воспроизведения анимации с данного кадра;

_root.prevFrame () — переход на предыдущий кадр;

_root.nextFrame () — переход на следующий кадр.

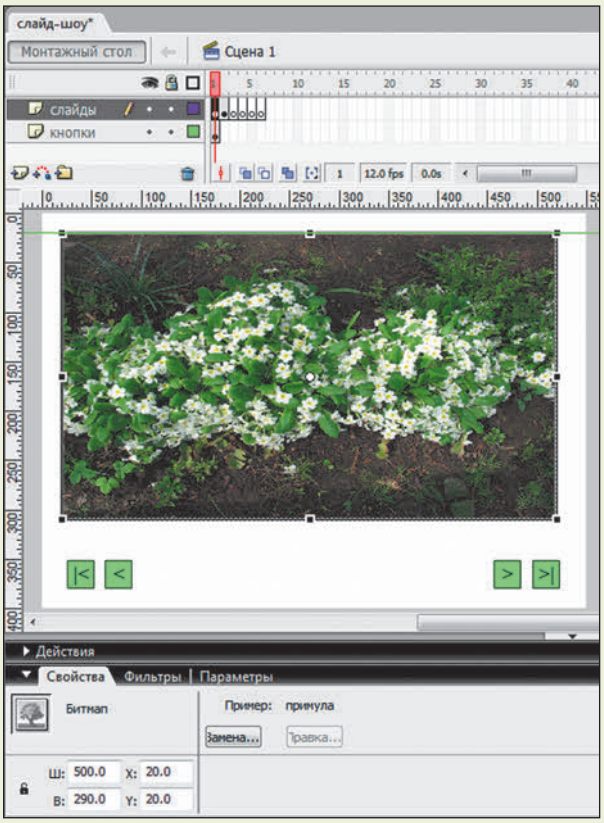
Практические задания

1. Создадим новую анимацию. Создадим в ней два слоя с именами “кнопки” и “слайды”.

В слое “кнопки” в первом ключевом кадре создадим четыре кнопки (символа типа “кнопка”), разместив их симметрично по краям:

2. В слое “слайды” создадим новые ключевые кадры, расположенные подряд друг за другом. Количество этих ключевых кадров должно быть равно количеству слайдов, включаемых в слайд-шоу.

3. Перейдем на первый ключевой кадр слоя “слайды”. Перетащим на него первую фотографию (она импортируется как растровое изображение в библиотеку флеш-редактора). Определим размещение и размеры фотографии так, чтобы она аккуратно занимала верхнюю часть поля анимации. Для этого сначала масштабируем и разместим фотографию вручную. Затем, выделив ее мышью, посмотрим значения ее ширины/высоты и координат точки привязки в соответствующих полях панели свойств. Заменим имеющиеся там значения на целые числа (желательно — кратные 10). Запомним эти значения.



Перейдем на второй кадр слоя “слайды”. Перетащим на него вторую фотографию. Для ее масштабирования и размещения выделим ее мышью и введем в поля ширины/высоты и координат те же самые значения, что и на первом кадре.

Повторим указанные действия для всех остальных кадров, размещая на них остальные фотографии.

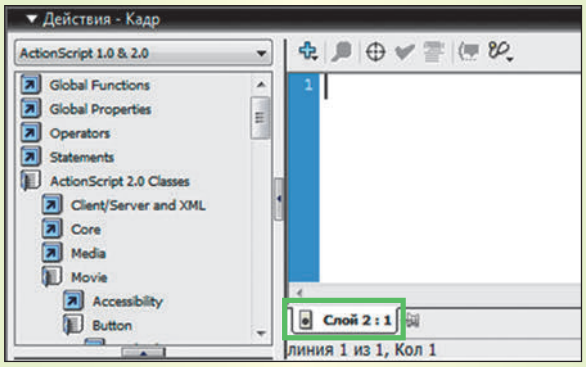
4. Вновь перейдем на первый кадр слоя “кнопки”. Перетащим на него из библиотеки первую из ранее загруженных туда фотографий. Масштабируем ее, сильно уменьшив и превратив в миниатюру. Разместим ее между ранее созданными кнопками при помощи встроенных средств выравнивания. При этом размеры и размещение миниатюры подберем так, чтобы между ранее созданными кнопками разместились миниатюры всех фотографий. Для задания размеров миниатюр фотографий используем тот же прием, что и при масштабировании фотографий в слое “слайды” (масштабируем первую миниатюру вручную — заменив значения ширины/высоты в полях свойств объекта на целые, желательно кратные 10 — и установим для остальных миниатюр те же самые значения ширины/высоты).

В результате содержимое слоя “кнопки” должно принять вид, подобный следующему:

5. Чтобы полученная анимация не проигрывалась автоматически (все слайды не “проскакивали”

за доли секунды), нужно назначить каждому кадру слоя “слайды” программный код из одной-единственной команды — вызова функции **stop()**.

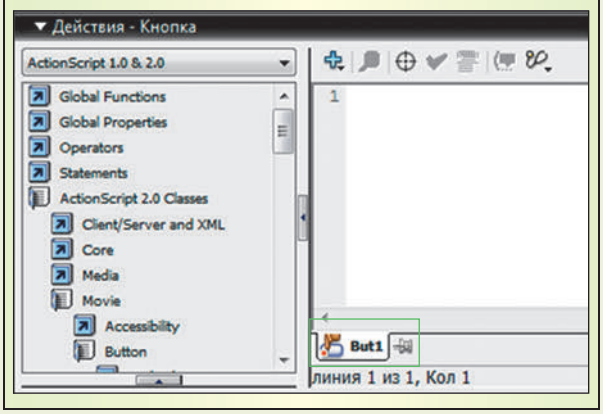
Чтобы назначить программный код для кадра анимации, нужно сначала выделить в панели **Timeline** нужный кадр, а затем раскрыть панель **Действия (Actions)**. При этом в нижнем левом углу панели **Действия (Actions)** будет указано имя слоя и через двоеточие — номер кадра.



6. Кнопке |< назначим программный код, реализующий переход на первый кадр анимации.

Для этого выделим кнопку |< и раскроем панель **Действия** (**Actions**).

Чтобы назначить программный код объекту (например, кнопке), нужно сначала выделить этот объект, а затем раскрыть панель **Actions**. Имя объекта, которому присваивается программный код, указывается в нижнем левом углу панели.



Программный код для кнопки:

```
on (press) {
  gotoAndStop(1);
}
```

7. Кнопке >| назначим программный код, реализующий переход на последний кадр анимации (номер последнего кадра равен количеству слайдов).

```
on (press) {
  gotoAndStop(6);
}
```

8. Кнопке < назначим программный код, реализующий переход на предыдущий кадр анимации.

```
on (press) {
  _root.prevFrame();
  stop();
}
```

9. Кнопке > назначим программный код, реализующий переход на предыдущий кадр анимации.

```
on (press) {
  _root.nextFrame();
  stop();
}
```

10. Миниатюры слайдов превратим в кнопки. Для этого нужно выделить очередную миниатюру и преобразовать ее в символ типа “кнопка” (размеры и расположение объекта при этом останутся без изменений).

Каждой кнопке — миниатюре слайда назначим программный код, соответствующий переходу на слайд с соответствующей фотографией. Ниже показан программный код для миниатюры первого слайда.

```
on (release) {
  gotoAndStop(1);
}
```

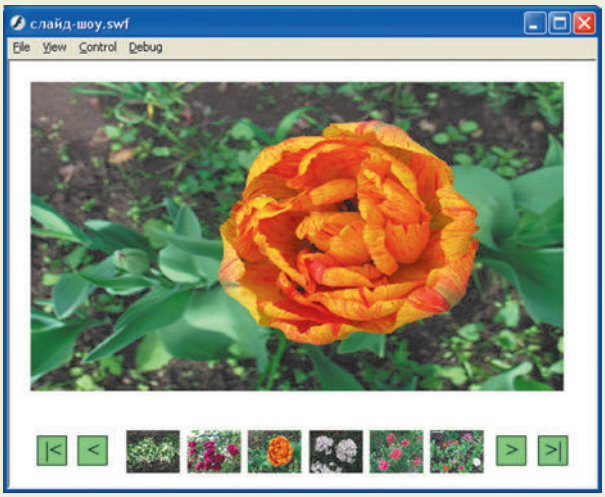
(в скобках после **gotoAndStop** указываются номера соответствующих кадров).

11. В слое “кнопки” создадим ключевые кадры в позициях, соответствующих второму, предпоследнему и последнему ключевым кадрам слоя “слайды”. Благодаря этому созданные кнопки (вместе с их программным кодом) будут иметься на всех кадрах.

Поскольку на первом кадре переход на предыдущий кадр не имеет смысла, на первом кадре слоя “кнопки” удалим кнопку <.

Аналогично, поскольку на последнем кадре переход на следующий кадр не имеет смысла, на последнем кадре слоя “кнопки” удалим кнопку >.

12. Запустим созданную анимацию на выполнение. Проверим ее работу — переходы по слайдам при помощи кнопок |<, <, >, >| и кнопок-миниатюр слайдов.

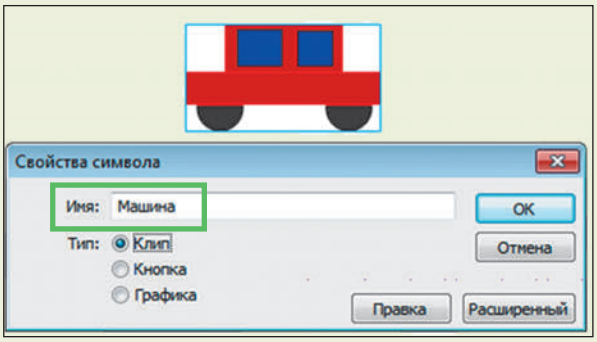


УРОК 3. УПРАВЛЕНИЕ РАЗМЕРАМИ ОБЪЕКТОВ

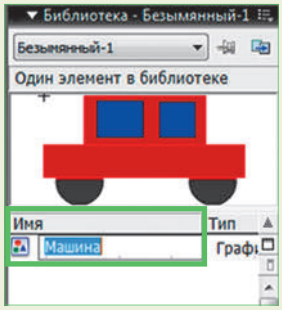
Имя объекта

Имя объекта — это имя конкретного экземпляра символа. Например, символ имеет имя “Машина”, а имена экземпляров — “Машина_1”, “Машина_2”, “Мерседес”, “Пикап” и т.д.

Присваивание имени символу при его создании:



Изменение имени символа в библиотеке:



Двойной щелчок мышью на имени объекта в библиотеке для переименования

Задание имени экземпляра символа (только для клипов MovieClip и кнопок Button):

Путь к объекту

Во Flash объекты (символы) могут быть вложены друг в друга, образуя иерархию. В этом случае можно из объекта обращаться напрямую по именам только к объектам, вложенным в него (стоящим на один уровень глубже в иерархии объектов). Например, символ “**Машина**” может быть собран из объектов “**Колесо1**” и “**Колесо2**”, тогда из программного кода объекта “**Машина**” можно обращаться к объектам “**Колесо1**” и “**Колесо2**”. При обращении к объектам на более глубоких уровнях иерархии нужно через точку записать последовательность имен объектов по пути переходов по ним по уровням вложенности. Например, из объекта “**Машина**” можно записать путь обращения к объекту “**Колесо.Спица**”.

Специальные имена и элементы пути

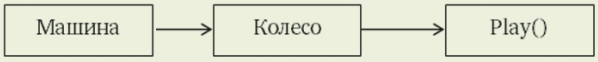
`this` — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

`_parent` — имя “родительского” объекта (стоящего на один уровень выше текущего объекта);

`_root` (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

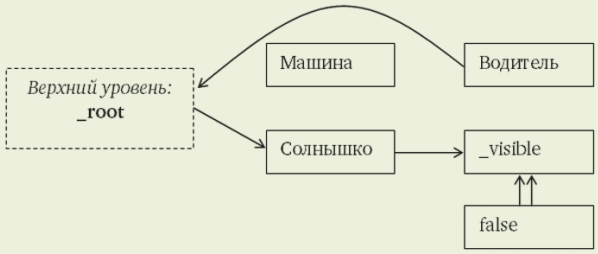
Примеры:

1) **`Колесо.play();`** — для вложенного в объект “**Машина**” объекта-клипа “**Колесо**” вызвать функцию **`play()`** — запуск анимации этого клипа;



2) **`_parent.Грузовик.Колесо.stop();`** — обращение из объекта “**Машина**” к объекту “**Грузовик**” (размещенному на том же уровне вложенности), а далее — к его вложенному объекту “**Колесо**” для вызова для него функции **`stop ()`** — останова анимации клипа;

3) **`_root.Солнышко._visible = false;`** – обращение к объекту “**Солнышко**”, расположенному на самом верхнем уровне иерархии, к его свойству **`_visible`** (видимость): присваивание ему значения **`false`** делает указанный объект невидимым.



Свойства объекта (клипа, кнопки)

`_x`, `_y` — координаты объекта (в пикселях);

`_xscale`, `_yscale` — масштаб изображения (в процентах) по горизонтали и по вертикали относительно исходных размеров объекта;

`_width`, `_height` — ширина и высота объекта (в пикселях).

События

`on (событие) { программный код }` — обработчик событий, назначаемый для какого-либо

объекта (например, кнопки) для реализации реакции компьютера на событие, совершаемое над этим объектом. События для этого обработчика:

press — нажатие кнопки мыши, когда курсор находится в пределах кнопки;

release — отпускание кнопки мыши, когда курсор находится в пределах кнопки;

keyPress ("*клавиша*") — нажатие указанной клавиши на клавиатуре.

Названия специальных клавиш:

Клавиша	Название
Курсор влево	<Left>
Курсор вправо	<Right>
Курсор вниз	<Down>
Курсор вверх	<Up>
В начало (HOME)	<Home>
В конец (END)	<End>
На страницу вверх (PAGEUP)	<PageUp>
На страницу вниз (PAGEDOWN)	<PageDown>
Вставка (INSERT)	<Insert>
Удаление (DEL)	<Delete>
Стирание последнего символа (“забой”)	<Backspace>
Табуляция	<Tab>
Ввод	<Enter>
Отказ (ESC)	<Escape>
Пробел	<Space>

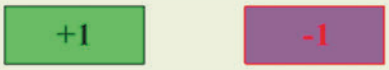
Практические задания

1) Создадим символ (MovieClip) с изображением автомобиля:

Имя символа в библиотеке: **Avto**.

Имя экземпляра, размещенного на слое анимации: **Mobil**.

2) В отдельном слое создадим две кнопки с надписями “+1” и “–1”.



Имена объектов в библиотеке: **Button1** и **Button2**.

Имена экземпляров: **But1** и **But2**.

3) Напишем программный код для кнопки “+1”. Для этого выделим кнопку 

и раскроем панель **Действия** (**Actions**).

Программный код для кнопки “+1”:

```
on (press)
{
  _root.Mobile._width =
  _root.Mobile._width + 1;
  _root.Mobile._height =
  _root.Mobile._height + 1;
}
```

Проанализируйте этот программный код. Определите, на какое событие, происходящее с объектом “кнопка”, рассчитан данный программный код, как осуществляется обращение к объекту **Mobile**, к какому его свойству и как меняется значение этого свойства.

Объясните смысл строк программного кода:

`on (press)` — обработка события _____;

`_root.Mobile._width` — обращение к свойству _____ объекта с именем _____, расположенного на _____ уровне иерархии, из другого объекта — кнопки;

`_root.Mobile._height` — обращение к свойству _____ объекта с именем _____, расположенного на _____ уровне иерархии, из другого объекта — кнопки;

`_root.Mobile._width = _root.Mobile._width+1;` — изменение значения свойства _____ — его _____ на ____;

`_root.Mobile._height = _root.Mobile._height+1;` — изменение значения свойства _____ — его _____ на ____.

Ответы:

`on (press)` — обработка события “*нажатие мышью экранной кнопки*”;

`_root.Mobile._width` — обращение к свойству *_width (ширина)* объекта с именем *Mobile*, расположенного на *самом верхнем* уровне иерархии, из другого объекта — кнопки;

`_root.Mobile._height` — обращение к свойству *_height (высота)* объекта с именем *Mobile*, расположенного на *самом верхнем* уровне иерархии, из другого объекта — кнопки;

`_root.Mobile._width = _root.Mobile._width + 1;` — изменение значения свойства*_width (ширина)* — его *увеличение* на *1 пиксель*;

`_root.Mobile._height = _root.Mobile._height + 1;` — изменение значения свойства*_height (высота)* — его *увеличение* на *1 пиксель*.

4) Запустим анимацию на исполнение. Нажимая мышью экранную кнопку “+1”, посмотрим, что происходит с объектом.

5) Напишем программный код для кнопки “–1”:

```
on (press)
{
  _root.Mobile._width =
  _root.Mobile._width - 1;
  _root.Mobile._height =
  _root.Mobile._height - 1;
}
```

Запустим анимацию на исполнение. Нажимая мышью экранные кнопки “–1” и “+1”, посмотрим, что происходит с объектом.

6) Изменим программный код для кнопок “–1” и “+1” так, чтобы изменение размеров объекта **Mobile** происходило с шагом 10 пикселей.

7) Создадим четыре кнопки любого вида, размещенные вне поля анимации.

8) Назначим двум из этих кнопок фрагменты программного кода.

Программный код для первой кнопки — реакция на нажатие на клавиатуре клавиши управления курсором ← (“влево”):

```
on(keyPress "left") {
  _root.Mobile._width -= 1;
  _root.Mobile._height -= 1;
}
```

> Использование операций “`-=`” и “`+=`” позволяет сократить запись программного кода.
>
> Операция “`-=`” означает уменьшение значения переменной или свойства, записанного слева от знака этой операции, на величину, указанную справа от нее.
>
> Операция “`+=`” означает увеличение значения переменной или свойства, записанного слева от знака этой операции, на величину, указанную справа от нее.

Программный код для второй кнопки — реакция на нажатие на клавиатуре клавиши управления курсором → (“вправо”):

```
on(keyPress "right") {
    _root.Mobile._width += 1;
    _root.Mobile._height += 1;
}
```

9) Назначим остальным двум кнопкам фрагменты программного кода.

Программный код для третьей кнопки — реакция на нажатие на клавиатуре клавиши управления курсором ↓ (“вниз”):

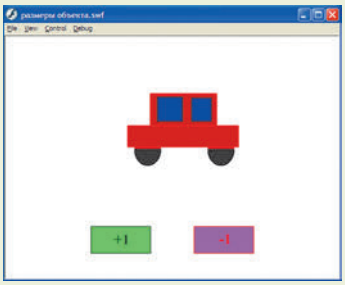
```
on(keyPress "down") {
  _root.Mobile._width -= 10;
  _root.Mobile._height -= 10;
}
```

Программный код для четвертой кнопки — реакция на нажатие на клавиатуре клавиши управления курсором ↑ (“вверх”):

```
on(keyPress "up") {
    _root.Mobile._width += 10;
    _root.Mobile._height += 10;
}
```

10) Запустим анимацию на выполнение. Проверим:

- при нажатии клавиши “курсор влево” размеры объекта должны уменьшаться на 1 пиксель;
- при нажатии клавиши “курсор вправо” размеры объекта должны увеличиваться на 1 пиксель;
- при нажатии клавиши “курсор вниз” размеры объекта должны уменьшаться на 10 пикселей;
- при нажатии клавиши “курсор вверх” размеры объекта должны увеличиваться на 10 пикселей.



УРОК 4. УСЛОВНЫЙ ОПЕРАТОР

Имя объекта

Имя объекта — это имя конкретного экземпляра символа. Например, символ имеет имя “Машина”, а имена экземпляров — “Машина_1”, “Машина_2”, “Мерседес”, “Пикап” и т.д.

Специальные имена и элементы пути

`this` — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

`_parent` — имя “родительского” объекта (стоящего на один уровень выше текущего объекта);

`_root` (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

Условный оператор

`if ( условие ) { программный код }` — программный код выполняется, если условие в круглых скобках истинно (неполная форма условного оператора);

`if ( условие ) { программный код 1 }else { программный код2 }` — в зависимости от истинности условия выполняется или программный код 1(если условие истинно), или программный код 2 (если условие ложно) (полная форма условного оператора).

Операции сравнения:

Знак операции	Описание
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
==	равно
!=	не равно

Объединение нескольких условий логическими операциями:

Знак операции	Описание
&&	И
||	ИЛИ
!	НЕ

Условие, проверяющее нажатие клавиши на клавиатуре: **`Key.isDown(`** ***`клавиша`*** **`)`**, где в скобках указывается числовой код клавиши, символ (в кавычках) или название (для специальных клавиш). Данное условие может проверять одновременное нажатие нескольких клавиш.

Названия специальных клавиш в записи условия **`Key`**:

Клавиша	Название
Курсор влево	`Key.LEFT`
Курсор вправо	`Key.RIGHT`
Курсор вниз	`Key.DOWN`
Курсор вверх	`Key.UP`
В начало (HOME)	`Key.HOME`
В конец (END)	`Key.END`
На страницу вверх (PAGEUP)	`Key.PGUP`
На страницу вниз (PAGEDOWN)	`Key.PGDN`
Alt	`Key.ALT`
Ctrl	`Key.CONTROL`
Shift	`Key.SHIFT`
Caps Lock	`Key.CAPSLOCK`
Вставка (INSERT)	`Key.INSERT`
Удаление (DEL)	`Key.DELETEKEY`
Стирание последнего символа (“забой”)	`Key.BACKSPACE`
Табуляция	`Key.TAB`
Ввод	`Key.ENTER`
Отказ (ESC)	`Key.ESCAPE`
Пробел	`Key.SPACE`

Свойства объекта (клипа, кнопки)

`_x`, **`_y`** — координаты объекта (в пикселях);

`_width`, **`_height`** — ширина и высота объекта (в пикселях);

`_visible` — видимость объекта.

События

`on (`*`событие`*`) { `*`программный код`*` } — обработчик событий, назначаемый для какого-либо объекта (например, кнопки) для реализации реакции компьютера на событие, совершаемое над этим объектом. События для этого обработчика:

`press` — нажатие кнопки мыши, когда курсор находится в пределах кнопки;

`release` — отпускание кнопки мыши, когда курсор находится в пределах кнопки;

`onClipEvent(enterFrame) {` ***`программный код`*** **`}`** — обработчик событий, назначаемый клипу. Здесь используется событие **`enterFrame`** — “запуск данного кадра на воспроизведение”.

Если в анимации содержится только один кадр, которому назначен данный обработчик, то при воспроизведении такой анимации этот кадр зацикливается, так что данный обработчик выполняется постоянно и записанный в нем программный код будет выполняться все время;

`onClipEvent(load) {` ***`программный код`*** **`}`** — обработчик событий, назначаемый клипу. Здесь используется событие **`load`** — “загрузка анимации”. Оно выполняется однократно, поэтому его программный код может быть использован для задания исходных значений переменных.

Практические задания

1. Импортируем в флеш-редактор рисунок. Создадим на его базе символ (MovieClip):

Имя символа в библиотеке: **Мяч**.

Имя экземпляра, размещенного на слое анимации: **Ball**.

2. Напишем программный код, предусматривающий реакцию на нажатие клавиш управления курсором ← и →. В отличие от предыдущего занятия этот программный код будет привязан не к кнопкам, расположенным вне поля анимации, а непосредственно к объекту-клипу.

> Чтобы назначить программный код объекту, нужно сначала выделить его, а затем раскрыть панель **Действия (Actions)**.

Программный код:

```
onClipEvent(enterFrame) {
  if (Key.isDown(Key.LEFT)) {
    this._x -= 1;
  }
  if (Key.isDown(Key.RIGHT)) {
    this._x += 1;
  }
}
```

> Определите по этому программному коду, какие клавиши задействованы и что должно происходить при их нажатии.

3. Запустим анимацию на выполнение. Проверим свои предположения. Проверим работу анимации при одновременном нажатии двух клавиш, например, “влево” и “вверх” или “вправо” и “вниз”.

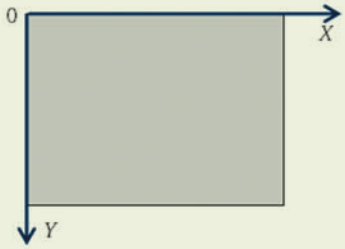
4. Изменим программный код так, чтобы увеличить скорость перемещения объекта при нажатии клавиш.

5. Допишем программный код так, чтобы реализовать перемещение объекта по вертикали при нажатии клавиш управления курсором ↑ и ↓.

Дополненный программный код:

```
onClipEvent(enterFrame) {
  if (Key.isDown(Key.LEFT)) {
    this._x -= 5;
}
  if (Key.isDown(Key.RIGHT)) {
    this._x += 5;
  }
  if (Key.isDown(Key.UP)) {
    this._y -= 5;
}
  if (Key.isDown(Key.DOWN)) {
   this._y += 5;
  }
}
```

Обратим внимание: движение объекта *вверх* программируется *уменьшением* значения свойства **_y** (вертикальной координаты), а движение *вниз* — *увеличением* значения свойства **_y**. Это связано с тем, что система координат на экране компьютера (и в том числе на поле анимации) имеет следующий вид:

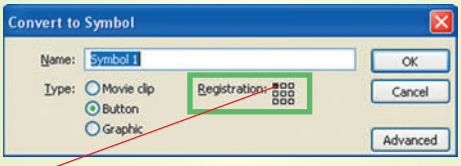


6. Изменим программный код так, чтобы объект (мячик) сам двигался в направлении, заданном нажатой клавишей, все время, пока он не уткнется в край поля анимации.

Для этого нужно использовать следующие принципы построения алгоритма программного кода:

- для указания направления движения по оси *x* используем переменную *osx*, которой при нажатии клавиши ← прибавляется значение –1, а при нажатии клавиши → — значение +1;
- аналогично, для указания направления движения по оси *y* используем переменную *osy*, которой при нажатии клавиши ↑ прибавляется значение –1, а при нажатии клавиши ↓ — значение +1 (не забываем про особенности системы координат на экране компьютера);
- зададим (однократно) ширину и высоту объекта (мячика), тогда ситуация, когда мячик достигает краев поля анимации, обозначается условиями (помним, что координаты объекта определяются координатами его верхнего левого угла):
  - $x \le 0$ пикселей — левый край,
  - $x \ge (L - xw)$ пикселей (xw — ширина объекта; L — ширина поля анимации) — правый край,
  - $y \le 0$ пикселей — верхний край,
  - $y \ge (H - yh)$ пикселей (yh — высота объекта; H — высота поля анимации) — нижний край;

В данном случае “точка привязки” объекта предполагается в его верхнем левом углу. Положение “точки привязки” устанавливается при конвертировании объекта в символ: для этого нужно щелкнуть мышью на соответствующем квадратике в мини-панели **Registration**:



Щелкнуть мышью в этой панели там, где требуется разместить “точку привязки”

- при каждом вызове программного кода нужно:
  - проверять, нажата ли клавиша, меняющая направление движения объекта, и если одно из таких условий истинно, то изменить значение соответствующей переменной (*osx* или *osy*),
  - проверить — если при дальнейшем перемещении объекта с учетом нового значения переменных *osx* и *osy* его новые координаты не попадают в краевые зоны (т.е. не выполнены соответствующие условия достижения краев поля анимации), то изменить значения свойств соответствующих координат объекта.

Определить размеры поля анимации (L и H) можно в панели свойств после щелчка мышью в любом свободном месте поля анимации:

Программный код.

```
onClipEvent(load) {

  L = 550;

  H = 400;

  xw = 50;
  yh = 50;
  osx = 0;

  osy = 0;

  xx = 0;

  yy = 0;

  this._x = 275;

  this._y = 200;

  this._width = xw;

  this._height = yh;
}
```

Присваивание исходных значений:
L — ширина поля анимации,
H — высота поля анимации,
xw — ширина объекта,
yh — высота объекта,
osx — приращение координаты по оси X,
osy — приращение координаты по оси Y,
xx — координата x объекта,
yy — координата y объекта,
исходное положение объекта (x),
исходное положение объекта (y),
задание ширины объекта,
задание высоты объекта

```
onClipEvent(enterFrame) {

  xx = this._x;
  yy = this._y;

  if (Key.isDown(Key.LEFT)) {
    osx -= 1;
  }

  if (Key.isDown(Key.RIGHT)) {
    osx += 1;
  }

  if (Key.isDown(Key.UP)) {
    osy -= 1;
  }

  if (Key.isDown(Key.DOWN)) {
    osy += 1;
  }

  if (xx + osx > 0 && xx + osx < L - xw) {
    xx += osx;
  }

  if (yy + osy > 0 && yy + osy < H - yh) {
    yy += osy;
  }

    this._x = xx;
    this._y = yy;
}
```

Основная часть программного кода:

— считать текущие значения координат объекта,

— если нажата клавиша ←, уменьшить osx,

— если нажата клавиша →, увеличить osx,

— если нажата клавиша ↑, уменьшить osy,

— если нажата клавиша ↓, уменьшить osy,

— проверка, не уходит ли объект за левый или правый край, если нет — изменение его координаты по x,

— проверка, не уходит ли объект за верхний или нижний край, если нет — изменение его координаты по y,

— перемещение объекта на новые координаты

7. Запустим анимацию на выполнение. Проверим ее работу. Первоначально мяч должен быть неподвижен. Нажатия клавиш управления курсором должны не только определять направление движения объекта, но и скорость его перемещения: повторное нажатие клавиши в том же направлении увеличивает скорость, в противоположном — уменьшает.

8. Изменим программный код так, чтобы мячик “отскакивал” от краев экрана. Для этого нужно изменить программный код внутри условных операторов проверки условий достижения объектом краев экрана: в этом случае нужно менять значение переменных *osx* или *osy* на противоположные по знаку.

Программный код:

```
onClipEvent(load) {
  L = 550;
  H = 400;
  xw = 50;
  yh = 50;
  osx = 0;
  osy = 0;
  xx = 0;
  yy = 0;
  this._x = 275;
  this._y = 200;
  this._width = xw;
  this._height = yh;
}
onClipEvent(enterFrame) {
  xx = this._x;
  yy = this._y;
  if (Key.isDown(Key.LEFT)) {
    osx -= 1;
  }
  if (Key.isDown(Key.RIGHT)) {
    osx += 1;
  }
  if (Key.isDown(Key.UP)) {
    osy -= 1;
  }
  if (Key.isDown(Key.DOWN)) {
    osy += 1;
  }
  if (xx + osx <= 0 || xx + osx >= L - xw) {
    osx = -osx;
  }
  if (yy + osy <= 0 || yy + osy >= H - yh) {
    osy = -osy;
  }
  xx += osx;
  yy += osy;
  this._x = xx;
  this._y = yy;
}
```

Самостоятельно разберите, как работает этот программный код и чем он отличается от предыдущего.

9. Запустим анимацию на выполнение. Проверим его работу — “отскакивание” мячика от краев поля анимации и изменение направления и скорости мячика нажатиями клавиш управления курсором.

УРОК 5. СТОЛКНОВЕНИЕ ОБЪЕКТОВ

Имя объекта

Имя объекта — это имя конкретного экземпляра символа. Например, символ имеет имя “Машина”, а имена экземпляров — “Машина_1”, “Машина_2”, “Мерседес”, “Пикап” и т.д.

Специальные имена и элементы пути

this — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

_root (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

Свойства объекта (клипа, кнопки)

_x, _y — координаты объекта (в пикселях);

_width, _height — ширина и высота объекта (в пикселях);

_visible — видимость объекта (**true** — видимый, **false** — невидимый).

Математические функции

Math.sin() — синус значения в скобках (угол в радианах);

Math.cos() — косинус значения в скобках (угол в радианах);

Math.round() — округление значения в скобках до ближайшего целого (по правилам математики);

Math.random() — случайное значение от 0 до 1;

Math.random()*N — случайное значение от 0 до N;

(N - M)*Math.random() + M — случайное значение от M до N.

События

onClipEvent(enterFrame) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **enterFrame** — “запуск данного кадра на воспроизведение”.

Если в анимации содержится только один кадр, которому назначен данный обработчик, то при воспроизведении такой анимации этот кадр зацикливается, так что данный обработчик выполняется постоянно и записанный в нем программный код будет выполняться все время.

onClipEvent(load) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **load** — “загрузка анимации”. Оно выполняется однократно, поэтому его программный код может быть использован для задания исходных значений переменных.

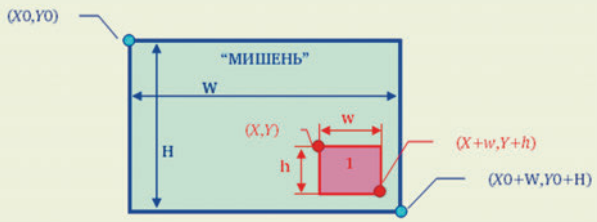
Условие, проверяющее нажатие клавиши на клавиатуре: **Key.isDown(*клавиша*)**, где в скобках указывается числовой код клавиши, символ (в кавычках) или название (для специальных клавиш). Данное условие может проверять одновременное нажатие нескольких клавиш.

Названия специальных клавиш в записи условия **Key**:

Клавиша	Название
Курсор влево	Key.LEFT
Курсор вправо	Key.RIGHT
Курсор вниз	Key.DOWN
Курсор вверх	Key.UP
Пробел	Key.SPACE

Определение “столкновения” объектов путем вычислений координат

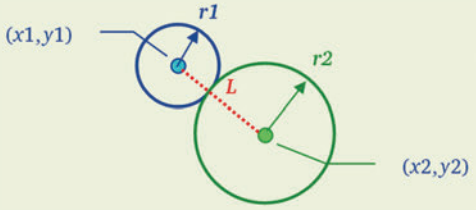
(предполагается, что “точка привязки” объектов размещена в верхнем левом углу)



Определение полного нахождения объекта (**1**) внутри указанной области (“мишени”):

$$\begin{cases} (X \ge X0) \text{ и } ((X+w) \le (X0+W)), \\ (Y \ge Y0) \text{ и } ((Y+h) \le (Y0+H)). \end{cases}$$

Определение внешнего соприкосновения объектов — задача более сложная, поскольку в этом случае нужно учитывать форму объектов и все возможные случаи касания (если не удается вывести какую-то общую формулу условия касания). В наиболее простом случае — для касания двух окружностей — такую общую формулу условия касания можно вывести через расстояние между их центрами:

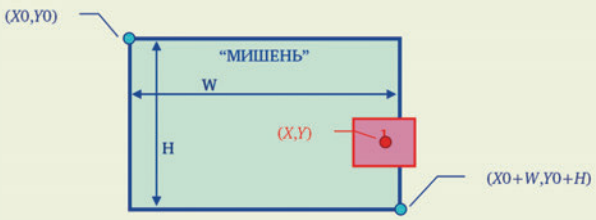


Круги пересекаются, если расстояние между их центрами: $L = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ меньше, чем сумма радиусов ($r1 + r2$).

Упрощенное условие попадания объекта в другой объект

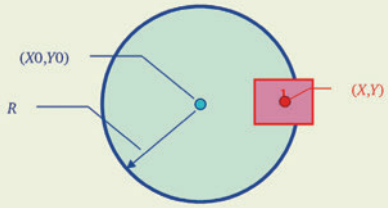
В упрощенном случае можно считать, что объект-“снаряд” попал в объект-“мишень”, если точка привязки объекта-“снаряда” (расположенная в его середине!) находится в пределах области, занятой объектом-“мишенью”, при этом ширина и высота “снаряда” не учитываются. В этом случае объект-“снаряд” может при попадании частично выходить за пределы области “мишени”, но бóльшая часть “снаряда” будет все же располагаться поверх “мишени”.

Упрощенное условие попадания объекта-“снаряда” в прямоугольную “мишень”:



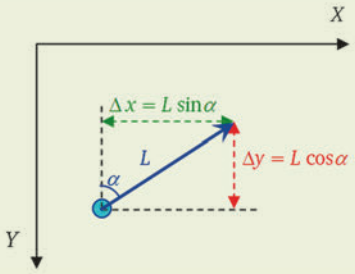
$$\begin{cases} (X \ge X0) \text{ и } (X \le (X0+W)), \\ (Y \ge Y0) \text{ и } (Y \le (Y0+H)). \end{cases}$$

Упрощенное условие попадания объекта-“снаряда” в круговую “мишень” (“точка привязки” “мишени” — в ее середине):



$$(X-X0)^2+(Y-Y0)^2 \le R^2$$

Движение под углом



Для перемещения объекта на расстояние L под углом α нужно изменить его текущие координаты (x,y) на $(x+L\sin\alpha, y+L\cos\alpha)$.

Практические задания

I. Задание 1.

1. Создадим новую анимацию.

Нарисуем прямоугольную рамку. Преобразуем ее в символ Movie Clip. Дадим имя экземпляру этого объекта: “**mishen**”. В его свойствах откорректируем значения ее координат и ширины/высоты на ближайшие кратные 10.

Создадим новый слой с именем “Снаряд”. В нем нарисуем кружочек. Преобразуем его в символ Movie Clip. Разместим его внизу кадра, примерно посередине. В свойствах объекта откорректируем значения координат X,Y на ближайшие кратные 10 и установим значения ширины/высоты равными 5 (тем самым масштабируя объект).

Создадим новый слой с именем “Надпись”. Разместим в нем (над прямоугольной рамкой) надпись “ПОПАЛ”. Преобразуем ее в символ Movie Clip. Дадим имя экземпляру этого объекта: “**nadpis**”.

2. Присвоим созданному символу (кружочку) программный код, состоящий из двух частей:

• первая часть — для события **load** (запуск анимации) задание начальных значений переменных и значения свойства “видимость” объекта **Nadpis**, чтобы сделать его невидимым;

• вторая часть — уже знакомый нам код управления перемещением объекта нажатиями клавиш управления курсором.

Программный код:

```
onClipEvent(load) {
  Wm = _root.mishen._width;
  Hm = _root.mishen._height;
  Xm = _root.mishen._x;
  Ym = _root.mishen._y;
  xs = this._x;
  ys = this._y;
  ws = this._width;
  hs = this._height;
  W = 550;
  H = 400;
  _root.nadpis._visible = false;
}
```

```
Первоначальные установки:
– считали ширину мишени,
– считали высоту мишени,
– считали координату X мишени,
– считали координату Y мишени,
– считали координату X снаряда,
– считали координату Y снаряда,
– считали ширину снаряда,
– считали высоту снаряда,
– ширина кадра анимации;
– высота кадра анимации;
– сделали надпись невидимой
```

```
onClipEvent(enterFrame) {
if (Key.isDown(Key.LEFT)) {
  xs -= 10;
}
if (Key.isDown(Key.RIGHT)) {
  xs += 10;
}
if (Key.isDown(Key.UP)) {
  ys -= 10;
}
if (Key.isDown(Key.DOWN)) {
  ys += 10;
}
if (xs <= 0) {
  xs = 0;
}
if (xs >= W-ws) {
  xs = W-ws;
  }
if (ys <= 0) {
  ys = 0;
}
if (ys >= H - hs) {
  ys = H-hs;
}
  this._x = xs;
  this._y = ys;
if ((xs >= Xm) && ((xs + ws) <= (Xm + Wm))
&& (ys >= Ym) && ((ys + hs) <= (Ym + Hm))
) {
  _root.nadpis._visible = true;
} else {
  _root.nadpis._visible = false;
}
}
```

```
Повторяющиеся действия в цикле:
— если "влево",
  то снаряд на 6 пикселей влево,

— если "вправо",
  то снаряд на 6 пикселей вправо,

— если "вверх",
  то снаряд на 6 пикселей вверх,

— если "вниз",
  то снаряд на 6 пикселей вниз,

— если снаряд ушел за левый край,
  то вернуть его к краю,

— если снаряд ушел за правый край,
  то вернуть его к краю,

— если снаряд ушел за верхний край,
  то вернуть его к краю,

— если снаряд ушел за нижний край,
  то вернуть его к краю,

— установить снаряд по новым
  координатам,
— проверка попадания в мишень,


— при попадании — сделать надпись
  видимой, иначе — снова невидимой.
```

3. Запустим анимацию. Проверим ее работу: нажатиями клавиш перемещаем “снаряд”, проверяем его невыход за границы кадра, затем проверяем появление надписи, когда “снаряд” пересекает границу “мишени” с любой ее стороны, и исчезновение надписи, когда “снаряд” перемещен вне ее.

II. Задание 2.

1. Напишем сценарий игры:

• снаряд (кружок малого размера) изначально расположен на нижнем крае кадра (заданы начальные координаты и смещение $L = 0$);

• в случайном месте кадра установлена мишень (другой объект): его размеры заданы жестко, а координаты определяются случайным образом;

• изначально значение угла α равно нулю (полет по вертикали); клавиши “курсор влево” и “курсор вправо”, соответственно, уменьшают или увеличивают значение α;

• по нажатию клавиши “ПРОБЕЛ” значение смещения L приравнивается 5;

• снаряд движется на расстояние $L = 5$ пикселей под углом α;

• выполняется проверка касания снаряда и мишени; если оно произошло, то задается смещение $L = 0$ (снаряд останавливается) и делается видимым объект “взрыв”, причем на таком месте, что его середина примерно соответствует снаряду;

• если снаряд ушел за края экрана, то для него задаются опять начальные координаты на нижнем крае кадра и смещение $L = 0$.

2. Создадим (импортируем) в новую анимацию изображение мишени. Преобразуем его в символ Movie Clip. Разместим его вне кадра. Дадим ему имя экземпляра “**mishen**”.

Нарисуем снаряд — маленький кружочек и разместим его в середине у нижнего края кадра (ширина и высота кружочка **ws = hs = 20**, координата **xs** = ½ ширины кадра, координата **ys** = высота кадра минус 20). Преобразуем его в символ Movie Clip.

Нарисуем (или импортируем) изображение взрыва. Преобразуем его в символ Movie Clip. Разместим его вне кадра. Дадим имя экземпляра “**vzriv**”.

3. Для объекта “снаряд” назначим программный код присваивания начальных значений переменных:

```
onClipEvent(load) {
  W = 550;
  H = 400;
  ws = 20;
  hs = 20;
  xs = W/2;
  ys = H - 25;
  this._x = xs;
  this._y = ys;
  wm = 200;
  hm = 100;
  xm = (W - wm) * Math.random();
  ym = (H - 2 * hm) * Math.random();
  _root.mishen._width = wm;
  _root.mishen._height = hm;
  _root.mishen._x = xm;
  _root.mishen._y = ym;
  _root.vzriv._visible = false;
  L = 0;
  alfa = 0;
  dx = 0;
  dy = 0;

}
```

```
Срабатывает при открытии анимации:
– ширина кадра,
– высота кадра,
– ширина снаряда,
– высота снаряда,
– середина ширины кадра,
– возле нижнего края кадра,
– снаряд выставляем по его координатам,

– ширина мишени,
– высота мишени,
– случайная коорд. x мишени,
– случайная коорд. y мишени,
– "образмериваем" мишень,

– выставляем мишень по полученным
  координатам,
– "взрыв" делаем невидимым,
– изначальное смещение снаряда,
– изначальный угол полета снаряда,
– рабочие переменные
```

4. Запуская анимацию на выполнение, проверим: в кадре каждый раз объект “снаряд” расположен в одном и том же месте, а объект “мишень” каждый раз появляется в разных местах.

5. Добавим для объекта “снаряд” после уже введенного программного кода новый программный код, реализующий “выстрел” по нажатию клавиши “ПРОБЕЛ” и перемещение “снаряда”:

```
onClipEvent(enterFrame) {
if (Key.isDown(Key.SPACE)) {
  L = 5;
}
xs = this._x;
ys = this._y;
dx = L * Math.sin(alfa);
dy = -L * Math.cos(alfa);

if (((xs + dx) <= 0) || ((xs + dx) >=
   (W - ws))) { dx = 0;
}
if (((ys + dx) <= 0) || ((ys + dy) >=
   (H - hs))) { dy = 0;
}
  xs += dx;
  ys += dy;
  this._x = xs;
  this._y = ys;
}
```

```
Повторяющиеся действия в цикле:
– если нажат "ПРОБЕЛ" ("выстрел"), то
  смещение равно 5,

– считали текущие координаты
  снаряда,
– вычислили смещение снаряда по X
  и по Y (оно отрицательное, так как
  движение — вверх, против оси Y),
– если выход за левый/правый край, то
  по X не смещаемся,

– если выход за верхний/нижний край,
  то по Y не смещаемся,

– вычисляем новые координаты снаряда,

– перемещаем снаряд в новое место.
```

6. Проверим анимацию: при нажатии клавиши “ПРОБЕЛ” снаряд должен начать лететь вверх до верхнего края.

7. Реализуем изменение значения угла полета снаряда. Для этого изменим программный код (его часть для события **enterFrame**) для “снаряда” на следующий:

```
onClipEvent(enterFrame) {
if (Key.isDown(Key.SPACE)) {
  L = 5;
}
if (Key.isDown(Key.LEFT)) {
  alfa -= 0.1;
}
if (Key.isDown(Key.RIGHT)) {
  alfa += 0.1;
}
xs = this._x;
ys = this._y;
dx = Math.round(L * Math.sin(alfa));
dy = -Math.round(L * Math.cos(alfa));
  xs += dx;
  ys += dy;
if ((xs <= 0) || (xs >= (W - ws))) {
  L = 0; xs = W/2; ys = H - 25; alfa = 0;
}
if ((ys <= 0) || (ys >= (H - hs))) {
  L = 0; xs = W/2; ys = H - 25; alfa = 0;
}
  this._x = xs;
  this._y = ys;
}
```

```
Повторяющиеся действия в цикле:
– если нажат "ПРОБЕЛ" ("выстрел"), то
  смещение равно 5,

– если нажато "ВЛЕВО", то
  угол уменьшаем,

– если нажато "ВПРАВО", то
  угол увеличиваем,
– считали текущие координаты
  снаряда,
– вычислили смещение снаряда по X
  и по Y (оно отрицательное, так как
  движение — вверх, против оси Y),
– вычисляем новые координаты снаряда,

– если выход за левый/правый край, то
  снаряд на исходные позиции,

– если выход за верхний/нижний край,
  снаряд на исходные позиции,

– перемещаем снаряд в новое место.
```

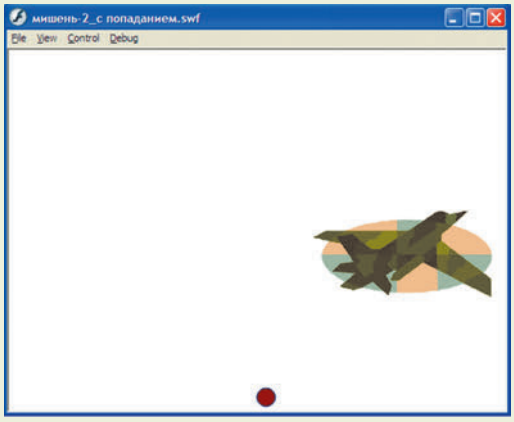
8. Запустим анимацию. Проверим: при нажатии клавиши ПРОБЕЛ снаряд должен начать двигаться по вертикали вверх, а при нажатии клавиш “Курсор влево”/“Курсор вправо” — менять направление своего движения, отклоняясь в соответствующую сторону. При достижении края экрана снаряд снова должен появляться в исходной позиции в ожидании нового “выстрела”.

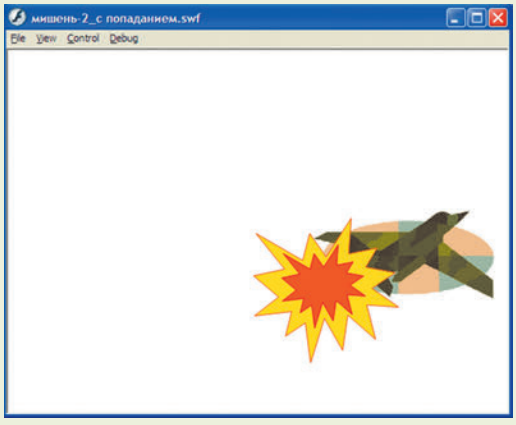
9. Добавим теперь отслеживание попадания в мишень. Для этого дополним программный код (его часть для события **enterFrame**) для “снаряда” следующим образом:

```
onClipEvent(enterFrame) {
if (Key.isDown(Key.SPACE)) {
  L = 5;
}
if (Key.isDown(Key.LEFT)) {
  alfa -= 0.1;
}
if (Key.isDown(Key.RIGHT)) {
  alfa += 0.1;
}
xs = this._x;
ys = this._y;
dx = Math.round(L * Math.sin(alfa));
dy = -Math.round(L * Math.cos(alfa));

xs += dx;
ys += dy;
if ((xs <= 0) || (xs >= (W - ws))) {
  L = 0; xs = W/2; ys = H - 25; alfa = 0;
}
if ((ys <= 0) || (ys >= (H - hs))) {
  L = 0; xs = W/2; ys = H - 25; alfa = 0;
}
  this._x = xs;
  this._y = ys;
if ( (xs >= xm) && ((xs + ws) <=
   (xm + wm)) && (ys >= ym) &&
   ((ys + hs) <= (ym + hm)) ) {
  L = 0; alfa = 0;
  _root.vzriv._x =
      (xs + ws/2)-_root.vzriv._width / 2;
  _root.vzriv._y =
     (ys + hs/2)-_root.vzriv._height / 2;
  _root.vzriv._visible = true;
      }
}
```

```
Повторяющиеся действия в цикле:
– если нажат "ПРОБЕЛ" ("выстрел"), то
  смещение равно 5,

– если нажато "ВЛЕВО", то
  угол уменьшаем,

– если нажато "ВПРАВО", то
  угол увеличиваем,

– считали текущие координаты
  снаряда,
– вычислили смещение снаряда по X
  и по Y (оно отрицательное, так как
  движение — вверх, против оси Y),
– вычисляем новые координаты снаряда,

– если выход за левый/правый край, то
  снаряд на исходные позиции,

– если выход за верхний/нижний край,
  снаряд на исходные позиции,

– перемещаем снаряд в новое место.

– проверяем условие попадания
  в мишень, если оно достигнуто, то

– останавливаем снаряд,
– позиционируем изображение "взрыва",
  располагая его центр
  по центру снаряда,

– делаем "взрыв" видимым
```

10. Запустим анимацию. Проверим: когда снаряд “наползает” на изображение мишени, анимация должна “остановиться”[1], а на месте снаряда должно появиться изображение взрыва. Для повторного “выстрела” надо закрыть анимацию и запустить ее заново.





УРОК 6. СТОЛКНОВЕНИЕ ОБЪЕКТОВ: ПОВОРОТ, ПРОГРАММИРОВАНИЕ ПОВЕДЕНИЯ НЕСКОЛЬКИХ ОБЪЕКТОВ

Специальные имена и элементы пути

this — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

_root (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

Свойства объекта (клипа, кнопки)

_x, _y — координаты объекта (в пикселях);

_width, _height — ширина и высота объекта (в пикселях);

_rotation — угол поворота объекта (в градусах);

_xscale, _yscale — масштабные коэффициенты по ширине и высоте относительно значений, заданных свойствами **_width** и **_height**, в процентах. Значение 100 соответствует исходным размерам. Изменение знака на противоположный приводит к зеркальному отражению объекта по горизонтали или по вертикали соответственно;

_visible — видимость объекта (**true** — видимый, **false** — невидимый).

Математические функции

Math.sin() — синус значения в скобках (угол в радианах);

Math.cos() — косинус значения в скобках (угол в радианах);

r = _rotation * Math.PI / 180; — пересчет угла из градусов в радианы;

Math.round() — округление значения в скобках до ближайшего целого (по правилам математики);

Math.random() — случайное значение от 0 до 1;

Math.random() * N — случайное значение от 0 до N;

(N - M) * Math.random() + M — случайное значение от M до N.

События

onClipEvent(enterFrame) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **enterFrame** — “запуск данного кадра на воспроизведение”.

Если в анимации содержится только один кадр, которому назначен данный обработчик, то при воспроизведении такой анимации этот кадр зацикливается, так что данный обработчик выполняется постоянно и записанный в нем программный код будет выполняться все время.

onClipEvent(load) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **load** — “загрузка анимации”. Оно выполняется однократно, поэтому его программный код может быть использован для задания исходных значений переменных.

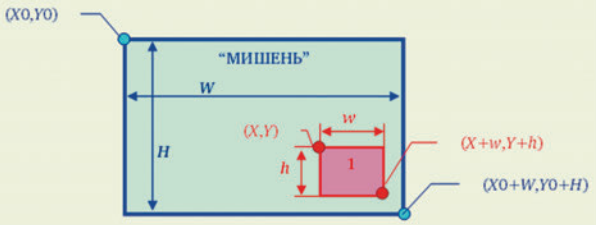
Условие, проверяющее нажатие клавиши на клавиатуре: **Key.isDown(*клавиша*)**, где в скобках указывается числовой код клавиши, символ (в кавычках) или название (для специальных клавиш). Данное условие может проверять одновременное нажатие нескольких клавиш.

Названия специальных клавиш в записи условия **Key**:

Клавиша	Название
Курсор влево	Key.LEFT
Курсор вправо	Key.RIGHT
Курсор вниз	Key.DOWN
Курсор вверх	Key.UP
Пробел	Key.SPACE

[1] На самом деле анимация продолжает выполняться, но теперь на ней ничто не движется, так что кажется, что анимация остановлена.

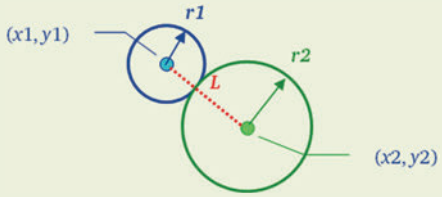
Определение “столкновения” объектов путем вычислений координат



Определение полного нахождения объекта (**1**) внутри указанной области (“мишени”):

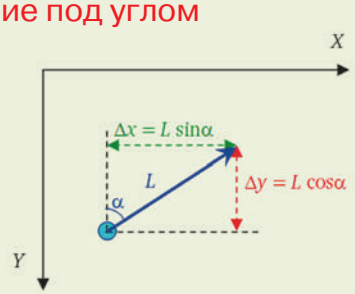
$$\begin{cases} (X \ge X0) \text{ и } ((X+w) \le (X0+W)), \\ (Y \ge Y0) \text{ и } ((Y+h) \le (Y0+H)). \end{cases}$$

Определение внешнего соприкосновения объектов — задача более сложная, поскольку в этом случае нужно учитывать форму объектов и все возможные случаи касания (если не удается вывести какую-то общую формулу условия касания). В наиболее простом случае — для касания двух окружностей — такую общую формулу условия касания можно вывести через расстояние между их центрами:



Круги пересекаются, если расстояние между их центрами: $L=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ меньше, чем сумма радиусов ($r1+r2$).

Движение под углом



Для перемещения объекта на расстояние L под углом α нужно изменить его текущие координаты (x,y) на $(x+L\sin\alpha, y+L\cos\alpha)$.

```
onClipEvent(load) {
  wm = this._width;
  xm = this._x;
  W = 550;
  dm = 20*Math.random()-10;
}
onClipEvent(enterFrame) {
if ((xm <= 0) || (xm >= W - wm)) {
  dm = -dm;
  }
  xm += dm;
  this._x = xm;
}
```

Для большей реалистичности можно также поворачивать сам объект на тот же угол, используя свойство **_rotation**. Его значение — угол поворота объекта в градусах. Для согласования этого значения со значениями функций синуса и косинуса (требующих значение угла в радианах) используется формула:

```
<угол в радианах> =
   <угол в градусах>*π/180.
```

Константа π в Action Script записывается как **Math.PI**.

Практические задания

I. Задание 1.

Продолжим работу, начатую на предыдущем уроке.

Откроем анимацию задания 2 из урока 5, где мы создавали игру — “стрелялку” снарядом по мишени.

1. Ранее стрельба производилась по неподвижной мишени. Сделаем ее движущейся влево-вправо с “отражением” от левого и правого краев кадра.

Вспомним, что задание координат мишени и ее размеров, а также ее размещение в кадре производятся в программном коде для события **load** объекта-снаряда. Поэтому для управления движением мишени мы напишем для объекта-мишени свой программный код, в котором значения координат, ширины и высоты мишени считываются как свойства этого объекта. Для события **load** объекта-мишени в программном коде случайным образом будем определять ее смещение — число из диапазона (–10,10) (отрицательное значение — движение влево, положительное — вправо, модуль значения определяет скорость движения). Для события **enterFrame** объекта-мишени напишем программный код, реализующий при каждом повторении смещение мишени на указанную величину, контроль достижения краев кадра и инверсию значения смещения, что приводит к “отражению” мишени от краев экрана.

В программном коде для объекта-мишени мы используем переменные, одноименные с использованными в программном коде для объекта-снаряда. Это допустимо, так как объекты — разные, и переменные в их программном коде “видны” только для соответствующего объекта.

```
Первоначальные установки:
— считали ширину мишени,
— считали координату X мишени,
— ширина кадра анимации;
— случайное смещение от -10 до 10
-----------------------------------
Повторяющиеся действия в цикле:
— при достижении левого/правого края
  изменить знак смещения

— вычислить новое значение координаты,
— сместить мишень
```

2. Запустим анимацию. Проверим ее работу. Отметим недостаток: при попадании мишень продолжает двигаться. Чтобы устранить эту недоработку, надо повторить в объекте-мишени проверку попадания снаряда в мишень. Проще всего это сделать, проверяя свойство видимости объекта "взрыв" ("**vzriv**"), который, как мы помним, становится у нас видимым именно в этот момент. При выполнении этого условия нужно обнулять значение смещения **dx**.

Дополненный программный код для события **enterFrame** имеет вид:

```
onClipEvent(enterFrame) {
if ((xm <= 0) || (xm >= W - wm)) {
  dm = -dm;
  }
flag = _root.vzriv._visible;

if ( flag ) {
  dm = 0;
}
  xm += dm;
  this._x = xm;
}
```

```
Повторяющиеся действия в цикле:
– при достижении левого/правого края
  изменить знак смещения

– считали свойство видимости объекта
  "взрыв",
– если оно равно true (объект видим),
  обнуляем смещение мишени,

– вычислить новое значение координаты,
– сместить мишень
```

В данном случае мишень движется несколько неестественно — после изменения направления ее движения самолет "летит хвостом вперед". Чтобы реализовать его "разворот", можно одновременно с изменением знака переменной **dm** менять знак значения свойства **_xscale**:

```
this._xscale = -this._xscale;
```

Однако после зеркального отражения меняется расположение "точки привязки" объекта. Поэтому "точку привязки" при использовании зеркальных отражений с помощью свойств **_xscale** и **_yscale** лучше выбирать в середине объекта.

Кроме того, потребуется предусмотреть изменение знака значения свойства **_xscale** в самом начале, сразу после случайного выбора значения **dm**, чтобы "синхронизировать" изображение объекта с направлением движения мишени.

3. Запустим анимацию. Проверим ее работу. Убедимся, что теперь все работает правильно.

Итак, в анимации можно программировать "поведение" не одного, а нескольких объектов. При этом для каждого такого объекта пишется свой программный код. Этот код может быть назначен для одного и того же события, тогда при его совершении каждый объект запускает свой программный код.

События, связанные с нажатиями клавиш на клавиатуре, не указывают сами по себе, к какому именно объекту они относятся. Поэтому можно или реализовать управление с клавиатуры только для одного какого-то объекта, а остальные должны быть "неуправляемыми", либо нажатие некоторой клавиши будет восприниматься и приводить к соответствующим действиям синхронно во всех объектах, где запрограммировано отслеживание нажатия этой клавиши.

II. Задание 2.

В предыдущем случае снаряд можно было постоянно "корректировать" в полете нажатиями клавиш "Курсор влево" и "Курсор вправо". Но реальный снаряд (в отличие от управляемой ракеты) летит в направлении, заданном один раз перед выстрелом (изменение траектории под действием силы тяжести мы пока не рассматриваем).

Изменим анимацию так, чтобы она реализовывала физическую модель полета снаряда из пушки. Для этого в программном коде предыдущей анимации для объекта-снаряда:

• в условиях, связанных с проверкой нажатия клавиш "LEFT" и "RIGHT", добавим условие проверки, что $L = 0$ (т.е. "выстрел" еще не сделан), и только в этом случае будем менять значение угла α; причем будем ограничивать изменение этого угла интервалом от –90 до +90 градусов;

• чтобы можно было отследить текущий угол поворота "пушки", сделаем снаряд не круглым, а, например, в виде треугольника, и используем его свойство **_rotation** (поворот), чтобы отобразить, на какой угол он повернут; скорость движения снаряда увеличим (для чего при "выстреле" зададим $L = 20$ вместо 5).

1. Загрузим ранее созданную анимацию. Удалим из нее круглый объект — снаряд (программный код, "привязанный" к нему, можно заранее скопировать и вставить в текстовый редактор, чтобы не перенабирать его заново, а только модифицировать). Нарисуем в том же месте снаряд в виде треугольника вершиной вверх. Преобразуем его в символ Movie Clip. Так как созданный объект на данном слое расположен выше остальных, выделим объект "взрыв" и перенесем его в слое на самый верх.

2. Для объекта-снаряда напишем тот же программный код, который ранее использовался в предыдущей анимации. Запустим анимацию. Проверим, что ее работа не изменилась, поменялась только форма “снаряда”.

3. Модифицируем программный код, внося в него описанные выше изменения. Угол **alfa** удобнее теперь задавать в градусах, а потом пересчитывать в радианы для вычисления синуса/косинуса.

```
onClipEvent(load) {
  W = 550;
  H = 400;
  ws = 20;
  hs = 20;
  xs = W/2;
  ys = H - 25;
  this._x = xs;
  this._y = ys;
  wm = 200;
  hm = 100;
  xm = (W - wm) * Math.random();
  ym = (H - 2 * hm) * Math.random();
  _root.mishen._width = wm;
  _root.mishen._height = hm;
  _root.mishen._x = xm;
  _root.mishen._y = ym;
  _root.vzriv._visible = false;
  L = 0;
  alfa = 0;
  al = 0;
  dx = 0;
  dy = 0;
}
onClipEvent(enterFrame) {
if (Key.isDown(Key.SPACE)) {
  L = 20;
}
if (Key.isDown(Key.LEFT) && (L == 0)) {
  if (alfa > -90) {
    alfa -= 10; }
  al = alfa * Math.PI / 180;
  this._rotation = alfa;
}
if (Key.isDown(Key.RIGHT) && (L == 0)) {
  if (alfa < 90) {
    alfa += 10; }
  al = alfa * Math.PI / 180;
  this._rotation = alfa;
}
xs = this._x;
ys = this._y;
dx = Math.round(L*Math.sin(al));
dy = -Math.round(L*Math.cos(al));

xs += dx;
ys += dy;
if ((xs <= 0) || (xs >= (W - ws))) {
  L = 0; xs = W/2; ys = H - 25;
  al = 0; alfa = 0; this._rotation = 0;
}
if ((ys <= 0) || (ys >= (H - hs))) {
  L = 0; xs = W/2; ys = H - 25;
  al = 0; alfa = 0; this._rotation = 0;
}
```

Срабатывает при открытии анимации:
— ширина кадра,
— высота кадра,
— ширина снаряда,
— высота снаряда,
— середина ширины кадра,
— возле нижнего края кадра,
— снаряд выставляем по его координатам,
— ширина мишени,
— высота мишени,
— случайная коорд. x мишени,
— случайная коорд. y мишени,
— "образмериваем" мишень,
— выставляем мишень по полученным координатам,
— "взрыв" делаем невидимым,
— изначальное смещение снаряда,
— изначальный угол полета снаряда,
— рабочие переменные

Повторяющиеся действия в цикле:
— если нажат "ПРОБЕЛ" ("выстрел"), то **смещение равно 20 (скорость выше!),**

— если нажато "ВЛЕВО" **и выстрела еще не было, и угол больше -90°, то угол уменьшаем на 10°, пересчитываем в радианы, поворачиваем снаряд,**

— если нажато "ВПРАВО" **и выстрела еще не было, и угол меньше 90°, то угол увеличиваем на 10°, пересчитываем в радианы, поворачиваем снаряд,**

— считали текущие координаты снаряда,

— вычислили смещение снаряда по X и по Y (оно отрицательное, так как движение — вверх, против оси Y),
- вычисляем новые координаты снаряда,

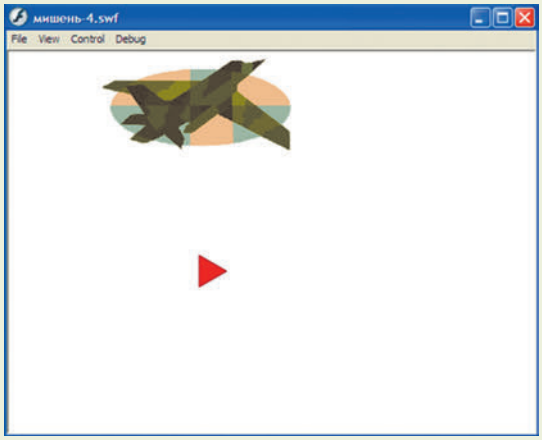
— если выход за левый/правый край, то снаряд на исходные позиции, включая угол его поворота,

— если выход за верхний/нижний край, снаряд на исходные позиции, включая угол его поворота,

```
this._x = xs;
this._y = ys;
if ( (xs >= xm) && ((xs + ws) <=
     (xm + wm)) && (ys >= ym) &&
     ((ys + hs) <= (ym + hm)) ) {
  L = 0; al = 0; alfa = 0;
  this._rotation = 0;
  _root.vzriv._x =
     (xs + ws / 2) -_root.vzriv._width / 2;
  _root.vzriv._y =
    (ys + hs / 2) - _root.vzriv._height / 2;
  _root.vzriv._visible = true;
}
}
```

```
— перемещаем снаряд в новое место.

— проверяем условие попадания
  в мишень, если оно достигнуто, то

— останавливаем снаряд, включая его
  поворот на 0 градусов,
— позиционируем изображение "взрыва",
  располагая его центр
  по центру снаряда,

— делаем "взрыв" видимым
```

4. Запустим анимацию, проверим ее работу.



Итак, мы научились управлять поворотом объекта. Правда, поворот объекта производится вокруг “точки привязки”, обозначенной крестиком при выделении объекта. А она размещена у левого верхнего угла. Поэтому поворот снаряда — несколько “неестественный”. Можно сместить точку привязки, чтобы сделать повороты более “естественными” (например, относительно центра объекта), но тогда нужно будет менять расчеты в условиях достижения краев и попадания в мишень (так как мы писали их из расчета, что точка регистрации расположена именно в верхнем левом углу). Поэтому мы не будем сейчас это делать.

УРОК 7. РАБОТА С МЫШЬЮ. “НЕСТАНДАРТНЫЙ” КУРСОР. “ПОПАДАНИЕ” МЫШЬЮ В ОБЪЕКТ

Специальные имена и элементы пути

this — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

_root (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

Свойства объекта (клипа, кнопки)

_x, _y — координаты объекта (в пикселях);

_width, _height — ширина и высота объекта;

_root._xmouse, _root._ymouse — ширина и высота объекта (в пикселях);

Mouse.hide() — спрятать “стандартный” для Windows курсор мыши;

Mouse.show() — показать “стандартный” для Windows курсор мыши;

_visible — видимость объекта (**true** — видимый, **false** — невидимый).

Методы

stop() — привязывается к кадру и останавливает воспроизведение анимации (клипа) на этом кадре;

play(*номер_кадра*) — запуск анимации (клипа) на воспроизведение, начиная с указанного номера кадра.

События

onClipEvent(enterFrame) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **enterFrame** — “запуск данного кадра на воспроизведение”.

Если в анимации содержится только один кадр, которому назначен данный обработчик, то при воспроизведении такой анимации этот кадр зацикливается, так что данный обработчик выполняется постоянно и записанный в нем программный код будет выполняться все время.

onClipEvent(load) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **load** — “загрузка анимации”. Оно выполняется однократно, поэтому его программный код может быть использован для задания исходных значений переменных;

onClipEvent (mouseDown) — обработчик событий, назначаемый клипу-курсору мыши. Данное событие является внешним и выполняется многократно, каждый раз при нажатии левой кнопки мыши;

onClipEvent (mouseUp) — обработчик событий, назначаемый клипу-курсору мыши. Данное событие является внешним и выполняется многократно, каждый раз при отпускании левой кнопки мыши;

on (press) { *программный код* } — обработчик событий, назначаемый для объекта-мишени.

Данное событие также является внешним и выполняется многократно, каждый раз при нажатии левой кнопки мыши на этом объекте;

`on (release) { программный код }` — обработчик событий, назначаемый для объекта-мишени. Данное событие также является внешним и выполняется многократно, каждый раз при отпускании левой кнопки мыши на этом объекте.

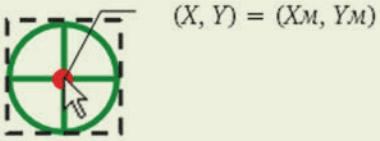
Замена курсора мыши на "свой". Чтобы создать такой эффект, нужно:

- отключить стандартный курсор Windows;
- обеспечить движение объекта, изображающего "новый" курсор мыши, синхронно с перемещением мыши.

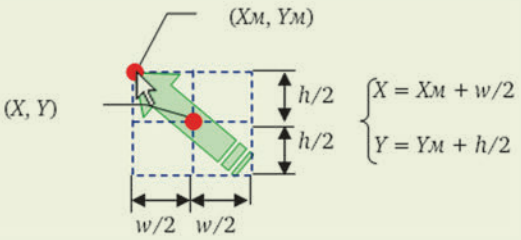
При этом нужно помнить, что точка привязки такого объекта должна совпадать с точкой указания курсора мыши:

- если объект — новый курсор представляет собой стрелку, то его точку привязки нужно расположить в верхнем левом углу;
- если объект — новый курсор представляет собой "прицел", то его точку привязки нужно расположить в середине.

В этом случае достаточно, чтобы координаты этого объекта совпадали с координатами мыши, например, для "прицела":



Если "точка указания" нового курсора не совпадает с его центром, то нужно выполнить пересчет координат. Например, если для курсора в виде стрелки, направленной влево вверх, точка указания расположена в середине, а не в верхнем левом углу, то пересчет координат выполняется так:



В качестве объекта — "нового" курсора можно выбрать любой Movie Clip — как статичный, так и представляющий собой анимацию.

Проверка попадания мышью в объект. В отличие от проверки попадания объекта-"снаряда" в объект-"мишень" здесь расчеты проще, так как точка указания (вершина курсора мыши) не имеет размеров (соответствует одному пикселю).

Для упрощения задачи проверки попадания мышью в объект используется специальная функция: **`_root.имя_объекта.hitTest(x, y, true)`**, где (x,y) — ранее считанные координаты мыши. Значение этой функции равно **`true`** при попадании координат (x,y) в объект либо **`false`** в противном случае.

При присваивании самому объекту-мишени события **`on(press)`** отдельное отслеживание попадания мышью в этот объект, очевидно, не требуется, так как само это событие срабатывает именно при нажатии мышью на этот объект.

Практические задания

Задание I. Работа с нестандартными курсорами мыши

1. Создадим новую анимацию. Нарисуем в ней новый объект — "прицел". Сгруппируем и смасштабируем картинку. Преобразуем картинку в Movie Clip. Разместим объект вне кадра.

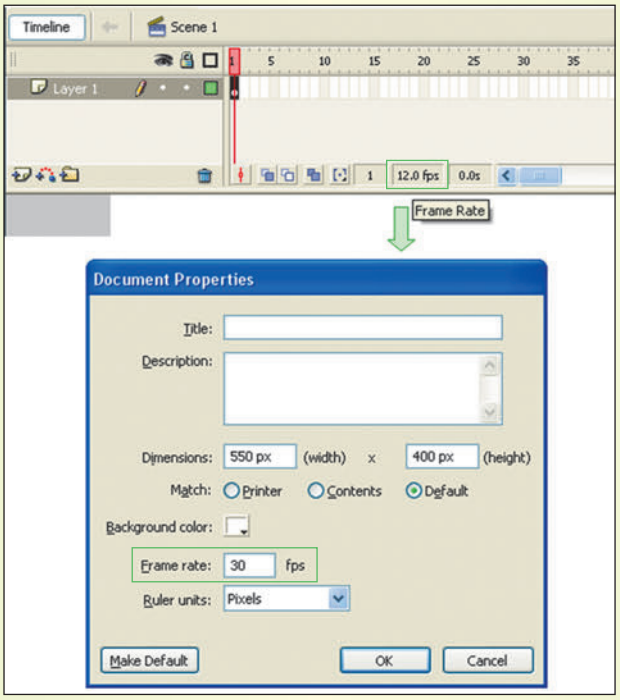
2. Привяжем к объекту следующий программный код:

```
onClipEvent(enterFrame) {
  this._x =_root._xmouse;
  this._y =_root._ymouse;
}
```

```
Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши
```

3. Запустим анимацию. Проверим: при перемещении курсора мыши в пределах кадра анимации созданный нами объект — прицел должен "следовать за курсором мыши как приклеенный".

Реально при быстрых движениях мыши можно заметить, что объект от него “отстает”. Чтобы этого не происходило, надо увеличить частоту кадров, вызвав диалоговое окно двойным щелчком мыши на обозначении частоты кадров под таймлайном.



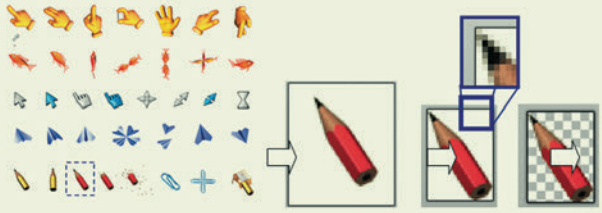
4. Добавим в программный код отключение видимости курсора мыши:

```
onClipEvent(load) {
  Mouse.hide();
}
onClipEvent(enterFrame) {
  this._x =_root._xmouse;
  this._y =_root._ymouse;
}
```

```
Первоначальные установки:
— спрятать "настоящий" курсор мыши,

Повторяющиеся действия в цикле:
— координаты объекта
  повторяют координаты курсора мыши
```

5. Запустим анимацию, проверим ее работу. Теперь мы полностью получаем эффект “нового вида” курсора мыши.

6. Создадим изображение “нового” курсора. В графическом редакторе Paint.Net (или аналогичном) вырежем подходящую стрелку из подборки курсоров (их можно найти в Интернете). Обрезку произведем точно по вершине стрелки. Сделаем окружающий фон прозрачным. Сохраним рисунок в формате PNG.



7. Создадим новую анимацию. Импортируем в нее (в библиотеку флеш-редактора) полученную картинку курсора (можно нарисовать такую картинку и непосредственно в Flash). Смасштабируем картинку. Преобразуем ее в Movie Clip. Разместим объект вне кадра.

8. Предположим, что у этого объекта точка привязки (крестик) расположена в центре, т.е. не совпадает с точкой указания — концом карандаша:

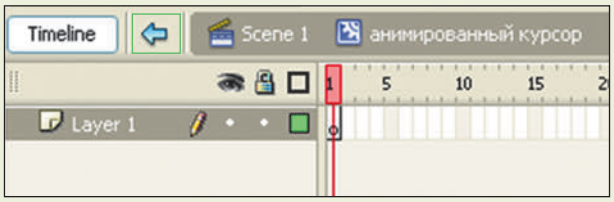
Привяжем объекту следующий программный код:

```
onClipEvent(load) {
  Mouse.hide();
  w = this._width;
  h = this._height;
}
onClipEvent(enterFrame) {
  this._x =_root._xmouse + w/2;
  this._y =_root._ymouse + h/2;
}
```

```
Первоначальные установки:
– спрятать "настоящий" курсор мыши,
– считать ширину объекта-курсора,
– считать высоту объекта-курсора,

Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши
  с учетом их пересчета
```

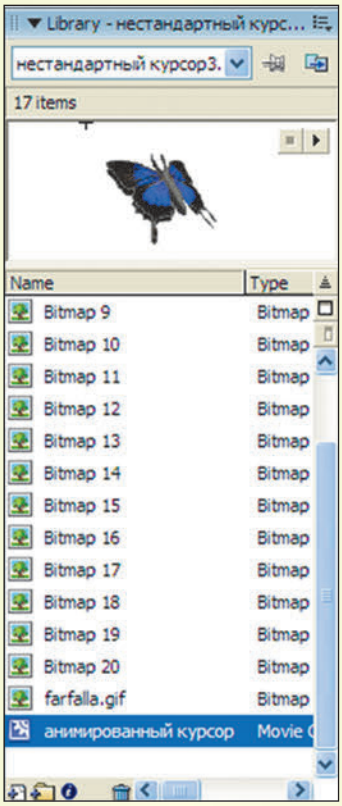
9. Запустим анимацию, проверим ее работу.

10. Создадим новую анимацию. В ней в меню выберем пункт **Insert > New Symbol** (**Вставить > Новый символ**), в появившемся окне выберем переключатель **Movie Clip**. В результате в библиотеке будет создан новый (пустой) клип Movie Clip, а флеш-редактор войдет в режим редактирования этого клипа.



11. Выбрав имеющийся первый ключевой кадр, перетащим на поле анимации файл формата “анимированный GIF”. При этом в клипе автоматически создается нужное количество ключевых кадров.

В библиотеку будет импортирован не только этот рисунок GIF, но и набор отдельных битмапов — это кадры GIF-анимации, стирать их из библиотеки нельзя!



12. Выйдем из режима редактирования клипа, щелкнув мышью на голубой стрелке в заголовке таймлайна. Из библиотеки перетащим созданный объект-клип в кадр, смасштабируем его и разместим вне кадра.

13. У этого объекта точка привязки (крестик) расположена в верхнем левом углу. Чтобы точка указания располагалась не в этом месте (“оторванно” от изображения), можно “сместить” ее, пересчитав координаты. Например, программный код может быть таким:

```
onClipEvent(load) {
  Mouse.hide();
  dx = 30;
  dy = 20;
}
onClipEvent(enterFrame) {
  this._x =_root._xmouse – dx/2;
  this._y =_root._ymouse – dy/2;
}
```

```
Первоначальные установки:
– спрятать "настоящий" курсор мыши,
– смещение координаты X (подбором),
– смещение координаты Y (подбором),

Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши
  с учетом их пересчета
```

14. Запустим анимацию.

Можно временно отключить команду скрытия настоящего курсора мыши, введя перед ней две косые черты: `// Mouse.hide();` и тем самым превратив эту команду в не исполняемый комментарий. Тогда можно проследить, где окажется точка указания курсора мыши, и выполнить подгонку смещений координат **dx** и **dy**. Завершив подгонку, можно убрать эти две косые черты и вновь включить в работу команду “отключения курсора”.

Задание II. Фиксация попадания мышью в объект

1. Создадим новую анимацию. Установим для нее частоту кадров 30 fps. Импортируем и разместим в ней любые картинки, изображающие мишени (4–5 штук). Масштабируем, разместим их в желаемых местах кадра. Каждую картинку превратим в символ Movie Clip. Дадим этим объектам соответствующие по их смыслу имена экземпляров.

2. Импортируем в библиотеку из библиотеки предыдущей анимации курсор в виде прицела. Переместим его на поле кадра. Привяжем к нему уже знакомый нам программный код:

```
onClipEvent(load) {
  Mouse.hide();
}
onClipEvent(enterFrame) {
  this._x =_root._xmouse;
  this._y =_root._ymouse;
}
```

```
Первоначальные установки:
– спрятать "настоящий" курсор мыши,

Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши
```

3. Запустим анимацию, проверим ее работу: мышью можно перемещать курсор-прицел, но пока еще щелчки мыши на объектах не дают никакой реакции.

4. Добавим реакцию на мишени. Для этого дополним программный код объекта-“прицела” поочередными проверками попадания щелчками мыши в каждый объект-мишень.

```
onClipEvent(load) {
  Mouse.hide();
  _root.kaban.visible = true;
  _root.ezj.visible = true;
  _root.suslik.visible = true;
  _root.homjak.visible = true;
  _root.olen.visible = true;
}
onClipEvent(enterFrame) {
  this._x =_root._xmouse;
  this._y =_root._ymouse;
}
onClipEvent (mouseDown) {
  x = _root._xmouse;
  y = _root._ymouse;
  if (_root.kaban.hitTest(x, y, true)) {
    _root.kaban._visible = false;
  }
  if (_root.ezj.hitTest(x, y, true)) {
    _root.ezj._visible = false;
  }
  if (_root.suslik.hitTest(x, y, true)) {
    _root.suslik._visible = false;
  }
  if (_root.homjak.hitTest(x, y, true)) {
    _root.homjak._visible = false;
  }
  if (_root.olen.hitTest(x, y, true)) {
    _root.olen._visible = false;
  }
}
```

```
Первоначальные установки:
– спрятать "настоящий" курсор мыши,
– показать мишень 1 ("кабан"),
– показать мишень 2 ("ёж"),
– показать мишень 3 ("суслик"),
– показать мишень 4 ("хомяк"),
– показать мишень 5 ("олень"),

Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши

Реакция на нажатие левой кнопки мыши:
– считываем текущие координаты
  "точки указания" (курсора мыши),
– проверка попадания в кабана,

– если да, то делаем мишень невидимой,

– проверка попадания в ежа,

– если да, то делаем мишень невидимой,

– проверка попадания в суслика,
– если да, то делаем мишень невидимой,

– проверка попадания в хомяка,
– если да, то делаем мишень невидимой,
– проверка попадания в оленя,
– если да, то делаем мишень невидимой,
```

5. Запустим анимацию на выполнение. Проверим: при щелчке мышью на каждой мишени она должна исчезать.

6. Данный способ написания программы неудобен тем, что с увеличением количества мишеней нужно все время наращивать длину программного кода (а с массивами объектов мы пока работать не умеем). Поэтому можно отдельно присваивать каждый программный код проверки попадания мышью в мишень самой мишени.

Программный код для объекта-прицела:

```
onClipEvent(load) {
  Mouse.hide();
  _root.kaban.visible = true;
  _root.ezj.visible = true;
  _root.suslik.visible = true;
  _root.homjak.visible = true;
  _root.olen.visible = true;
}
onClipEvent(enterFrame) {
this._x =_root._xmouse;
this._y =_root._ymouse;
}
```

```
Первоначальные установки:
— спрятать "настоящий" курсор мыши,
— показать мишень 1 ("кабан"),
— показать мишень 2 ("ёж"),
— показать мишень 3 ("суслик"),
— показать мишень 4 ("хомяк"),
— показать мишень 5 ("олень"),

Повторяющиеся действия в цикле:
— координаты объекта
  повторяют координаты курсора мыши
```

Программный код для каждой из мишеней:

```
on (press) {
  x = _root._xmouse;
  y = _root._ymouse;
  if (this.hitTest(x, y, true)) {
    this._visible = false;
  }
}
```

```
При щелчке мышью на данном объекте:
— считываем текущие координаты
  "точки указания" (курсора мыши),
— проверка попадания в данный объект,
— если да, то делаем мишень невидимой,
```

7. Запустим анимацию. Проверим: она должна работать точно так же, как и предыдущая (исчезновение мишени после щелчка мышью по ней).

8. “Стрельба” мышью по неподвижной мишени неинтересна. Добавим для каждой мишени ее случайное размещение в кадре, случайные направление и скорость движения и возможность “отскакивания” от краев экрана (все эти программные коды были рассмотрены ранее).

Программный код для каждой из мишеней (обратим внимание: в данном случае условия задания начальных координат и отражения от краев изменены, так как точка привязки располагается в середине объектов, а не в верхнем левом углу):

```
onClipEvent(load) {
  W = 550;
  H = 400;
  wm = this._width;
  hm = this._height;
  osx = Math.round(10 * Math.random() - 5);
  osy = Math.round(10 * Math.random() - 5);
  xm = Math.round((W-wm)*Math.random() + wm/2);
  ym = Math.round((H-hm)*Math.random() + hm/2);
  this._x = xm;
  this._y = ym;
}
onClipEvent(enterFrame) {
  if (xm - wm / 2 + osx <= 0 ||
      xm + wm / 2 + osx >= W) {
    osx = -osx;
  }
  if (ym - hm / 2 + osy <= 0 ||
      ym + hm / 2 + osy >= H) {
    osy = -osy;
  }
  xm += osx;
  ym += osy;
  this._x = xm;
  this._y = ym;
}
```

```
Первоначальные установки:
— ширина кадра анимации,
— высота кадра анимации,
— ширина данного объекта,
— высота данного объекта,
— смещения по x и y
  заданы случайно от -5 до 5,


— координаты мишени
  заданы случайно от края до края,


— позиционирование мишени в кадре,

Повторяющиеся действия в цикле:

— "отскок" слева или справа
  путем смены знака смещения,

— "отскок" сверху или снизу
  путем смены знака смещения,

— изменение координат
  объекта-мишени,
```

```
on (press) {
  x = _root._xmouse;
  y = _root._ymouse;
  if (this.hitTest(x, y, true)) {
    this._visible = false;
    osx = 0; osy = 0;
  }
}
```

```
— перемещение мишени по новым
  координатам,

При щелчке мышью на объекте:
— считываем текущие координаты
  "точки указания" (курсора мыши),
— проверка попадания в объект,
— да – делаем мишень невидимой
  и останавливаем ее
```

9. Запустим анимацию на воспроизведение. Проверим: объекты-мишени должны двигаться в случайно выбранных направлениях, со случайно выбранной скоростью и начиная со случайно выбранной позиции, а также “отскакивать” от краев кадра. А щелчок мышью на каждом объекте должен приводить к его исчезновению.

Особенность данной анимации: одно и то же событие (начальная загрузка анимации, открытие данного кадра, щелчок мышью) обрабатывается для каждого объекта по-разному, согласно заданному для данного объекта и данного события программному коду.

10. Добавим объектам-мишеням при “попадании” в них эффект “падающей мишени”. Для этого каждый статичный объект-MovieClip превратим в анимацию из нескольких кадров, в которой высота объекта меняется от исходной величины до нуля. При этом к первому кадру такого объекта-клипа привяжем скрипт, останавливающий воспроизведение этого клипа (чтобы изначально мишень не “падала”). При “попадании” будем запускать воспроизведение клипа объекта-мишени со второго его кадра (“мишень падает”). А для исключения “зацикливания” показа “падения” мишени последнему кадру ее клипа тоже привяжем скрипт останова воспроизведения.

1) выполним двойной щелчок мыши на прототипе объекта-мишени — т.е. на ее библиотечном символе MovieClip — он откроется для редактирования (в “шапке” панели Timeline появится синяя стрелка и шкала кадров именно этого клипа);

2) создадим в таймлайне этого клипа конечный ключевой кадр из расчета, что частота кадров у нас составляет 30 fps. Например, если мишень должна “падать” в течение секунды, то финальный ключевой кадр разместим в позиции 30;

3) перейдем в первый ключевой кадр клипа и создадим *анимацию Motion Tween*;

4) в финальном кадре клипа инструментом трансформации уменьшим высоту объекта. После этого надо так сместить уменьшенный по высоте объект, чтобы его низ соответствовал “ногам” зверя-мишени (так как уменьшение по высоте производится одновременно и снизу, и сверху). Удобно использовать “линейки” (Rules) и перетащенную до трансформации с горизонтальной линейки линию-направляющую, отмечающую низ мишени. Обратим внимание — точка привязки объекта, помеченная крестиком, при этом остается на прежнем месте и тем самым оказывается над объектом;

5) “пробегая” мышью по кадрам клипа, проверим — высота объекта должна плавно уменьшаться;

6) первому и последнему кадрам клипа присвоим скрипт из единственной команды `stop()`;

7) щелчком мыши на голубой стрелке в “шапке” Timeline вернемся в режим работы с анимацией в целом;

8) повторим шаги 1–7 для каждого объекта-мишени (для его библиотечного прототипа).

11. Изменим программный код, привязанный к объектам-мишеням и к событию “щелчок мышью на мишени” — заменим команду выключения видимости объекта командой, запускающей анимацию клипа (“падение мишени”) при попадании мышью:

Программный код для каждой из мишеней:

```
on (press) {
  x = _root._xmouse;
  y = _root._ymouse;
  if (this.hitTest(x, y, true)) {
    this.play(2);
    osx = 0; osy = 0;
  }
```

```
При щелчке мышью на объекте:
— считываем текущие координаты
  "точки указания" (курсора мыши),
— проверка попадания в объект,
— да — запускаем клип с его кадра 2
  и останавливаем движение объекта
```

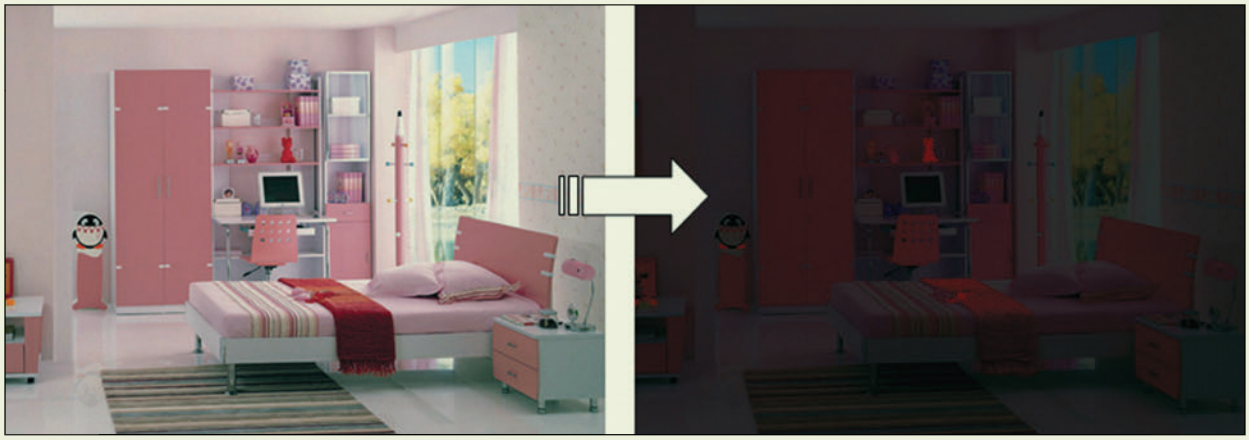
12. Запустим анимацию. Проверим ее работу. Теперь мишени при щелчке мышью на них должны “падать” и оставаться на месте “упавшими”.

Если мишени “падают” слишком медленно, то можно или переделать анимацию клипов, размещая финальный кадр анимации не на позиции 30, а раньше, либо увеличить частоту кадров для всей анимации (например, до 45 fps). Правда, в последнем случае увеличится и скорость движения мишеней (подумайте и самостоятельно ответьте на вопрос: почему?).

Задание III. Сочетание программирования работы с мышью и эффектов обычной анимации

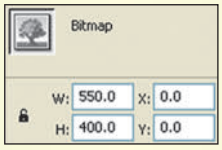
1. Создадим новую анимацию. В ней создадим три слоя (сверху вниз): “луч фонаря”, “светлая” и “темная”. Зададим частоту кадров анимации равной 30 fps.

2. В графическом редакторе обработаем выбранный рисунок комнаты (уменьшим его яркость, снизим контрастность). Полученный рисунок “затемненной” комнаты сохраним под другим именем.



3. Импортируем в редактор флеш рисунки комнаты — исходный и “затемненный”. Первый разместим в слое “светлая”, второй — в слое “темная”.

Для масштабирования и размещения картинки во весь кадр достаточно в панели свойств картинки задать значения ее координат *x*,*y* равными нулю, а значения ширины и высоты — равными ширине и высоте кадра (550 × 400).



4. В слое “луч фонаря” нарисуем черный кружок желаемого размера. Преобразуем его в символ Movie Clip. Разместим полученный объект в верхнем левом углу.

5. Привяжем этому объекту программный код, реализующий замену стандартного курсора мыши на свой собственный (т.е. на данный объект):

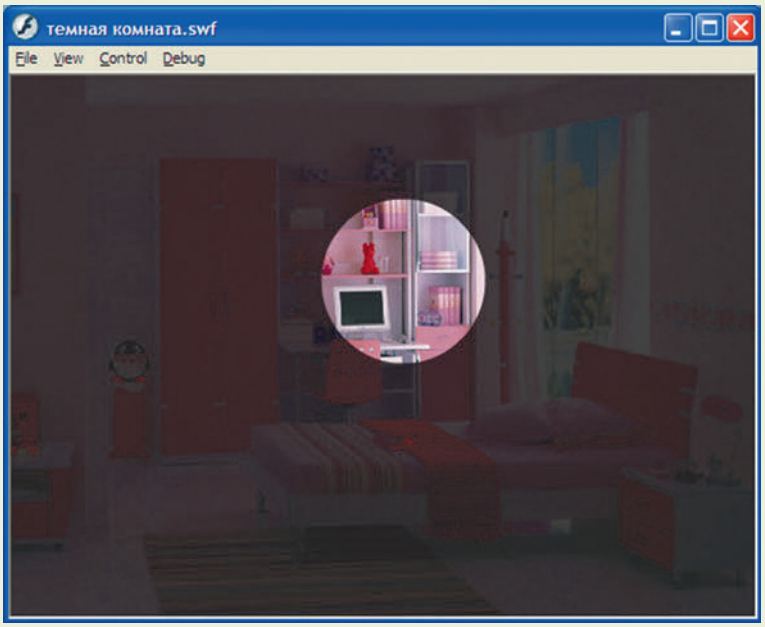
```
onClipEvent(load) {
  Mouse.hide();
}
onClipEvent(enterFrame) {
this._x =_root._xmouse;
this._y =_root._ymouse;
}
```

```
Первоначальные установки:
– спрятать "настоящий" курсор мыши,

Повторяющиеся действия в цикле:
– координаты объекта
  повторяют координаты курсора мыши
```

6. Запустив анимацию на выполнение, проверим: черный кружок должен перемещаться по кадру при движении мыши.

7. Вернувшись к редактированию анимации, преобразуем слой “луч фонаря” в *слой-маску*.

8. Запустив анимацию на выполнение, убедимся в том, что теперь при помощи мыши мы можем перемещать маску, “высвечивая” на затемненной комнате те или иные участки по своему желанию, словно лучом фонарика.



Таким образом, мы можем сочетать программное управление поведением тех или иных объектов с применением различных эффектов, реализуемых обычными средствами Flash.

УРОК 8. ПЕРЕТАСКИВАНИЕ ОБЪЕКТОВ МЫШЬЮ

Специальные имена и элементы пути

this — указатель на сам текущий объект, используется при вызове свойства или метода текущего объекта;

_root (“корень”) — начало всей иерархии объектов, используется при записи абсолютного пути.

Свойства объекта (клипа, кнопки)

_x, _y — координаты объекта (в пикселях);
_width, _height — ширина и высота объекта;
_root._xmouse, _root._ymouse — координаты курсора мыши;
_visible — видимость объекта (**true** — видимый, **false** — невидимый).

События

onClipEvent(enterFrame) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **enterFrame** — “запуск данного кадра на воспроизведение”.

Если в анимации содержится только один кадр, которому назначен данный обработчик, то при воспроизведении такой анимации этот кадр зацикливается, так что данный обработчик выполняется постоянно и записанный в нем программный код будет выполняться все время.

onClipEvent(load) { *программный код* } — обработчик событий, назначаемый клипу. Здесь используется событие **load** — “загрузка анимации”. Оно выполняется однократно, поэтому его программный код может быть использован для задания исходных значений переменных.

on (press) { *программный код* } — обработчик событий, назначаемый для объекта-мишени. Данное событие также является внешним и выполняется многократно, каждый раз при нажатии левой кнопки мыши на этом объекте.

on (release) { *программный код* } — обработчик событий, назначаемый для объекта-мишени. Данное событие также является внешним и выполняется многократно, каждый раз при отпускании левой кнопки мыши на этом объекте.

Перетаскивание объекта мышью. Мы уже умеем перемещать объект синхронно с курсором мыши, чтобы создать “собственный курсор”. Фактически это и есть перетаскивание мышью. Нужно лишь реализовать теперь возможность “включения” перетаскивания объекта и “выключение” перетаскивания (“сброс”) объекта на новое место. Для этого используем переменную-флаг — признак перетаскивания:

1) изначально для объекта значение переменной-флага равно **false** (перетаскивание не производится);

2) при нажатии кнопки мыши на объекте его переменная-флаг меняет значение с **false** на **true** — перетаскивание включено;

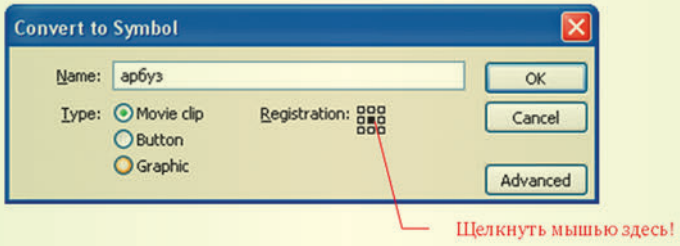
3) при отпускании кнопки мыши значение переменной-флага снова меняется с **true** на **false** — перетаскивание выключено.

Практические задания

Задание I. Перетаскивание одного объекта

1. Создадим новую анимацию. Разместим на ней произвольную картинку (импортированную или нарисованную). Преобразуем ее в объект Movie Clip. Дадим объекту имя экземпляра **obj1**.

Нам удобнее, чтобы при перетаскивании курсор мыши соответствовал середине объекта. Поэтому "точку привязки" объекта мы размещаем в середине, а не в верхнем левом углу.



2. Привяжем к данному объекту следующий программный код для события **load**:

```
onClipEvent(load) {
  drag = false;
}
```

```
Начальные присваивания:
– перетаскивание выключено
```

3. Дополним программный код объекта реакцией на нажатие и отпускание кнопки мыши:

```
on (press) {
  drag = true;
}
on (release) {
  drag = false;
}
```

```
Нажатие кнопки мыши на объекте:
– перетаскивание включено

Отпускание кнопки мыши на объекте:
– перетаскивание выключено
```

4. Дополним программный код для объекта изменением его координат синхронно с координатами мыши (собственно перетаскивание). Этот код соответствует событию **enterFrame**:

```
onClipEvent(enterFrame) {
  if (drag) {
    this._x =_root._xmouse;
    this._y =_root._ymouse;
  }
}
```

```
Повторяющиеся действия в цикле:
– если перетаскивание включено,
  то менять координаты этого объекта
  синхронно с координатами мыши
```

5. Запустим анимацию на выполнение. Проверим возможность перетаскивания объекта мышью при удержании левой кнопки мыши нажатой, когда курсор мыши наведен на объект.

6. Неприятным является "скачок" объекта при нажатии кнопки мыши для его перетаскивания. Это вызвано тем, что курсор мыши не обязательно находится в середине объекта, а в момент включения перетаскивания к курсору мыши "привязывается" именно средняя точка. Чтобы избежать этого, используем еще две переменные — **deltax** и **deltay** для сохранения отступов между серединой объекта и местом нажатия кнопки мыши на объекте, чтобы при изменении координат объекта учитывать эти отступы.

Изменим программный код:

```
onClipEvent(load) {
  drag = false;
  deltax = 0;
  deltay = 0;
}
on (press) {
  deltax = this._x - _root._xmouse;
  deltay = this._y - _root._ymouse;
  drag = true;
}
on (release) {
  drag = false;
}
onClipEvent(enterFrame) {
  if (drag) {
    this._x =_root._xmouse + deltax;
    this._y =_root._ymouse + deltay;
  }
}
```

Начальные присваивания:
– перетаскивание выключено
– **определение переменных – отступов**

Нажатие кнопки мыши на объекте:
– **сначала запоминаем отступы от середины объекта до курсора,**
– **потом включаем перетаскивание**

Отпускание кнопки мыши на объекте:
– перетаскивание выключено

Повторяющиеся действия в цикле:
– если перетаскивание включено, то менять координаты этого объекта синхронно с координатами мыши **с учетом отступов**

7. Запустим анимацию на выполнение. Проверим, как выполняется перетаскивание объекта “за край”.

Для более “гладкого” перетаскивания увеличим частоту кадров анимации до 30 fps.

Задание II. Выборочное перетаскивание нескольких объектов

1. Создадим в уже имеющейся анимации еще несколько объектов Movie Clip. Дадим им имена экземпляров **obj2**, **obj3** и т.д. Масштабируем и разместим эти объекты произвольно на кадре анимации.

Чтобы белый фон не мешал при перетаскивании одного объекта на другой, нужно заранее (перед импортом картинок в флеш-редактор) в графическом редакторе убрать на этих картинках фон (сделать его прозрачным) и сохранить картинки в формате png.

2. ***Каждому*** из этих объектов привяжем программный код, который ранее был привязан объекту **obj1**.

Рекомендуется (хотя и не обязательно) каждый объект создавать в отдельном слое — тогда можно будет управлять взаимным наложением объектов друг поверх друга, меняя порядок слоев.

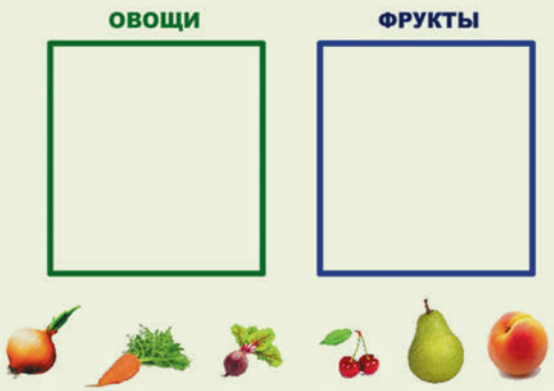
3. Запустим анимацию. Проверим перетаскивание каждого объекта по очереди. Обратим внимание: если два объекта наложены друг на друга, то перетаскивается тот, который в точке наведения курсора мыши оказывается в более верхнем слое.

Задание III. Сортировка объектов

Реализуем тестовое задание, в котором требуется отсортировать объекты — перетащить тот или иной объект в правильную группу.

1. Создадим новую анимацию. Разместим на ней несколько объектов небольшого размера. Каждый объект преобразуем в символ Movie Clip. Дадим им имена экземпляров **obj1**, **obj2** и т.д. Разместим эти объекты в нижней части кадра. Каждому объекту привяжем уже знакомый нам программный код, реализующий перетаскивание. Запустим анимацию, проверим перетаскивание каждого объекта.

2. Нарисуем на кадре анимации в отдельном слое (самом нижнем!) два прямоугольника, в которые нужно будет перетаскивать объекты. Добавим к ним соответствующие текстовые заголовки — названия групп. Например:



Каждый прямоугольник (без его названия!) преобразуем в символы Movie Clip, устанавливая для них точку привязки в левом верхнем углу. Дадим им имена экземпляров по смыслу (например, в нашем случае левый прямоугольник имеет имя **ovo**, а правый — **fru**).

3. Создадим кнопку (символ типа Button) с надписью “Проверить”. Разместим ее внизу кадра.

4. В отдельных слоях (самых верхних!) создадим два “баннера” (каждый в своем слое) с надписями: “ПРАВИЛЬНО!” и “НЕПРАВИЛЬНО!”. Временно разместим их где-либо вне кадра. Преобразуем их в символы Movie Clip. Дадим им имена экземпляров **verno** и **neverno** соответственно. Каждому из них привяжем простой программный код:

```
onClipEvent(load) {
  this._visible = false;
}
```

```
Начальные присваивания:
— спрятать данный объект
```

Позже, после завершения создания анимации, эти “баннеры” нужно разместить на кадре где-либо в середине (поверх остальных объектов).

5. Объекту-кнопке привяжем следующий программный код. Его “типовые” фрагменты проверяют попадание того или иного объекта в “правильную” область.

```
on (press) {
  otvet = 0;
  xm1 = _root.ovo._x;
  ym1 = _root.ovo._y;
  wm1 = _root.ovo._width;
  hm1 = _root.ovo._height;
  xm2 = _root.fru._x;
  ym2 = _root.fru._y;
  wm2 = _root.fru._width;
  hm2 = _root.fru._height;
// первый объект
  xs1 = _root.obj1._x;
  ys1 = _root.obj1._y;
  ws1 = _root.obj1._width;
  hs1 = _root.obj1._height;
if ( (xs1 >= xm1) && (xs1 <= (xm1 + wm1))
&& (ys1 >= ym1) && (ys1 <= (ym1 + hm1)) )
{
  otvet += 1;
}
// второй объект
  xs2 = _root.obj2._x;
  ys2 = _root.obj2._y;
  ws2 = _root.obj2._width;
  hs2 = _root.obj2._height;

if ( (xs2 >= xm2) && (xs2 <= (xm2 + wm2))
&& (ys2 >= ym2) && (ys2 <= (ym2 + hm2)) )
{
  otvet += 1;
}
// третий объект
  xs3 = _root.obj3._x;
  ys3 = _root.obj3._y;
  ws3 = _root.obj3._width;
  hs3 = _root.obj3._height;

if ( (xs3 >= xm2) && (xs3 <= (xm2 + wm2))
&& (ys3 >= ym2) && (ys3 <= (ym2 + hm2)) )
{
  otvet += 1;
}
// четвертый объект
  xs4 = _root.obj4._x;
  ys4 = _root.obj4._y;
  ws4 = _root.obj4._width;
  hs4 = _root.obj4._height;

if ( (xs4 >= xm2) && (xs4 <= (xm2 + wm2))
&& (ys4 >= ym2) && (ys4 <= (ym2 + hm2)) )
{
  otvet += 1;
}
```

```
Нажатие кнопки мыши на объекте:
— счетчик правильных ответов
— считываем координаты первой области

— считываем размеры первой области

— считываем координаты второй области

— считываем размеры второй области

— обработка первого объекта:
  — считываем координаты
    и размеры
    первого
    объекта
  — проверка попадания средней точки
    1 объекта в первую группу (ovo)
  — если да, то увеличиваем количество
    правильных ответов
— обработка второго объекта:
  — считываем координаты
    и размеры
    второго
    объекта

  — проверка попадания средней точки
    2 объекта во вторую группу (fru)
  — если да, то увеличиваем количество
    правильных ответов

— обработка третьего объекта:
  — считываем координаты
    и размеры
    третьего
    объекта


  — проверка попадания средней точки
    3 объекта во вторую группу (fru)
  — если да, то увеличиваем количество
    правильных ответов

— обработка четвертого объекта:
  — считываем координаты
    и размеры
    четвертого
    объекта
 — проверка попадания средней точки
    4 объекта во вторую группу (fru)
  — если да, то увеличиваем количество
    правильных ответов
```

```
// пятый объект
  xs5 = _root.obj5._x;
  ys5 = _root.obj5._y;
  ws5 = _root.obj5._width;
  hs5 = _root.obj5._height;
if ( (xs5 >= xm1) && (xs5 <= (xm1 + wm1))
&& (ys5 >= ym1) && (ys5 <= (ym1 + hm1)) )
{
  otvet += 1;
}

// шестой объект
  xs6 = _root.obj6._x;
  ys6 = _root.obj6._y;
  ws6 = _root.obj6._width;
  hs6 = _root.obj6._height;

if ( (xs6 >= xm1) && (xs6 <= (xm1 + wm1))
&& (ys6 >= ym1) && (ys6 <= (ym1 + hm1)) )
{
  otvet += 1;
}

// общие выводы
if (otvet == 6) {

  _root.verno._visible = true;
}
  else {
  _root.neverno._visible = true;
}
}
```

```
— обработка пятого объекта:
  — считываем координаты
    и размеры
    пятого
    объекта
  — проверка попадания средней точки
    5 объекта в первую группу (ovo)
  — если да, то увеличиваем количество
    правильных ответов


— обработка шестого объекта:
  — считываем координаты
    и размеры
    шестого
    объекта

  — проверка попадания средней точки
    6 объекта в первую группу (ovo)
  — если да, то увеличиваем количество
    правильных ответов


— реакция на ответы:
  — если все 6 объектов
    перетащены правильно, то
  — показать "баннер" "ПРАВИЛЬНО!"

  — иначе -
  — показать "баннер" "НЕПРАВИЛЬНО!"
— конец программного кода
```

Обратим внимание: кодирование правильных и неправильных ответов (т.е. факта попадания каждого объекта в “правильную” или “неправильную” группу/область) здесь производится вручную — в каждом операторе проверки попадания мы сами записываем координаты и размеры или первой области, или второй — в зависимости от того, какая область для данного объекта “правильная”. При этом для упрощения записи мы проверяем только попадание в область средней точки объекта (т.е. объект может, например, краем чуть выходить за границу области, а не обязательно умещаться целиком в ней).

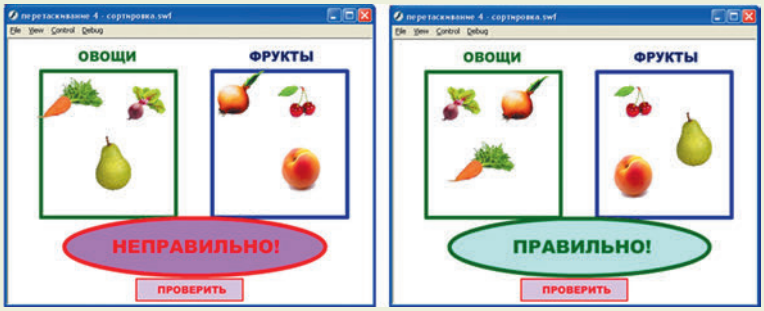
Кроме того, данный программный код не оптимален — много повторов одинаковых команд для тех или иных объектов. Сделать программу более короткой можно:

а) используя подпрограммы;

б) применяя массивы и циклы.

Эти возможности мы пока не рассматриваем.

6. Запустим анимацию на выполнение. Проверим ее работу. При нажатии кнопки “ПРОВЕРИТЬ”, если не все объекты правильно размещены в своих областях (овощи — слева, фрукты — справа), то должен выдаваться “баннер” “НЕПРАВИЛЬНО!”. Иначе (при правильном размещении всех объектов) должен выдаваться “баннер” “ПРАВИЛЬНО!”.



Продолжение читайте в следующем номере.

Педагогический университет
«Первое сентября»

Лицензия Департамента образования г. Москвы 77Л01 № 0007183, рег. № 036377 от 23.07.2015

ДИСТАНЦИОННЫЕ КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

(с учетом требований ФГОС)

До 15 января 2016 г. ведется прием заявок на второй поток 2015/16 учебного года

образовательные программы:

- НОРМАТИВНЫЙ СРОК ОСВОЕНИЯ – 108 УЧЕБНЫХ ЧАСОВ

  Стоимость – 4990 руб.

- НОРМАТИВНЫЙ СРОК ОСВОЕНИЯ – 72 УЧЕБНЫХ ЧАСА

  Стоимость – от 3990 руб.

По окончании выдается удостоверение о повышении квалификации установленного образца

Перечень курсов и подробности – на сайте edu.1september.ru

Пожалуйста, обратите внимание:

заявки на обучение подаются только из Личного кабинета, который можно открыть на любом сайте портала www.1september.ru

Задание ЕГЭ на использование вспомогательной функции (задание 21)

Д.М. Златопольский,
Москва

Начиная с 2012 года в демонстрационных вариантах ЕГЭ по информатике и ИКТ приводится задание по программированию на использование вспомогательных функций. В данной статье описывается методика выполнения нескольких вариантов таких заданий.

Задание 21 из контрольно-измерительных материалов ЕГЭ 2015 года (досрочный период) [1]

Условие

Напишите в ответе число, равное количеству различных входных значений, при которых приведенная ниже программа выводит тот же ответ, что и при входном значении $k = 10$. Значение $k = 10$ также включается в подсчет различных значений k. Для вашего удобства программа приведена на пяти языках программирования.

Бейсик

```
DIM K, I AS LONG
INPUT K
I = 1
WHILE F(I) < K
  I = I + 1
WEND
IF F(I) - K <= K - F(I - 1)
THEN
  PRINT I
ELSE
  PRINT I - 1
END IF

FUNCTION F(N)
BEGIN
  F = N * N * N
END
```

Алгоритмический

```
алг
нач цел k, i
  ввод k
  i := 1
  нц пока F(i) < k
    i := i + 1
  кц
  если F(i) – k <= k – F(i – 1)
    то
      вывод i
    иначе
      вывод i – 1
  все
кон
алг цел F(цел n)
нач
  знач := n * n * n
кон
```

Си

```
#include<stdio.h>
  long f(long n) {
    return n * n * n;
  }
  void main()
  {
    long k, i;
    scanf("%ld", &k);
    i = 1;
    while (f(i) < k)
      i++;
    if (F(i) – k <= k – F(i – 1)) {
      printf("%ld", i);
    } else {
      printf("%ld", i – 1);
    }
  }
```

Python

```
def f(n):
  return n * n * n
i = 1
k = int(input())
while f(i) < k:
  i += 1
if f(i) – k <= k – f(i – 1):
  print(i)
else:
  print(i – 1)
```

Паскаль

```
var i, k: longint;
function F(n : longint): longint;
begin
  F := n * n * n
end;
begin
  readln(k);
  i := 1;
  while F(i) < k do
    i := i + 1;
  if F(i) – k <= k – F(i – 1)
    then writeln(i)
    else writeln(i – 1);
end.
```

Анализ программ

Вспомогательная функция F вычисляет куб аргумента.

Оператор цикла с предусловием работает до нахождения минимального значения i аргумента функции F, при котором значение $F = i^3$ больше либо равно числу k. Для заданного значения $k = 10$ таким значением i является 3 ($3^3 = 27$).

После этого происходит сравнение двух разностей — превышения i^3 над k и превышения k над $(i – 1)^3$:

```
F(i) – k <= k – F(i – 1)
```

В заданном случае первая разность ($27 – 10 = 17$) больше, то есть сработает "ветвь" **иначе** (**else**) условного оператора и в результате будет выведено число 2 ($i – 1$).

Для нахождения других значений k, для которых также будет выведен ответ 2, перебор, например, с помощью таблицы со значениями двух указанных разностей:

	k	9	10	11	12	13	14	15	16	17	18	19	20
$F(i) = 27$	**$27 – k$**	18	17	16	15	14	13	12	11	10	9	8	7
$F(i – 1) = 8$	**$k – 8$**	1	2	3	4	5	6	7	8	9	10	11	12
	$27 – k \le k – 8$	нет	нет	нет	нет	нет	нет	нет	нет	нет	да	да	

— хотя и позволяет получить количество подходящих значений k (9), является трудоемким.

Вместо перебора значений в таблице можно (что даже быстрее позволяет получить результат) решить неравенство:

$27 – k > k – 8$ (оно противоположно приведенному в таблице!)

$$35 > 2k$$

$$k < 17{,}5$$

То есть подходят целые значения 9, 10, …, 17.

Кроме того, значение, равное 2, будет выведено, когда условие в программе окажется истинным. Это может произойти при k, меньшем или равном 8, но большем 1 (при $k = 1$ имеем $i = 1$, $F(i) = 1$, $F(i – 1) = 0$, $F(i) – k = 0$, $k – F(i – 1) = 1$, $F(i) – k < k – F(i – 1)$, то есть будет выведено $i = 1$). Проверка этих значений (см.

таблицу ниже) показывает, что для требуемого результата подходят числа 5, 6, 7 и 8.

	k	2	3	4	5	6	7	8
$F(i) = 8$	**8 – k**	6	5	4	3	2	1	0
$F(i-1) = 1$	**k – 1**	1	2	3	4	5	6	7
	8 – k <= k – 1	нет	нет	нет	да	да	да	да

Здесь также удобно решить неравенство:

$$8 - k \le k - 1$$
$$9 \le 2k$$
$$k \ge 4{,}5$$

Подходят значения k: 5, 6, 7, 8.

Общий диапазон подходящих по условию значений — 5..17 (количество значений 17 – 5 + 1 = 13).

Итак, ответ: 13.

Решим также аналогичную задачу для случаев, когда заданное значение k дает другой результат (отличающийся от 2).

Пусть $k = 22$.

Анализ работы программы показывает, что:

— работа оператора цикла с предусловием закончится при i, равном 3 ($F(i) = 3^3 = 27$); при этом $F(i-1) = 2^3 = 8$;

— $F(i) - k = 27 - 22 = 5$;

— $k - F(i-1) = 22 - 8 = 14$;

— $F(i) - k \le k - F(i-1)$, то есть на экран будет выведено значение i, равное 3.

Для поиска других значений k, дающих этот же результат, следует решить два неравенства.

Первое из них связано с истинностью условия в программе:

$$3^3 - k \le k - 2^3.$$

Его решение дает следующие значения k: 18, 19, … . Конечное значение этого отрезка — 27 (при $k = 28$ оператор цикла с предусловием закончится при $i = 4$).

Второе неравенство должно учитывать, что i должно быть равно четырем и условие, противоположное приведенному в программе:

$$4^3 - k > k - (4-1)^3.$$

Здесь $k \le 45$.

Итак, диапазон искомых значений: 18..45, то есть их количество — 28.

Можно сформулировать общие указания по выполнению заданий указанного типа.

1. Определить i, при котором оператор цикла прекратит работу.

2. Найти число, выводимое на экран.

3.1. Если два найденных числа отличаются (на 1), то решить[1] два неравенства:

1) $i^3 - k > k - (i-1)^3$, которое по сути противоположно приведенному в программе.

Найденное решение $k < \ldots$ для целого k будет правой границей диапазона искомых значений k;

2) $(i-1)^3 - k \le k - (i-2)^3$.

Его решение $k \ge \ldots$ будет левой границей диапазона искомых целых значений k.

3.2. Если два найденных числа совпадают, то решить два неравенства:

1) $i^3 - k \le k - (i-1)^3$.

Решение $k > \ldots$ даст левую границу искомого диапазона значений;

2) $(i+1)^3 - k > k - i^3$.

Здесь решение $k < \ldots$ дает правую границу.

4. Рассчитать искомый в задании результат по формуле:

правая граница – левая граница + 1,

где *правая граница* и *левая граница* — значения, найденные при решении неравенств.

Задание 21 из демонстрационного варианта ЕГЭ 2015 года [2]

Условие

Напишите в ответе число различных значений входной переменной k, при которых программа выдает тот же ответ, что и при входном значении $k = 64$. Значение $k = 64$ также включается в подсчет различных значений k. Для вашего удобства программа приведена на пяти языках программирования.

Бейсик

```
DIM K, I AS LONG
INPUT K
I = 12
WHILE I > 0 AND F(I) >= K
  I = I - 1
WEND
PRINT I
FUNCTION F(N)
BEGIN
  F = N * N
END
```

Алгоритмический

```
алг
нач цел k, i
  ввод k
  i := 12
  нц пока i > 0 и F(i) >= k
    i := i - 1
  кц
  вывод i
кон
алг цел F(цел n)
нач
  знач := n * n
кон
```

Си

```
#include<stdio.h>
int f(int n) {
  return n * n;
}
```

[1] Имеется в виду решение неравенств не в общем виде, а для найденного значения i и заданного значения k, как это делалось в рассмотренных примерах.

```
void main()
{
  int k, i;
  scanf("%d", &k);
  i = 12;
  while (i > 0 && f(i) >= k)
    i--;
  printf("%d", &i);
}
```

Python

```
def f(n):
  return n * n
k = int(input())
i = 12
while i > 0 and f(i) >= k:
  i = i - 1
print(i)
```

Паскаль

```
var i, k: longint;
function F(n : longint): longint;
begin
  F := n * n
end;
begin
  readln(k);
  i := 12;
  while i > 0 и F(i) >= k do
    i := i - 1;
  write(i)
end.
```

Решение

Анализ показывает, что в программе находится (с помощью оператора цикла с предусловием) и выводится на экран максимальное значение переменной i, квадрат которого меньше заданного значения k. Для $k = 64$ таким значением является 7. Число 64 является максимальным среди всех чисел, так сказать, “с 7” (при $k = 65$ на экран будет выведено 8). А какое число минимальное? Ответ — 50 ($7^2 + 1$). Значит, искомое количество различных значений k, при которых на экран выводится 7, равно $64 - 50 + 1 = 15$.

Если провести аналогичные расчеты для других значений, выводимых на экран (например, от 1 до 6 и 8), то можно получить табл. 1.

Из нее следует, что границы диапазонов значений k, соответствующих некоторому выводимому числу i, связаны с квадратом i и $(i + 1)$:

$$i^2 + 1..(i + 1)^2.$$

Интересно, что если условие в операторе цикла, связанное со значением функции, будет со строгим неравенством:

```
нц пока i > 0 и F(i) > k
  i := i - 1
кц
```

— то в таблице значения “сместятся” на 1 (см. табл. 2).

При этом количество значений в каждом диапазоне останется тем же.

Рассмотрим также вариант задания, в котором в программе происходит увеличение значения i.

Условие

Напишите в ответе число различных значений входной переменной k, при которых приведенная ниже программа выдает тот же ответ, что и при входном значении $k = 25$. Значение $k = 25$ также включается в подсчет различных значений k.

```
алг
нач цел k, i
  ввод k
  i := 1
  нц пока i < 100 и F(i) <= k
    i := i + 1
  кц
  вывод i
кон
алг цел F(цел n)
нач
  знач := n * n
кон
```

Решение

Здесь в программе находится и выводится на экран минимальное значение переменной i, квадрат которого больше заданного значения k. Для $k = 25$ таким значением является 6. Это же зна-

Таблица 1

k	2	3	4	5	…	9	10	…	16	17	…	25	26	…	36	37	…	49	50	…	64	65	…	81
i	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8

Таблица 2

k	1	2	3	4	…	8	9	…	15		16	…	24	25	…	35	36	…	48	49	…	63	64	…	80
i	1	1	1	2	2	2	3	3	3		4	4	4	5	5	5	6	6	6	7	7	7	8	8	8

Таблица 3

k	1	2	3	4	…	8	9	…	15	16	…	24	25	…	35	36	…	48	49	…	63	64	…	80
i	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9	9	9

Таблица 4

k	2	3	4	5	…	9	10	…	16	17	…	25	26	…	36	37	…	49	50	…	64	65	…	81
i	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9	9	9

чение будет выведено и для ряда бóльших чисел. Максимальное из них — 35 (при $k = 36$ на экран будет выведено 7). Значит, искомое количество равно $35 - 25 + 1 = 11$.

Табл. 3 будет иметь вид, аналогичный табл. 1, а табл. 2 (соответствующая строгому неравенству в условии) — табл. 4.

Рассмотрим также задания, связанные с использованием вспомогательной функции, приведенные в демонстрационных вариантах ЕГЭ предыдущих лет (аналогичные задания представлены в книгах от разработчиков ЕГЭ — см., например, [3–4]).

Задача A14 демонстрационного варианта ЕГЭ 2012 года [5]

Условие

Определите, какое число будет напечатано в результате работы следующей программы (для вашего удобства программа представлена на четырех языках):

Бейсик

```
MODULE A14
SUB Main()
  DIM d, a, b, t, M, R AS DOUBLE
  a = -3 : d = 0.1
  d = 0.1
  t = a: M = a: R = F(a)
  WHILE t < b
    IF F(t) < R THEN
      M = t
      R = F(t)
    END IF
    t = t + d
  END WHILE
  Console.Write(M)
END SUB
FUNCTION F(ByVal x AS DOUBLE) AS DOUBLE
  RETURN (x - 1) * (x - 3)
END FUNCTION
END MODULE
```

Си

```
#include <stdio.h>
double F(double x)
{
  return (x - 1) * (x - 3);
}
void main()
{
  double d, a, b, t, M, R;
  a = -3; b = 3;
  d = 0.1;
  t = a; M = a; R = F(a);
  while (t < b) {
    if ( F(t) < R ) {
      M = t; R = F(t);
    }
    t = t + d;
  }
printf("%f", M);
}
```

Паскаль

```
var d, a, b, t, M, R : real;
function F(x: real): real;
begin
  F := (x - 1) * (x - 3)
end;
begin
  a := -3; b := 3;
  d := 0.1;
  t := a; M := a; R := F(a);
  while t < b do
  begin
    if F(t) < R then
      begin
        M := t; R := F(t)
      end;
   t := t + d
  end;
  write(M)
end.
```

Алгоритмический язык

```
алг
нач вещ d, a, b, t, M, R
a := -3; b := 3
d := 0.1
t := a; M := a; R := F(a)
нц пока t < b
  если F(t) < R
    то
     M := t; R := F(t)
  все
  t := t + d
кц
вывод M
кон
алг вещ F (вещ x)
нач
  знач := (x - 1) * (x - 3)
вещ
```

1) –1 3) –3
2) 2 4) 24

(Приведены варианты ответа. — **Прим. авт.**)

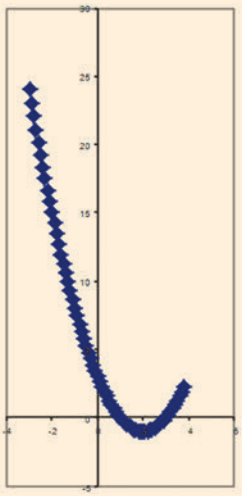
Решение

Полная трассировка программы в данном случае крайне трудоемка — тело оператора цикла будет выполняться 60 раз (начальное значение t равно –3, и оно увеличивается до 2,9 с шагом 0,1; величина b не меняется).

Итак, условие $t < b$ будет истинным, пока $t \le 2{,}9$. Но означает ли это, что окончательное значение величины M также будет равно 2,9? Для ответа на этот вопрос следует знать, сколько раз будет меняться значение M в “теле” условного оператора, или, по-другому, при каком значении t прекратит выполняться это “тело”, то есть когда станет ложным условие $F(t) < R$.

Проанализируем это условие. Из программы видно, что значение R тоже равно значению функции, то есть тоже зависит от t, но при каждом очередном выполнении тела цикла (на каждой итерации) — от предыдущего значения t. Значит, нужно

определить, когда значение функции $F(t)$ станет больше, чем на предыдущем шаге[2]. Для этого необходимо исследовать на экстремум заданную функцию, которую можно представить в виде $x^2 - 4x + 3 = 0$. Подробное исследование мы проводить не будем (11-классники должны уметь сделать это самостоятельно), а приведем только график этой функции.



Анализ графика показывает, что последнее значение t, при котором $F(t) < R$, равно 2,0. Это и будет окончательным значением величины M.

Ответ: 2.

Задача B14 демонстрационного варианта 2013 года [6]

Определите, какое число будет напечатано в результате выполнения следующего алгоритма (для вашего удобства алгоритм представлен на четырех языках).

Бейсик

```
DIM A, B, T, M, R AS INTEGER
A = -20: B = 20
M = A: R = F(A)
FOR T = A TO B
  IF F(T) < R THEN
  M = T
  R = F(T)
ENDIF
NEXT T
PRINT M
FUNCTION F(x)
  F = 3 * (x - 8) * (x - 8)
END FUNCTION
```

Си

```
#include<stdio.h>
int F(int x)
{
  return 3 * (x - 8) * (x - 8);
}
void main()
{
  int a, b, t, M, R;
  a = -20; b = 20;
  M = a; R = F(a);
  for (t = a; t <= b; t++){
    if (F(t) <R ) {
      M = t; R = F(t);
    }
  }
printf("%d", M);
}
```

Паскаль

```
var a,b,t,M,R :integer;
function F(x:integer):integer;
  begin
    F := 3 * (x - 8) * (x - 8)
  end;
begin
  a := -20; b := 20;
  M := a; R := F(a);
  for t := a to b do begin
    if (F(t) < R) then begin
      M := t;
      R := F(t)
    end
  end;
  write(M);
end.
```

Алгоритмический

```
алг
нач цел a, b, t, R, M
a := -20; b := 20
M := a; R := F(a)
нц для t от a до b
  если F(t) < R
    то
      M := t; R := F(t)
  все
кц
вывод M
кон
алг цел F(цел x)
нач
  знач := 3 * (x - 8) * (x - 8)
кон
```

Решение

"Внешнее" отличие данной задачи (кроме другой вспомогательной функции и других переменных величин) от предыдущей в том, что в ней используется оператор цикла с параметром. Однако сути это не меняет, так как этот оператор является аналогом оператора цикла с предусловием (на школьном алгоритмическом языке):

```
t := a
нц для t <= b
  если F(t) < R
    то
      M := t; R := F(t)
  все
  t := t + 1
кц
```

Анализ зависимостей величин M и R от параметра цикла t показывает, что и здесь следует определить, когда значение функции $F(t)$ станет больше, чем на предыдущем шаге (опять же, если быть точным, — значение аргумента, при котором это происходит).

Исследовав приведенную в программе функцию, которую удобно представить в виде $F = 3(x - 8)^2$,

[2] Иными словами, нужно найти минимум функции $F(t)$, точнее — значение аргумента, при котором этот минимум достигается.

можно установить, что минимум функции достигается при $x = 8$ (в основной части программы — при $t = 8$).

Ответ: 8.

Задача B14 демонстрационного варианта 2014 года [7]

Условие

Напишите в ответе число, которое будет напечатано в результате выполнения следующего алгоритма (для вашего удобства алгоритм представлен на четырех языках).

Бейсик

```
DIM A, B, T, M, R AS INTEGER
A = -11: B = 11
M = A: R = F(A)
FOR T = A TO B
  IF F(T) < R THEN
    M = T
    R = F(T)
  ENDIF
NEXT T
PRINT M + 6
FUNCTION F(x)
  F = 2 * (x * x - 16) *
        (x * x - 16) + 5
END FUNCTION
```

Си

```
#include<stdio.h>
int F(int x)
{
  return 2 * (x * x - 16) *
         (x * x - 16) + 5;
}
void main()
{
  int a, b, t, M, R;
  a = -11; b = 11;
  M = a; R = F(a);
  for (t = a; t <= b; t++){
    if (F(t) < R) {
      M = t; R = F(t);
    }
  }
printf("%d", M + 6);
}
```

Паскаль

```
var a, b, t, M, R: integer;
Function F(x: integer) :integer;
  begin
    F := 2 * (x * x - 16) *
         (x * x - 16) + 5
  end;
begin
  a := -11; b := 11;
  M := a; R := F(a);
  for t := a to b do begin
    if (F(t) < R) then begin
      M := t;
      R := F(t)
    end
  end;
  write(M + 6)
end.
```

Алгоритмический

```
алг
нач цел a, b, t, R, M
  a := -11; b := 11
  M := a; R := F(a)
  нц для t от a до b
    если F(t) < R
      то
        M := t; R := F(t)
    все
  кц
  вывод M + 6
кон
алг цел F(цел x)
нач
  знач := 2 * (x * x - 16) *
          (x * x - 16) + 5
кон
```

Задача решается аналогично. Обратим внимание на то, что выводится значение, на шесть большее найденного в результате исследования функции.

Задания для самостоятельной работы учащихся

1. Выполните задание 21 из контрольно-измерительных материалов ЕГЭ 2015 года (досрочный период), условие которого приведено в начале статьи, для случая, когда входное значение k равно 50.

2. Напишите в ответе число различных значений входной переменной k, при которых программа выдает тот же ответ, что и при входном значении $k = 25$. Значение $k = 25$ также включается в подсчет различных значений k.

Литература

1. КИМ ЕГЭ 2015 (досрочный период). *http://www.fipi.ru/sites/default/files/document/2015/05.pdf*.
2. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена по информатике и ИКТ 2015 года. *http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory*.
3. *Лещинер В.Р.* ЕГЭ: Информатика: 2015. Типовые тестовые задания (рекомендовано ИСМО Российской академии образования для подготовки выпускников всех типов образовательных учреждений РФ к сдаче экзаменов в форме ЕГЭ). М.: Изд-во “Экзамен”, 2015.
4. ЕГЭ-2014: Информатика: самое полное издание типовых вариантов заданий. / авт.-сост. *Д.М. Ушаков, А.П. Якушкин.* М.: АСТ: Астрель, 2014 (Федеральный институт педагогических измерений).
5. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена по информатике и ИКТ 2012 года. *http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory*.
6. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена по информатике и ИКТ 2013 года. *http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory*.
7. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена по информатике и ИКТ 2014 года. *http://fipi.ru/ege-i-gve-11/demoversii-specifikacii-kodifikatory*.

ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ г. МОСКВЫ
ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»
МОСКОВСКИЙ ПЕДАГОГИЧЕСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГЕНЕРАЛЬНЫЙ СПОНСОР: ИЗДАТЕЛЬСТВО «ДРОФА»

МОСКОВСКИЙ ПЕДАГОГИЧЕСКИЙ МАРАФОН УЧЕБНЫХ ПРЕДМЕТОВ 2016

21 МАРТА – 15 АПРЕЛЯ

РАСПИСАНИЕ ДНЕЙ ПЕДАГОГИЧЕСКОГО МАРАФОНА

21 марта	День учителя технологии *
22 марта	**Открытие Марафона** День классного руководителя
23 марта	День школьного психолога День учителя ОБЖ
24 марта	День здоровья детей, коррекционной педагогики, логопеда, инклюзивного образования и лечебной физической культуры
25 марта	День учителя начальной школы (день первый)
26 марта	День учителя начальной школы (день второй)
27 марта	День дошкольного образования
29 марта	День учителя географии
30 марта	День учителя химии
31 марта	День учителя биологии
1 апреля	День учителя информатики
2 апреля	День учителя физики
3 апреля	День учителя математики
5 апреля	День учителя истории и обществознания
6 апреля	День учителя МХК, музыки и ИЗО
7 апреля	День школьного и детского библиотекаря
8 апреля	День учителя литературы
9 апреля	День учителя русского языка
10 апреля	День учителя английского языка
12 апреля	День учителя французского языка
13 апреля	День школьной администрации
14 апреля	День учителя физической культуры
15 апреля	День учителя немецкого языка **Закрытие**

marathon.1september.ru

Обязательная предварительная регистрация на все дни Марафона с 22 февраля 2016 года на сайте marathon.1september.ru

Каждый участник Марафона, посетивший три мероприятия одного дня, получает официальный именной сертификат (6 часов)

В дни Марафона ведущие издательства страны представляют книги для учителей

Начало работы каждого дня – 9.00. Завершение работы – 15.00

УЧАСТИЕ БЕСПЛАТНОЕ. ВХОД ПО БИЛЕТАМ

РЕГИСТРИРУЙТЕСЬ, РАСПЕЧАТЫВАЙТЕ СВОЙ БИЛЕТ И ПРИХОДИТЕ!

Место проведения Марафона: МПГУ, ул. Малая Пироговская, дом 1, стр. 1 (в 5 минутах ходьбы от ст. метро «Фрунзенская»)
* Место проведения Дня учителя технологии: ЦО № 293, ул. Касаткина, 1а (ст. метро «ВДНХ»)

По всем вопросам обращайтесь, пожалуйста, по телефону **8-499-249-3138** или по электронной почте marathon@1september.ru

ШКОЛА ПРОГРАММИРОВАНИЯ

Продолжаем считать “счастливые” билеты

Д.М. Златопольский,
Москва

В статье [1] был приведен ряд вариантов программы решения задачи подсчета так называемых “счастливых” билетов (автобусных или других билетов, в шестизначных номерах которых сумма трех первых цифр равна сумме трех последних). В результате был получен вариант программы, решающий задачу за время, примерно в четыре раза меньшее, чем первоначальный вариант. Оказывается, имеется возможность решить задачу за еще более короткое время. Методика, реализующая такую возможность, описывается в данной статье.

Шестизначный номер билета состоит из двух частей: левой (три цифры) и правой (тоже три цифры), причем в обеих частях номера “счастливого” билета сумма цифр одинакова. В каждой части сумма цифр может быть от 0 до 27. Обозначим эту сумму — *сум*. Некоторое значение *сум* могут давать в общем случае несколько трехзначных чисел. Например, сумму, равную 2, — числа 002[1], 011, 020, 101, 110, 200 (всего их шесть). Значит, общее число “счастливых” билетов с такой суммой равно 6 × 6 = 36. Поэтому можем сказать, что общее количество шестизначных “счастливых” билетов будет равно

$$КЧ_0 + КЧ_1 + КЧ_2 + \ldots + КЧ_{26} + КЧ_{27} = \\ = \sum_{сум=0}^{27} КЧ_{сум} * КЧ_{сум}$$

где $КЧ_{сум}$ — количество трехзначных чисел с суммой *сум*.

Но как найти значения $КЧ_{сум}$?

Начнем с более простого случая — определим $КЧ_{сум}$ для двузначных чисел. Для них значение *сум* может быть равно 0, 1, 2, …, 18. Обсудим, какой может быть последняя цифра двузначного числа при известной сумме цифр. Для этого составим и исследуем таблицу:

Таблица 1

Сумма двух цифр (*сум*)	Возможные значения последней цифры
0	0
1	0 1
2	0 1 2
…	
9	0 1 2 … 0
10	1 2 … 9
11	2 3 … 9
…	
17	8 9
18	9

Из табл. 1 можно установить минимальное и максимальное значения последней цифры по следующему правилу:

если *сум* > 9
то
мин_посл = *сум* – 9
макс_посл = 9
иначе
мин_посл = 0
макс_посл = *сум*
все

Общее число возможных значений последней цифры равно *макс_пят* – *мин_пят* + 1. А так как каждому из возможных значений соответствует только одна оставшаяся цифра, то общее число двузначных чисел, сумма которых составляет *сум*, и будет равно значению последнего выражения.

Вернемся к основной задаче — определению количества *трехзначных* чисел с суммой цифр *сум*. Здесь также можно установить[2] правило определе-

[1] Примем, что номера могут иметь начальные нули и что возможен номер билета 000000.

[2] Предлагаем читателям убедиться в справедливости приведенного далее правила самостоятельно (см. также [1]).

ния минимального и максимального значений последней цифры в зависимости от *сум*:

если *сум* > 9
то
макс_посл = 9
иначе
макс_посл = *сум*
все
если *сум* > 18
то
мин_посл = *сум* – 18
иначе
мин_посл = 0
все

Определив все возможные значения последней цифры, можно для каждой из них найти соответствующее число вариантов двухзначных чисел и просуммировать все найденные числа. То есть мы пришли к аналогичной задаче, но для двухзначных чисел. Это означает, что можно применить прием, который в программировании называется “рекурсией” [2–3].

Рекурсивная, то есть использующая рекурсию, функция КЧ, возвращающая количество двух- или трехзначных чисел, сумма цифр которых составляет *сум*, может быть оформлена следующим образом (на школьном алгоритмическом языке):

```
алг цел КЧ(арг цел k, сум)
  |k - количество разрядов в числе (2 или 3)
  |сум – сумма цифр числа
нач цел мин_посл, макс_посл, посл, всего
  |посл – последняя цифра числа
  |всего – меняющееся количество
  |искомых чисел
если k = 2
  то
    |Определяем границы значений
    |последней цифры
    если сум > 9
      то
        мин_посл := сум - 9
        макс_посл := 9
      иначе
        мин_посл := 0
        макс_посл := сум
    все
    |Сразу можем определить значение функции
    знач := макс_посл - мин_посл + 1
  иначе |k = 3
    |Определяем границы значений
    |последней цифры
    если сум > 9
      то
        макс_посл := 9
      иначе
        макс_посл := сум
    все
    если сум > 18
      то
        мин_посл := сум - 18
      иначе
        мин_посл := 0
    все
    всего := 0
    |Для всех значений последней цифры
    нц для посл от мин_посл до макс_посл
      |рекурсивно вызываем эту же
      |функцию КЧ, но для k = 2 и для
      |оставшейся суммы (сум – посл),
      |и прибавляем значение функции
      |к "старому" значению величины всего
      всего := всего + КЧ(2, сум - посл)
    кц
    |Значение функции для 3-значного числа
    знач := всего
все
кон
```

С использованием этой функции основная часть программы решения подсчета количества “счастливых” шестизначных билетов оформляется очень кратко:

```
алг Число_шестизначных_счастливых_билетов
нач цел кол, сумма_цифр
  |кол – искомое количество
  кол := 0
  |Для всех возможных значений суммы цифр
  |вызываем функцию КЧ и, используя
  |формулу, приведенную в начале статьи,
  |определяем новое значение искомой
  |величины кол
  нц для сумма_цифр от 0 до 27
    кол := кол + КЧ(3,сумма_цифр) *
           КЧ(3,сумма_цифр)
  кц
  вывод нс, кол
кон
```

Использованный подход к решению задачи (который, как показывают расчеты, приводит к значительному уменьшению времени работы программы) может быть использован для решения задачи в общем виде — определения количества *k*-значных “счастливых” номеров билетов. Для общего случая рекурсивная функция КЧ должна быть немного изменена. Изменения следующие:

1) вместо случая *k* = 3 должен быть рассмотрен случай *k* > 2;

2) минимально возможные значения последней цифры должны рассчитываться по правилу:

если *сум* > (*k* – 1) * 9
то
мин_посл = *сум* – (*k* – 1) * 9
иначе
мин_посл := 0
все

3) при рекурсивном вызове должно использоваться не значение 2 (см. выше), а *k* – 1.

При этих изменениях завершающая часть функции примет вид:

```
...
если k = 2
    то
      ...
    иначе |k > 2
      |Определяем границы значений
      |последней цифры
      если сум > 9
        то
          макс_посл := 9
        иначе
          макс_посл := сум
      все
      если сум > 18
        то
          мин_посл := сум - 18
        иначе
          мин_посл := 0
      все
      всего := 0
      нц для посл от мин_посл до макс_посл
        |Рекурсивно вызываем эту же
        |функцию, но для k, уменьшенного
        |на 1, и для оставшейся суммы
        |(сум - посл)
        всего := всего + КЧ(k - 1,
                 сум - посл)
      кц
      |Значение функции
      знач := всего
    все
  кон
```

В основной части программы целесообразно использовать дополнительные переменные величины:

— *разрядность_номера* — число разрядов в номере билета (4, 6, 6 или 8);

— *разр* — число разрядов в половине номера.

Вся основная часть программы имеет вид:

```
алг Число_счастливых_билетов
нач цел разрядность_номера, разр, кол,
сумма_цифр
|Вводим разрядность номера и рассчитываем
значение разр
вывод нс, "Задайте число разрядов
          в билете (4, 6, 8 или 10) "
ввод разрядность_номера
разр := div(разрядность_номера, 2)
|Определяем сумму квадратов значений
|функции КЧ
кол := 0
нц для сумма_цифр от 0 до 9 * разр
  кол := кол + КЧ(разр, сумма_цифр) *
         КЧ(разр, сумма_цифр)
кц
|Выводим ответ
вывод нс, "Количество билетов с ",
          разрядность_номера,
          "-значными 'счастливыми'
          номерами равно ", кол
кон
```

Задания для самостоятельной работы

1. Разработав программу на языке программирования, которым вы владеете, определите количество “счастливых” четырех-, шести-, восьми- и десятизначных номеров.

2. Описанный в статье подход может быть реализован без использования рекурсии — расчеты значений $КЧ_{сум}$ можно проводить от меньшего числа разрядов к большему. Посмотрите на табл. 2. В ней приведены значения $КЧ_{сум}$ для различных значений суммы цифр и числа разрядов. Например, выделенное число 73 означает, что именно столько существует трехзначных чисел с суммой цифр 12.

Для однозначных чисел все значения известны (они равны 1). Для остальных каждое число в таблице получается, если просуммировать 10 элементов из столбца слева, стоящих в одной строке с ним и сверху от него (оттенены значения, сумме которых равно число 73). Правда, есть исключение — для чисел, соответствующих сумме цифр, меньшей 9, значения определяются суммированием меньшего количества значений.

Таблица 2

Значение функции $КЧ_{сум}$

Сумма цифр	Количество разрядов в половине номера			
	1	**2**	**3**	**4**
0	1	1	1	1
1	1	2	3	4
2	1	3	6	10
3	1	4	10	20
4	1	5	15	35
5	1	6	21	56
6	1	7	28	84
7	1	8	36	120
8	1	9	45	165
9	1	10	55	220
10		9	63	282
11		8	69	348
12		7	**73**	415
13		6	75	480
14		5	75	540
15		4	73	592
16		3	69	633
17		2	63	660
18		1	55	670
19			45	660
20			...	...

После заполнения таблицы для нахождения количества “счастливых” билетов нужно просуммировать квадраты чисел, стоящих в том или ином

столбце. Если нужно было бы подсчитать количество 10-значных “счастливых” номеров, то потребовалось бы заполнить еще один столбец до значения суммы, равного 45.

Разработайте соответствующий вариант программы.

3. Сравните время выполнения описанного в данной статье варианта программы, варианта программы согласно заданию 2 и вариантов, приведенных в [1]. Для этого используйте процедуры и функции, возвращающие текущее время, предусмотренные в системе программирования. Так как на продолжительность работы программы, кроме ее особенностей, влияют и другие факторы, для каждого варианта проведите серию расчетов и используйте минимальное значение в серии.

Литература

1. *Златопольский Д.М., Мирончик Е.А.* Еще раз о “счастливых” билетах. / “В мир информатики” № 212 (“Информатика” № 11/2015).

2. Рекурсия — эффектно, но не всегда эффективно. / “В мир информатики” № 188–189 (“Информатика” № 7–8/2014).

3. *Златопольский Д.М.* Программирование: типовые задачи, алгоритмы, методы. М.: Бином. Лаборатория знаний, 2007.

MICROSOFT EXCEL УГЛУБЛЕННО

Считаем “счастливые” билеты в среде электронных таблиц

В статье “Продолжаем считать «счастливые» билеты” в этом выпуске описан метод подсчета числа так называемых “счастливых” билетов, основанный на использовании значения количества двухзначных, трехзначных и т.д. чисел с некоторой суммой цифр *сум*. В табл. 2 статьи приведены значения этого количества для различных значений суммы цифр и числа разрядов (например, сумму цифр, равную 2, могут иметь шесть трехзначных чисел: 002[3], 011, 020, 101, 110, 200).

Этот метод может быть использован для решения задачи с помощью электронных таблиц (программы Microsoft Excel и аналогичных). Верхняя часть соответствующего листа показана на *рис*. 1.

В столбце A записываются все возможные значения суммы цифр (в нашем случае, в котором мы рассматриваем пятизначные числа, сумма цифр может составлять от 0 до 45). Эти значения удобно получить, используя автозаполнение ячеек.

Зачем нужны пустые строки? Дело в том, что все числа в столбцах C, D, E и F получаются путем суммирования 10 чисел из соседнего с данным столбца слева, стоящих в одной строке с рассчитываемым и сверху от него (например, желтым цветом выделены значения, сумме которых равно число 73). Правда, есть исключение — для чисел, соответствующих сумме цифр, меньшей 9, значения определяются суммированием меньшего количества значений — вверх только до строки с нулевой суммой цифр (см. пример для числа 10 в столбце E). Поэтому, чтобы использовать общую формулу для расчета, для таких чисел приходится создавать пустые ячейки с нулевыми значениями.

Для однозначных чисел (столбец B) все значения для суммы от 0 до 9 равны 1. А остальные значения

	A	B	C	D	E	F
1	**ЧИСЛО «СЧАСТЛИВЫХ» БИЛЕТОВ**					
2		**Количество разрядов в половине номера**				
3		**1**	**2**	**3**	**4**	**5**
4						
5						
6						
7						
8						
9						
10						
11						
12						
13	**Сумма цифр**					
14	0	1	1	1	1	1
15	1	1	2	3	4	5
16	2	1	3	6	**10**	15
17	3	1	4	10	20	35
18	4	1	5	15	35	70
19	5	1	6	21	56	126
20	6	1	7	28	84	210
21	7	1	8	36	120	330
22	8	1	9	45	165	495
23	9	1	10	55	220	715
24	10		9	63	282	996
25	11		8	69	348	1340
26	12		7	**73**	415	1745
27	13		6	75	480	2205
…						

Рис. 1

можно рассчитать, как только что говорилось, используя общую формулу. Для ячейки C19 она такая:

=СУММ(B10:B19)

[3] Принимается, что номера могут иметь начальные нули и что возможен также номер билета 000000.

Эта формула может быть скопирована во все необходимые ячейки столбцов C, D, E и F.

После расчетов для нахождения количества “счастливых” билетов нужно просуммировать квадраты чисел, стоящих в том или ином столбце. Удобно применить функцию СУММКВ, которая как раз и возвращает сумму квадратов значений из ячеек или диапазонов, указанных в качестве ее аргументов. Например, для шестизначных номеров “счастливых” билетов следует использовать данные столбца D, то есть формула для нахождения следующая:

=СУММКВ(D14:D41)

Задания для самостоятельной работы

1. Подготовьте вариант листа, в котором дополнительных пустых строк не будет. При этом все значения в столбцах C, D, E и F должны рассчитываться по формуле, которая вводится только в одну ячейку, а затем копируется.

Указание по выполнению. Используйте в формуле так называемую “комбинированную”, или “смешанную”, ссылку (адрес).

2. Определите количество “счастливых” четырех-, шести-, восьми- и десятизначных номеров билетов.

3. Интересно, что программа Microsoft Excel и подобная позволяет исследовать и более “значные” номера. Добавив на листе нужные столбцы с данными, установите, до какой “значности” номеров можно решать задачу, и найдите соответствующие количества.

4. Получите график зависимости доли (в процентах) “счастливых” билетов среди общего числа билетов с номером различной длины.

5. Подготовьте лист для нахождения числа “счастливых” билетов, длина номера которых задается в некоторой ячейке (см. *рис.* 2).

	A	B
1	Задайте количество цифр в номере билета	
2	Число таких “счастливых” билетов	
…		

Рис. 2

Указания по выполнению

1. Рассчитайте искомые количества для всех значений длины номера.

2. Для вывода в ячейке B2 нужного количества используйте:

— половину значения, задаваемого в ячейке B1;

— “вложенную” функцию ЕСЛИ:

=ЕСЛИ(значение1; АдресОтвета1; ЕСЛИ(значение2; АдресОтвета1);…)

Можно также использовать функцию ВЫБОР. Ее общий вид:

=ВЫБОР(номер;значение1;значение2;...)

Она возвращает значение1, если аргумент номер равен 1, значение2, если аргумент номер равен 2, и т.д.

Результаты присылайте в редакцию (можно выполнять не все задания).

ИСТОРИЯ ИНФОРМАТИКИ

Что такое “ванневар”?

В.В. Шилов,
Москва

В изданном на английском языке известном словаре хакерского жаргона можно найти словечко *vannevar*, которое составители словаря трактуют как “несбывшийся прогноз в области развития технологий или ошибочную инженерную концепцию, в первую очередь связанные с предположением о линейном и равномерном характере развития технологий”.

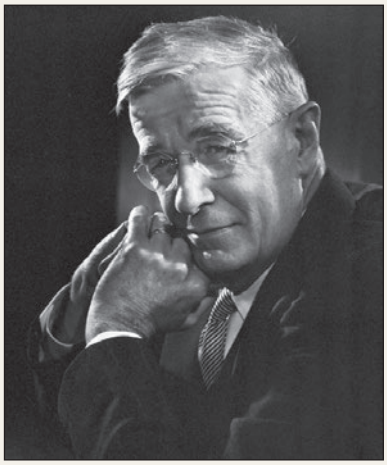
В словаре поясняется, что термин происходит от имени американского ученого и инженера Ванневара Буша (1890–1971), и что он появился и получил распространение после того, как однажды Буш сказал, что если человеку и удастся построить “искусственный мозг”, то он будет величиной с нью-йоркский небоскреб, а для его охлаждения понадобится поток воды с Ниагарский водопад. Иными словами, получается, что Буш выразил сомнение в перспективах создания малогабаритных компьютеров и искусственного интеллекта. При этом в словаре даже уточняется, что прогноз был сделан в то время, когда компьютеры уже были даже не ламповыми, а транзисторными, — то есть где-то в конце 1950-х годов! — и по этой причине оценивают его как еще более неудачный.

Что же, история науки, в частности история вычислительной техники, знает многочисленные примеры неудачных прогнозов путей ее будущего развития. Вспомним, например, высказывание английского математика и специалиста по страховому делу Фрэнсиса Уэйлса, сделанное в 1936 г. на научном семинаре при обсуждении возможностей двоичной системы счисления: “Не могу представить себе, чтобы кому-нибудь потребовалось выполнять умножение со скоростью 40 000 или даже 4000 операций в час”. Гениальный ученый Джон фон Нейман (1903–1957), узнав в 1954 году о начале работ по созданию языка программирования высокого уровня (который впоследствии стал языком ФОРТРАН), высказал сомнение в необходимости более мощного инструмента, нежели язык команд машины. Кеннет Олсен, основатель корпорации DEC, производившей знаменитые мини-компьютеры PDP, на самом пороге эры персональных компьютеров уверенно заявлял, что ему неизвестны причины, по которым кто бы то ни было захотел иметь дома компьютер… Аналогичные примеры можно при-

водить и приводить, однако имена авторов этих, действительно неудачных, высказываний нарицательными не стали.

Давайте попробуем разобраться! Для начала следует сказать, что Ванневар Буш был великим человеком — сомневаться в этом не приходится. Список его свершений не просто удивляет, а, скорее, потрясает. Он был крупным математиком: так, создатель кибернетики Норберт Винер (1894–1964) писал, что именно его друг Буш направил его мысли в сторону изучения систем с обратной связью. Буш воспитал несколько поколений специалистов — например, среди его учеников был создатель теории информации Клод Шеннон (1916–2001). Именно Буш дал ему тему магистерской диссертации, прославившей имя Шеннона.

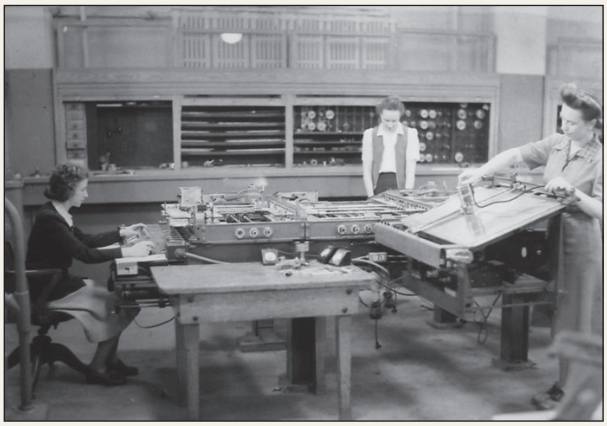
Буш внес огромный вклад в развитие вычислительной техники. В 1920–1940-х гг. он сконструировал и построил несколько механических и электрических аналоговых вычислительных машин — сетевой анализатор, интеграф, дифференциальный анализатор.



Ванневар Буш

Механический дифференциальный анализатор позволял решать линейные дифференциальные уравнения до шестого порядка. В его комплектацию входили шесть специальных узлов-интеграторов. Система трансмиссий позволяла соединять выход любого устройства с входом любого другого в зависимости от решаемой задачи. Дифференциальным анализатором управляли вручную один или несколько операторов. Они передвигали на входных чертежных столах специальный курсор (индекс) по вычерченному заранее с высокой точностью графику исходной функции (точнее, оператор должен был удерживать индекс на заданной кривой, в то время как специальный механизм перемещал индекс параллельно оси абсцисс). График, соответствующий результирующей функции, немедленно автоматически вычерчивался на выходном столе.

В конце 1941 г. под руководством Буша был построен так называемый “Рокфеллеровский дифференциальный анализатор” (его создание финансировал фонд Рокфеллера), который стал самой грандиозной и самой мощной вычислительной машиной докомпьютерной эры. Машина весила 100 тонн, а в ее конструкции были использованы около 2000 электронных ламп, более 350 км проводов, 150 электромоторов и тысячи реле. Ввод графика интегрируемой функции осуществлялся не вручную оператором, а с перфоленты.

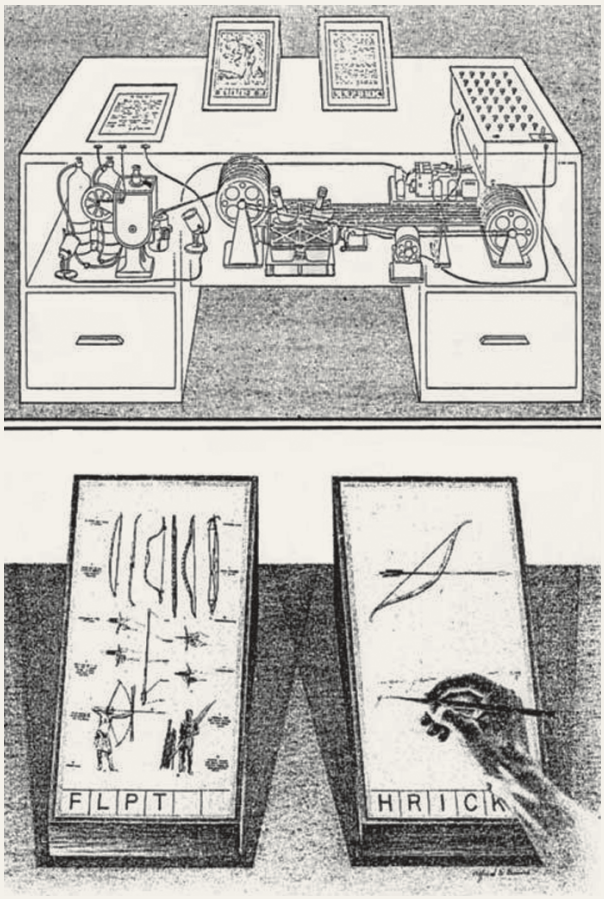


Механический дифференциальный анализатор

Рокфеллеровский дифференциальный анализатор

Еще в 1937 г. Буш инициировал работы по созданию *быстрого селектора* — устройства, предназначенного для поиска и выбора информации, записанной на микрофильмах. В итоге на их основе было построено семейство электронных вычислительных машин (аналогичных английскому “Колоссу”), использовавшихся в годы Второй мировой войны для дешифрирования перехватываемых немецких коммуникаций. Велика была роль Буша и в развертывании работ по созданию первых электронных цифровых компьютеров. Более того, он предсказал появление персонального компьютера (проект Memex, 1939 г.) и выдвинул идею гипертекста и Всемирной паутины (1945 г.)!

С 1939 по 1946 годы Буш возглавлял и координировал усилия американских ученых по созданию новых видов оружия и боевой техники — не



Рисунки к проекту Метех

случайно публикации взятого у него в 1942 г. интервью был предпослан заголовок: “Человек, который может выиграть войну”. Но он был также и политическим деятелем: в годы войны Буш входил в тесный круг из нескольких политиков, решавших вопросы, связанные с созданием атомной бомбы[4]. Опыт военных лет позволил Бушу сконструировать общепринятую сегодня в развитых странах модель взаимодействия общества, государства и науки — именно в процессе реализации этой модели родилась знаменитая Силиконовая долина, созданная энергией учеников Буша. Заметим, кстати, что Буш и сам был удачливым предпринимателем — несколько основанных им в молодости фирм выросли в гигантские электронные корпорации. Так что можно понять, почему один из биографов назвал Буша “инженером столетия”, — имея в виду, разумеется, не только его научные и инженерные достижения.

Кажется, столь выдающихся заслуг вполне хватило бы, чтобы имя человека стало нарицательным понятием — и вдруг нас уверяют, будто все эти заслуги были перевешены одним неудачным высказыванием… Однако парадоксальность ситуации заключается в том, что на самом деле Буш никогда не говорил ничего подобного!

Приведенные выше слова в действительности принадлежит другому выдающемуся ученому, одному из создателей теории нейронных сетей Уоррену Маккаллоку. В 1949 г. он написал в статье под названием “Мозг как вычислительная машина”, что электронная вычислительная машина, которая имела бы столько электронных ламп, сколько нейронов содержит человеческий мозг, потребовала бы для своего размещения здания Организации Объединенных наций в Нью-Йорке, Ниагарский водопад для обеспечения ее энергией и реку Ниагару для охлаждения. И, вообще говоря, это был отнюдь не прогноз, а констатация факта — мозг человека действительно устроен крайне сложно и содержит миллиарды нервных клеток, а электронные лампы действительно потребляют немало электроэнергии и выделяют много тепла!

Так что Буш вовсе не был неудачливым пророком. А появление упомянутого жаргонного слова объясняется, вероятно, двумя причинами — недостаточной эрудицией хакеров (или составителей упомянутых словарей) и редким, необычно звучащим, а потому запоминающимся именем ученого.

[4] В 1945 г. Буш поддержал решение об атомной бомбардировке Японии, полагая, что она приблизит окончание войны и сохранит жизнь десяткам тысяч американских солдат. Однако он считал, что вопрос о дальнейшей судьбе нового оружия должен стать предметом послевоенного обсуждения всеми участниками антигитлеровской коалиции: необходим “международный обмен всей научной и технической информацией по этому вопросу, осуществляемый международной комиссией, которая действует в интересах объединенных наций и обладает правом производить инспекции”. Позднее, в 1952 г., Буш пытался, хотя опять безуспешно, убедить американское правительство отменить испытания водородной бомбы (он их назвал “ужасным шагом”) и начать переговоры с Советским Союзом о прекращении гонки ядерных вооружений.

ПОИСК ИНФОРМАЦИИ

Про птиц и собаку

Ответы на приведенные ниже вопросы найдите в Интернете или по другим источникам информации.

1. Некий Нильс Олаф вот уже около 30 лет служит в гвардии одной из скандинавских стран и с завидной регулярностью получает очередное воинское звание. В этом не было бы ничего странного, если бы ни одно “но”: этот Нильс Олаф — птица. Какая именно?

2. Когда-то прежде у англичан эту птицу считали ой, кто “взбивает ветер”. Оно и понятно, она находится в прямом родстве с соколами и стремительность в полете для нее норма. Что это за птица?

3. В одной из азиатских столиц запрещено иметь в доме более одной собаки. Кроме того, закон строго ограничивает рост лучшего друга семьи. Что это за город? Какое ограничение по росту устанавливает закон?

Ответы (можно не на все вопросы) присылайте в редакцию.

ЯПОНСКИЙ УГОЛОК

Два судоку

Решите, пожалуйста, две японские головоломки “судоку”:

1) простую:

	9		6			4		7
4		1	9		8	5		
	8				5	9		
	7			4				
9	6			5			1	2
				6			3	
		6	3				7	
		2	7		6	1		8
7		9			2		4	

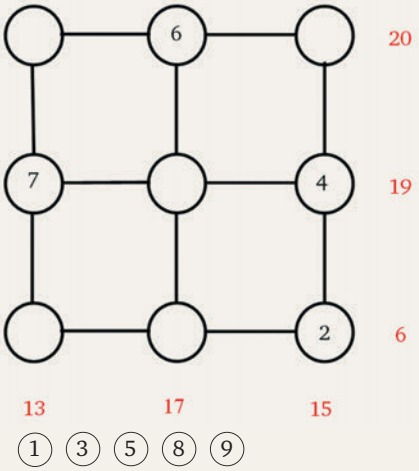
2) сложную:

	2		4					
4	6	8	1		9			
	5					4		
			9		3	5		
6					4			
			2			3		
	8	6	5				3	7
			3					6
7	3						9	4

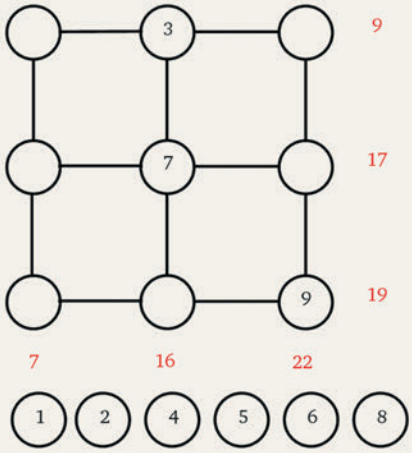
Сан-го-ку

Предлагаем читателям решить три японских головоломки другого типа — “сан-го-ку”. Их правила просты: необходимо расставить цифры в свободных кружочках на пустые места так, чтобы сумма цифр каждого ряда равнялась числу справа, а сумма цифр каждого столбца — числу снизу.

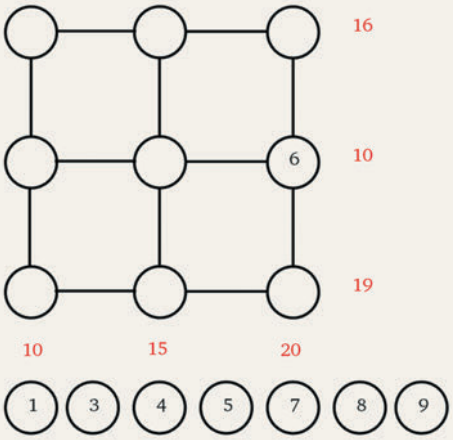
Сан-го-ку № 1



Сан-го-ку № 2



Сан-го-ку № 3



Решения (можно не всех головоломок) присылайте в редакцию.

ПРИ1Ы

Очень важный вывод

Если вы имеете опыт работы с текстовым редактором Microsoft Word, то, конечно, знаете, что в нем предусмотрена возможность проверки правильности правописания в набранном тексте. Для такой проверки надо нажать функциональную клавишу F7. Попробуйте набрать текст “хочу избежать службу в армии” (именно так и без кавычек) и нажмите F7 — Word “выдаст” вам очень важный вывод. Какой?

Программист считает в “уме”

У программиста спрашивают:

— Не помнишь, сколько будет два в четвертой степени?

— (*Без запинки*) Шестнадцать.

— А два в шестнадцатой?

— (*Без запинки*) Шестьдесят пять тысяч пятьсот тридцать шесть.

— Вот голова, ну ты даешь! Ну, а три в четвертой?

— (*После паузы*) Не помню точно. Кажется, дробное число получается…

Задача, которую вы решаете, может быть очень скромной, но если она бросает вызов вашей любознательности и если вы решите ее собственными силами, то вы сможете испытать ведущее к открытию напряжение ума и насладиться радостью победы.

Джордж Пойя

Через тернии к “звездам”

Ю.В. Пашковская,
г. Жуковский, Московская обл.

В одном из предыдущих выпусков “В мир информатики” (см. [1]) мы разбирали задание № 6 из ОГЭ по информатике. В статье по заданному алгоритму нужно было определить, какая фигура будет нарисована исполнителем Черепашкой. При решении мы опирались на следующее правило: “Для построения правильного *n*-угольника в алгоритме типа `Повтори n [Вперед 50 Направо m]` произведение числа повторений *n* на угол поворота должно быть равно 360”.

Но вот задача посложнее. Попробуйте определить, что будет нарисовано в результате выполнения алгоритма:

```
Повтори 5 [Вперед 50 Направо 144]
```
(1)

Для рисования правильного пятиугольника, согласно нашему правилу, угол поворота должен быть 360/5 = 72, то есть в два раза меньше, чем в заданном алгоритме. Если же подойти к решению с другой стороны, оттолкнувшись от угла поворота, то число повторений должно быть 360/144 = 2,5, что вообще бессмысленно. Так как же поведет себя Черепашка?

Эту задачу можно обобщить: как повлияет на результат выполнения алгоритма добавление еще одного параметра — целого числа *d*:

```
Повтори n [Вперед a Направо 360*d/n]
```
(2)

Чтобы ответить на этот вопрос, давайте проведем эксперимент. Для него удобно использовать среду программирования Scratch[5].

С помощью главного меню системы (пункт **File**) создадим проект “правильные снежинки”, а в нем — две переменные — *n* и *d* (кнопки **переменные** | **Создать переменную** — см. *рис.* 1):

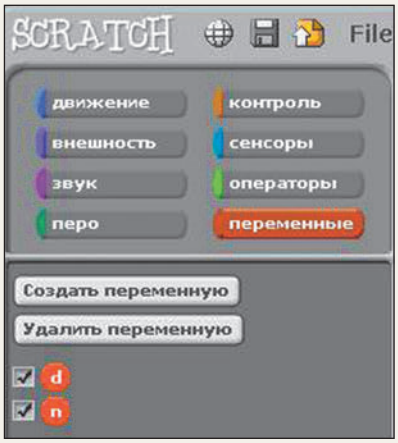


Рис. 1

Обе переменные отразятся на экране:

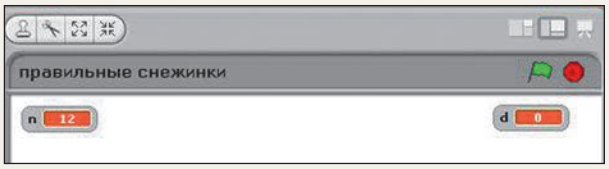


Рис. 2

Если по переменной на экране щелкнуть правой кнопкой мыши, то появится список:

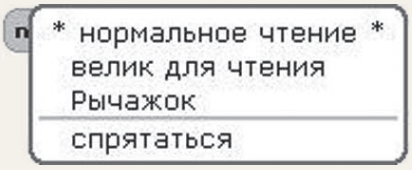


Рис. 3

Выберем пункт **Рычажок**. После этого переменная примет вид:

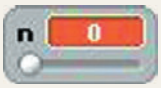


Рис. 4

Теперь, двигая белую кнопку, мы можем менять значение переменной.

Аналогично поступим и с переменной *d*.

По умолчанию диапазон изменения переменной — от 0 до 100. Чтобы его изменить, щелкнем еще раз по переменной *n* правой кнопкой мыши и выберем из списка пункт **Установить максимум и минимум рычажка**. После этого появится окошко:

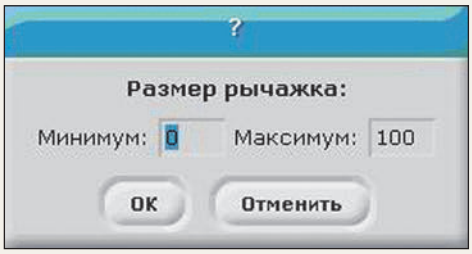


Рис. 5

Поскольку переменная *n* отвечает за количество углов в многоугольнике, назначим для нее минимум — 3, а максимум — 20.

Диапазон изменения для переменной *d* установим от 1 до 19.

Теперь нам осталось составить программу для исполнителя среды Scratch — Котенка. Она собирается, как конструктор Лего, из отдельных блоков, лежащих в разных тематических ящиках (блоки окрашены цветом ящика, в котором они находятся):

[5] В электронных материалах к данному выпуску журнала представлен установочный файл ScratchInstaller 1.4.exe этой системы. Можно также разработать алгоритм типа (2) в среде с исполнителем Черепашка или использовать графические возможности системы программирования, которую вы изучаете. — **Прим. ред.**

Рис. 6

В этой программе, кроме уже знакомого нам цикла, добавлены следующие блоки:

• запускающий блок **когда щелкнут по флажку** (ящик **Контроль**), дающий возможность запускать программу в режиме презентации;

• команды **очистить**, **поднять перо** и **опустить перо** (ящик **Перо**);

• команда **идти в точку** с координатами (0; 100);

• команда **повернуть в направлении 90**.

Последние две команды определяют начальное положение Котенка. Чтобы он не мешал воспринимать нарисованные им фигуры, щелкнем по команде **спрятаться** (ящик **Внешность**).

Теперь мы можем перейти в режим презентации, щелкнув по кнопке в верхнем правом углу (см. *рис.* 7):

Рис. 7

Варьируя значения переменных n и d, попробуйте определить, за что отвечает параметр d.

Например, при $n = 19$ и разных значениях d получаются следующие фигуры:

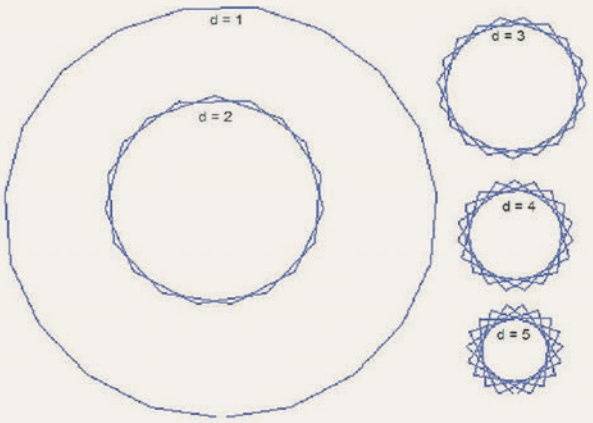


Рис. 8

Нетрудно заметить, что у всех этих фигур по 19 вершин. Однако для разных значений d фигуры отличаются. Чем? За что "отвечает" параметр d?

Нетрудно заметить, что у всех этих фигур по 19 вершин. Однако если при $d = 1$ эти вершины соединяются последовательно, то при $d = 2$ — через одну. При $d = 3$ — через две и т.д.

Таким образом, параметр d отвечает за шаг, с которым перебираются и соединяются вершины, делящие окружность на n равных частей.

Самостоятельно поэкспериментируйте: как будет выглядеть нарисованная фигура, если n и d окажутся кратны друг другу (например, при $n = 20$, $d = 4$)?

Вернемся теперь к задаче, связанной с алгоритмом (1). Поскольку 360 нацело на 144 не делится, из-за чего наше правило не может быть применено, то попробуем подобрать такие целые взаимно простые d и n, чтобы можно было воспользоваться алгоритмом типа (2). Для этого запишем:

$$360 \cdot d/n = 144$$

Разделив обе части равенства на 360, получим:

$$d/n = 2/5$$

А это значит, что на экране будет нарисована пятиконечная звезда ($n = 5$), в которой вершины будут соединяться через одну ($d = 2$).

Задачи для самостоятельной работы

1. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 7 [Вперед 100 Направо 60 Вперед 20]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) правильный треугольник;
3) правильный семиугольник;
4) правильный шестиугольник.

2. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 13 [Вперед 50 Направо 108]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) 10-угольная звезда;
3) правильный 13-угольник;
4) 13-угольная звезда.

3. Черепашке был дан для исполнения следующий алгоритм:

```
Повтори 18 [Вперед 15 Направо 54]
```

Какая фигура появится на экране?

1) незамкнутая ломаная линия;
2) правильный 18-угольник;
3) правильный 20-угольник;
4) 12-угольная звезда.

От редакции. Ответы к задачам для самостоятельной работы и ответ на вопрос относительно кратных значений n и d (см. выше) присылайте в редакцию. Можно выполнять не все задания.

Литература

1. *Пашковская Ю.В.* Об исполнителе Черепашке, или Готовимся к ОГЭ. / "В мир информатики" № 211 ("Информатика" № 10/2015).

50 лет назад

В одной из книг по вычислительной технике, выпущенной в 60-х годах прошлого века, приведена таблица, в которой представлено время выполнения различного количества операций при скорости электронно-вычислительной машины в 10 тыс. операций в секунду:

Количество операций	Время
10 000	
100 000	
1 000 000	
10 000 000	
100 000 000	
1 000 000 000	
10 000 000 000	

На основе данных этой таблицы (второй столбец в таблице в книге заполнен) делается вывод, что при такой производительности машины нельзя рассчитывать на решение задач, требующих больше, чем … операций. Какое значение указано в книге вместо многоточия?

В городе Обыворпол

В городе Обыворпол живут только обыватели, воры и полицейские (поэтому он так и называется). Полицейские всегда врут обывателям, воры — полицейским, а обыватели — ворам. Во всех остальных случаях жители Обыворпола говорят правду. Однажды несколько обыворполцев водили хоровод и каждый сказал своему правому соседу: “Я — полицейский”. Сколько обывателей было в этом хороводе?

Лесная школа танцев

В лесную школу танцев пришли слон, волк и лев. Партнершами у них были выбраны мышка, белочка и лисичка. Учитель-жираф должен расставить их в пары. Сколько вариантов составления есть у него, если белочка боится, что ее съест волк, а слон — что он раздавит мышку?

Литература

1. *Богомолова О.Б.* Логические задачи. М.: БИНОМ. Лаборатория знаний, 2005.

Сказочные гуси

В одной сказке над озерами летели гуси. На каждом озере садилась половина гусей и еще полгуся, остальные летели дальше. Все гуси сели на семи озерах. Сколько было гусей?

Три студента

Три студента — Андреев, Борисов и Воронов — учатся на различных факультетах педагогического института (историческом, физико-математическом и иностранных языков). Все они приехали из различных городов: Калязина, Твери и Вышнего Волочка, причем один из них увлекается футболом, другой — баскетболом, третий — волейболом. Известно, что:

1) Андреев не из Вышнего Волочка, а Борисов не из Твери;

2) студент, приехавший из Вышнего Волочка, учится на историческом факультете;

3) тверянин учится на факультете иностранных языков и увлекается футболом;

4) Воронов учится на историческом факультете;

5) студент физико-математического факультета не любит волейбол.

Из какого города приехал каждый студент, на каком факультете он учится и каким видом спорта увлекается?

Списки читателей, приславших ответы на задания, опубликованные в сентябрьском выпуске “В мир информатики”, будут приведены в следующем выпуске. Задержка связана с особенностями подготовки журнала (данный выпуск готовился в начале сентября).

СЕМИНАР

Обратная связь — что это такое?

Вы, уважаемый читатель, конечно, слышали о таком понятии, как “обратная связь”. Что же это такое? В Большом энциклопедическом словаре читаем: “Обратная связь — воздействие результатов функционирования какой-либо системы (объекта) на характер этого функционирования”. Иными словами, при обратной связи, когда объект А воздействует на объект В, то в результате изменения состояния В произойдет изменение состояния А и/или особенностей его воздействия на В.

Обратившись к жизненному опыту, можно вспомнить массу примеров обратной связи, применяемой в технических устройствах. Еще в 1765 году Иван Иванович Ползунов (1728–1766) устроил систему для регулирования уровня воды в котле паровой машины, соединив поплавок с краном водоподающей трубы так, что, если поплавок (объект В) опускался ниже данного уровня, кран (объект А) открывался, а если поднимался выше — закрывался.

Примеров обратной связи можно привести множество. Подробно ее свойства были изучены Христианом Гюйгенсом (1629–1695) в 1657 году. Гюйгенс исследовал обычные пружинные часы

с маятником, выяснив, как зависит ход часов от длины, положения и массы маятника. Работа машин и механизмов, имевших обратную связь (или, иначе, машин со следящими системами), поддается точному описанию и расчету. Занимается ею особая наука — теория автоматического регулирования.

В живых организмах также имеются системы регулирования с обратной связью, разумеется, во много раз сложнее. Для нормального существования живого организма необходимо строгое поддержание постоянства некоторых физико-химических величин, регулирование которых производится автоматически и протекает помимо ощущений, воли и сознания. Автоматические реакции по поддержанию постоянства внутренней среды организма (гомеостазис) присущи большинству живых организмов.

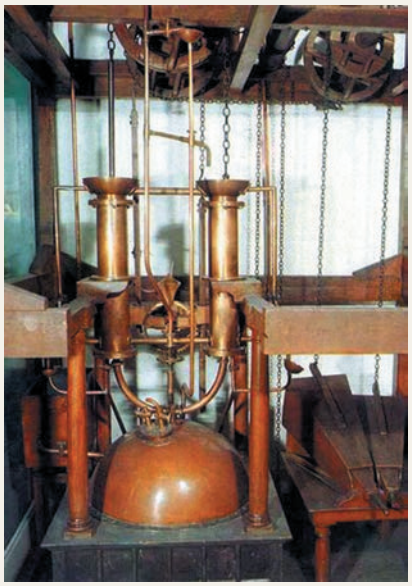


Рис. 1. Макет паровой машины И.И. Ползунова (краеведческий музей г. Барнаула)

Механизмы с обратной связью не только регулируют вегетативные функции внутренних органов, в ряде случаев они обеспечивают управление скелетной мускулатурой, устойчивость тела в поле тяжести, наилучшие условия работы рецепторов, а также многие другие безусловные рефлексы.

Английский физиолог Грей Уолтер, занимавшийся исследованием поведения живых организмов и изучением высшей нервной деятельности, не случайно обратился к техническим аналогам, которые можно было быстро построить и экспериментировать с ними в широких пределах. Сконструированный им робот-черепаха мог имитировать ряд поведенческих актов живого организма. Но еще раньше черепах Грея Уолтера появились роботы-животные, созданные отцом кибернетики — Норбертом Винером. Одну из своих моделей он назвал “моль”, другую — “клоп”. Действующая модель Винера представляла собой тележку с установленным на ней электродвигателем. Управление осуществлялось с помощью двух фоторезисторов, включенных на входы электронных усилителей. Если на фоторезисторы падало одинаковое количество света, то на выходе усилителей напряжение тоже было одинаковым, и модель оставалась неподвижной. При изменении освещенности одного из фотоэлементов равновесие нарушалось, срабатывало реле, включался электродвигатель, и тележка начинала движение, которое продолжалось до тех пор, пока не уравнивались освещенности обоих фоторезисторов. Моль была отрегулирована так, что “стремилась” двигаться к свету, а клоп, наоборот, прятался от света и обшаривал комнату в поисках темного угла.

Но вернемся к черепахам Грея Уолтера, которые прочно вошли в анналы кибернетики. Вот как рассказывает о них сам ученый: “Сейчас проявляется большой интерес к машинам, имитирующим жизнь… Машина, которой мы в основном занимались, — это маленькое создание с гладкой поверхностью и вытянутой шеей, в которой помещается единственный глаз, осматривающий окружающие предметы в поисках светового стимула, — мы назвали эту игрушку “Тестудо” или “черепахой”, которую машина напоминает своим внешним видом”.

Черепаха Элмер (электромеханический робот) представляла собой небольшую трехколесную тележку, на которой были установлены два мотора (ход вперед-назад и поворот), несколько электромагнитных реле, электронная аппаратура и питающий аккумулятор. Несмотря на простоту устройства, поведение этой черепахи было довольно сложным. Пока аккумулятор ее был заряжен, она вела себя как спокойное сытое животное, при слабом освещении или в темноте — медленно передвигалась по комнате, а столкнувшись с ножкой стола, буфетом и т.д., черепаха останавливалась, сворачивала в сторону и обходила препятствие. Если в комнате включали яркую лампу, Элмер вскоре замечал ее и направлялся к источнику света, но не подходил слишком близко, боясь “ослепления”. По мере разряда аккумулятора черепаха начинала проявлять все больший “интерес” к лампе, освещающей “кормушку”, — место для его зарядки, и, наконец, когда возникала необходимость в подзарядке, смело направлялась к источнику света.

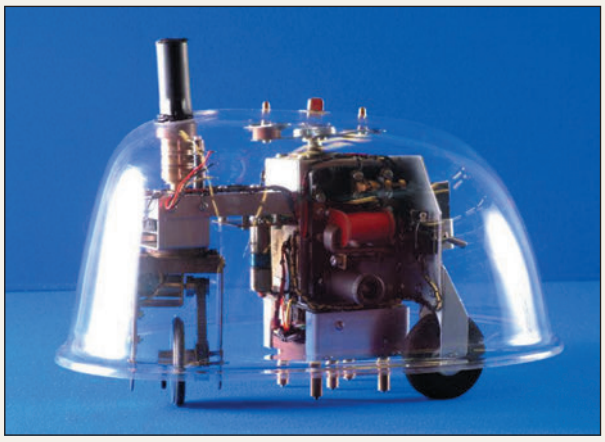


Рис. 2. “Черепаха” Элмер

Поведение Элмера можно было еще более усложнить, прикрепив к его панцирю горящую электри-

ческую лампочку. Если в комнате помещали зеркало, черепаха, как бы "узнавая" себя, устремлялась к нему и часами "рассматривала" свое отражение, то приближаясь к зеркалу, то удаляясь и поворачиваясь перед ним.

Позднее Грей Уолтер построил новую черепаху Элзи (ELSI, ***Electro-Light Sensitive*** — электронно-светочувствительный робот), которая внешне являлась точной копией своего "братца", но вела себя немного иначе: активнее реагировала на малейшие изменения освещенности, быстрее двигалась, расходовала больше энергии и чаще посещала кормушку. Если обеих черепах помещали в одну комнату, они быстро находили друг друга, сближались и начинали кружиться в своеобразном танце.

Более интересной была третья черепаха Грея Уолтера по имени Кора (Cora — ***Conditional Reflex Automat*** — автомат условного рефлекса). Этот кибернетический зверек обладал не только "зрением" и "осязанием", подобно своим предшественникам, но еще и "слухом": к его органам чувств конструктор добавил микрофон. Кроме того, черепаху можно было обучать, вырабатывая у нее что-то вроде условного рефлекса (благодаря наличию элемента памяти в виде конденсатора, способного в течение некоторого времени сохранять накопленный электрический заряд).

Грей Уолтер выработал у Коры условный рефлекс, обучив ее останавливаться перед препятствием и сворачивать в сторону по звуковому сигналу — свистку. Свисток раздавался всякий раз, когда Кора, двигаясь по комнате, натыкалась на какую-либо преграду. По сигналу свистка она останавливалась, отступала назад и сворачивала в сторону, даже если перед ней никакого препятствия не было.

Своеобразные черты поведения описанных кибернетических игрушек придавали им сходство с настоящими живыми существами, отличительной особенностью которых является умение действовать целесообразно, с учетом окружающей обстановки. В дальнейшем роботы, моделирующие поведение живых организмов, стали предметом пристального внимания ученых-кибернетиков, и не только ученых. В течение нескольких лет был создан "звериnец" кибернетических животных: черепах, лисиц, белок, собак и т.д. Объединенные общим принципом действия, эти зверюшки отличались главным образом внешним оформлением. Широкую известность получили: мышь, отыскивающая дорогу в лабиринте, — детище американского ученого Клода Шеннона (*рис*. 4); "белка", собирающая орехи и относящая их в гнездо, созданная американцем Эдмундом Беркли; лисицы Барбара и Джоб, построенные французом Альбером Докроком, и др. В СССР в Ленинградском электротехническом институте был сконструирован щенок, реагировавший на пищу и свет. Вместо свистка в качестве условного раздражителя использовалось… нажатие на его хвост.

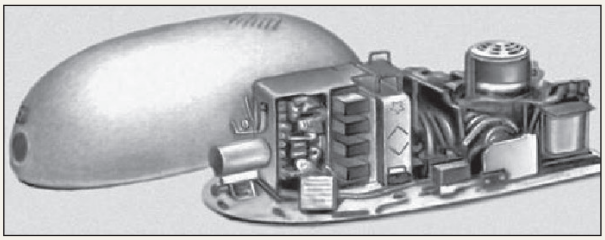


Рис. 3. Мышь К.Шеннона

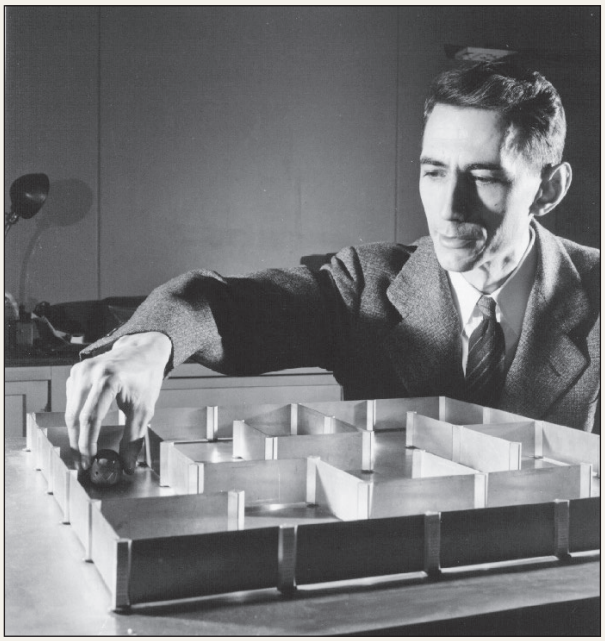


Рис. 4. Мышь К.Шеннона в лабиринте

Итак, создание кибернетического "зверинца" — что это было? Забава способных инженеров и известных ученых? Нет. Создание таких моделей — не пустое времяпрепровождение. Науку все больше и больше интересует вопрос об устройстве памяти и системах управления поведением животных и человека, а искусственные черепахи и белки — первые попытки смоделировать изучаемые явления в лабораторных условиях.

Искусственные животные помогают исследовать процессы обучения и забывания, выработки условных рефлексов. Роботы-животные, имеющие простое и понятное средство связи между собой, то есть своеобразный язык, могут осуществлять и более сложные формы "сотрудничества" и приспособления к изменяющейся внешней обстановке. Несомненно, что изучение психологии поведения и мышления с помощью роботов имеет большое будущее.

Литература

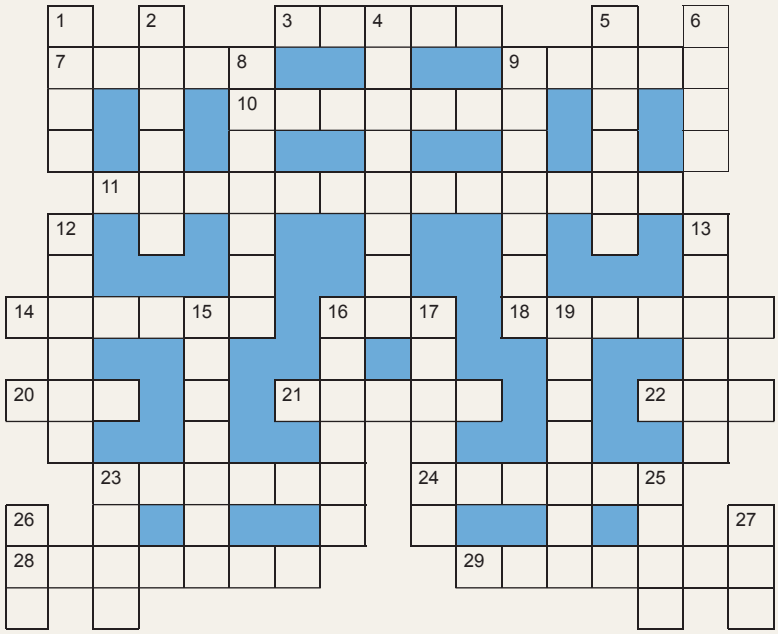
1. *Гордин А.Б.* Занимательная кибернетика. М.: Радио и связь, 1984.

2. *www.myrobot.ru/articles/hist_walter_tortoises.php*.

3. *www.barnaul-altai.ru/people/polzunov.php*.

Кроссворд

Решите, пожалуйста, кроссворд.



По горизонтали

3. Системная (материнская) …

7. Программа, обладающая способностью к самовоспроизведению.

9. Разработчик программы.

10. Электронная схема для запоминания одного бита информации.

11. Построение “заменителей” реально существующих объектов (предметов, явлений, процессов) с целью их исследования.

14. Наука о законах и формах мышления.

16. Средства массовой информации СМИ.

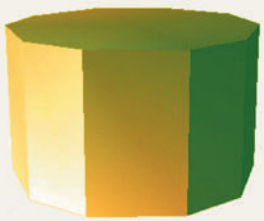
18. Так называют информацию, которая передается по Интернету.

20. Структура данных — двусторонняя очередь.

21. Один из контактов транзистора типа МОП (***М**еталл-**О**ксид-**П**олупроводник*).

22. Единица измерения скорости передачи данных.

23.



24. Шрифт наклонного начертания.

28. Разновидность, модификация, версия.

29. Часть представления числа с плавающей запятой, а также правильное, налаженное состояние, расположение чего-нибудь.

По вертикали

1. Так называют клавишу Enter.

2. Язык логического программирования.

4. Совокупность четко определенных правил для решения задачи за определенное число шагов.

5. Операция, производимая с файлом при работе компьютера.

6. Конечное число точек на плоскости, соединенных отрезками линий.

8. →.

9.

12. Название клавиши.

13. Перечень предметов, фамилий или т.п., а в программировании — совокупность элементов, каждый из которых содержит указатель на следующий элемент.

15. Элемент стандартного устройства для ввода информации в компьютер.

16. Структура, содержание.

17. Деталь печатающего элемента матричного принтера.

19. Место хранения информации в процессоре.

23. Точка подключения внешних устройств к внутренней шине микропроцессора.

25. H_2O.

26. Основание системы счисления, используемой в компьютере.

27. Буква греческого алфавита, которой, как правило, обозначают неизвестную величину.

Робот и книга

Представьте, что имеется робот, способный шагать, поднимать и опускать руки, сжимать и разжимать пальцы — и т.п. по соответствующим командам.

Ему была дана для выполнения программа для переноса книги на стол, состоящая из следующих команд:

1. Вытянуть правую руку.
2. Разжать пальцы правой руки.
3. Сжать пальцы правой руки (предполагается, что между командами 2 и 3 мы положили книгу в руку робота).
4. Опустить правую руку.
5. Повернуться на 180°.
6. Поставить правую ногу впереди левой.
7. Поставить левую ногу впереди правой.
8. Поставить правую ногу впереди левой.
9. Поставить левую ногу впереди правой.
10. Поставить правую ногу впереди левой.
11. Поставить левую ногу впереди правой.
12. Вытянуть правую руку.
13. Разжать пальцы правой руки.

Что произойдет в результате выполнения роботом такой программы?

Какова система команд робота?

Как можно уменьшить размер программы?

Еще один числовой ребус с “МАССИВОМ”

В сентябрьском выпуске “В мир информатики” был опубликован числовой ребус, в котором фигурировало слово-число **МАССИВ**. Решите, пожалуйста, еще один аналогичный ребус:

МАС = СИ^{В}

В нем, как принято в таких головоломках, одинаковыми буквами зашифрованы одинаковые цифры, разными буквами — разные цифры.

Ребусы в четверичной системе счисления. Часть 2

Здесь также одинаковые цифры зашифрованы одинаковыми буквами, разные цифры — разными буквами, но зашифрованы числа в четверичной системе счисления.

1.
$$\begin{array}{r} \mathbf{C} \\ +\ 2 \\ \hline \mathbf{D}\ \mathbf{D} \end{array}$$

3.
$$\begin{array}{r} 3 \\ +\ \mathbf{B} \\ \hline \mathbf{B}\ \mathbf{E} \end{array}$$

2.
$$\begin{array}{r} \mathbf{X} \\ +\ \mathbf{X} \\ \hline \mathrm{Y}\ 2 \end{array}$$

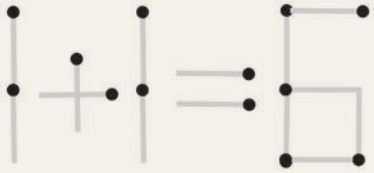
4.
$$\begin{array}{r} \mathbf{A} \\ +\ 3 \\ \hline \mathbf{B}\ \mathbf{C} \end{array}$$

Внимание! В разных ребусах одной и той же букве могут соответствовать разные цифры.

Ответы (можно не ко всем ребусам) присылайте в редакцию.

Получить верное равенство

Как, переложив одну спичку, получить верное равенство?



Задание предназначено для учащихся 1–7-х классов.

Три вопроса

1. Как, не производя никаких арифметических действий, найти остаток от деления любого двоичного числа на два?
2. Как проще всего умножить любое двоичное число на три?
3. Как проще всего умножить любое шестнадцатеричное число на десятичное число 24?

Переместить карточки

На листе бумаги нарисованы двенадцать клеток (*рис.* 1).

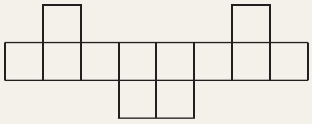


Рис. 1

На них положили восемь карточек с буквами, образующих слово ЛЕПЕСТОК (*рис.* 2):

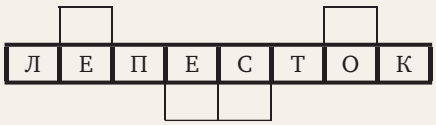


Рис. 2

Необходимо, перекладывая карточки на соседнюю свободную клетку листа, за минимальное количество перекладываний получить слово ТЕЛЕСКОП (*рис.* 3):

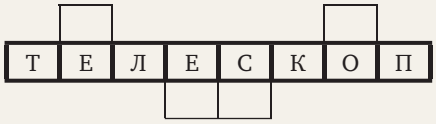


Рис. 3

Алгоритм решения задачи, пожалуйста, оформите с использованием символов “>” (перемещение карточки вправо), “<” (влево), “V” (вниз), “Λ” (вверх) в виде:

1. **Е** Λ.
2. **Т** >.

Ноутбук для… котов

Британские разработчики решили создать ноутбук, предназначенный исключительно для котов.

Владельцам ноутбука (и одновременно котов) хорошо известно, что, когда он “открыт”, то автоматически становится объектом изучения усатых “умников”.

Разработанное устройство является своеобразной игрушкой для животного и изготовлено из прочного картона. Внешне оно похоже на настоящий ноутбук, поскольку имеет клавиатуру, мышь и экран. Клавиатура представляет собой классическую когтеточку, которая позволяет котику не только поместить свои лапки на нее, но и полностью расположиться в ней.

Известно, что коты очень часто ложатся на настоящую клавиатуру или же заглядывают мордой в экран, поэтому разработчики предусмотрели все нюансы, чтобы животным было удобно. Мышь также адаптирована под хвостатого пользователя.

На экране картонного “девайса” будут показываться различные картинки, поглощающие внимание котов. Этот экран представляет собой панель со сменными бумажными картинками.

В Британии уверены, что эта игрушка поможет хозяевам, у которых коты страстно любят царапать технику.

По материалам сайта naked-science.ru

ПРОЕКТНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

Об экономичности… систем счисления

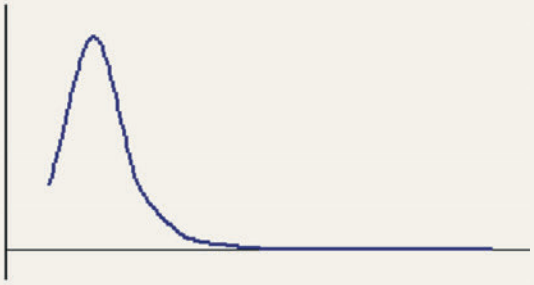
Мы привыкли, что понятие “экономичность” применяется к приборам, устройствам и т.п. Но оно связано и с системами счисления. Под экономичностью системы счисления обычно понимается количество чисел, которое можно записать в данной системе с помощью определенного количества знаков. Поясним это на примере.

Чтобы в десятичной системе записать 1000 чисел (от 0 до 999), необходимо 30 знаков (по 10 цифр для каждого из трех разрядов). А, например, в двоичной системе с помощью 30 знаков можно записать 215 различных чисел (так как для каждого двоичного разряда нужны только две цифры 0 и 1, то с помощью 30 цифр мы можем записывать числа, содержащие до 15 разрядов). Но $2^{15} > 1000$, поэтому, имея 15 двоичных разрядов, можно записать больше различных чисел, чем с помощью трех десятичных. Таким образом, двоичная система более экономичная, чем десятичная.

Но какая из систем счисления самая экономичная? Для ответа на этот вопрос рассмотрим следующую конкретную задачу. Пусть в нашем распоряжении имеется 60 знаков. Мы можем, разбив их на 30 групп по два элемента в каждой, записать с их помощью в двоичной системе любое число, имеющее не больше 30 двоичных разрядов, то есть в общей сложности 2^{30} чисел. Те же 60 знаков мы можем разбить на 20 групп по три элемента и, пользуясь троичной системой, записать 3^{20} различных чисел. Далее, разбив 60 знаков на 15 групп по четыре элемента в каждой, можно применить четверичную систему и записать 4^{15} чисел и т.д. В частности, воспользовавшись десятичной системой (то есть разбив все знаки на шесть групп по десять элементов в каждой), мы могли бы записать 10^6 чисел, а применив шестидесятеричную систему, можно было бы с помощью 60 знаков записать только 60 чисел.

Посмотрим, какая из возможных здесь систем самая экономичная, то есть позволяет записать с помощью данных 60 знаков наибольшее количество чисел. Иными словами, речь идет о том, какое из чисел 2^{30}, 3^{20}, 4^{15}, 5^{12}, 6^{10}, 10^6, 12^5, 15^4, 20^3, 30^2, 60 наибольшее.

Предлагаем читателям получить ответ на обсуждаемый вопрос с помощью электронной таблицы Microsoft Excel или подобной программы, построив график зависимости перечисленных чисел от основания соответствующей системы счисления (см. рисунок ниже).



GAMES.EXE

Три кучки спичек

Положите на стол три кучки спичек. В одну кучку положите 11 спичек, во вторую — 7, в третью — 6. Перекладывая спички из одной кучки в другую, нужно сделать так, чтобы в каждой кучке было бы по 8 спичек (это возможно, так как общее число спичек — 24 — делится на три без остатка). При этом требуется соблюдать следующее правило: к любой кучке разрешается дополнять ровно столько спичек, сколько в ней есть. Например, если в кучке шесть спичек, то и добавлять к ней можно только шесть, если в кучке четыре спички, то и добавлять к ней можно только четыре.

Решение, пожалуйста, оформите в виде:

Кучка	Начальное состояние	Первый ход	Второй ход	…
Первая	11			
Вторая	7			
Третья	6			

ВНИМАНИЕ! КОНКУРС!

Конкурс № 113

Участникам этого конкурса предлагаем решить задачу из предыдущей статьи и ответить на вопросы в задании “Три вопроса”.

Ответы (можно не на все вопросы) отправьте в редакцию до 25 декабря по адресу: 121165, Москва, ул. Киевская, д. 24, “Первое сентября”, “Информатика” или по электронной почте: *vmi@1september.ru*.

Дорогие ребята! Присылайте ответы на вопросы наших конкурсов, даже если по каким-то причинам (поздняя доставка газеты или др.) вы подготовите их позже срока, указанного в условии конкурса. Мы обязательно рассмотрим все присланные ответы и при необходимости дополнительно отметим лучшие из них.

При оформлении ответов будьте, пожалуйста, внимательны — указывайте все необходимые сведения о себе (имя, пожалуйста, приводите полностью), населенном пункте, учебном заведении и учителе информатики.

“ЛОМАЕМ” ГОЛОВУ

Двусторонний магический квадрат

Магическим квадратом называют таблицу из n строк и n столбцов, заполненную n^2 числами так, что суммы чисел в каждой строке, в каждом столбце и на обеих диагоналях равны одному и тому же числу. Выделяют традиционные (нормальные) магические квадраты, заполненные числами от 1 до n^2, и нетрадиционные, в которых могут быть любые натуральные числа. Например, один из возможных вариантов традиционного магического квадрата при $n = 3$ выглядит так:

6	1	8
7	5	3
2	9	4

Предлагаем читателям подумать над головоломкой, которую придумал В.И. Красноухов из г. Климовска Московской обл. Имеется набор из девяти фишек с числами на обеих сторонах (1 и 2, 3 и 4, 3 и 5, 7 и 7, 5 и 8, 6 и 6, 7 и 9, 8 и 9, 10 и 11). Требуется расположить квадратные фишки с числами в прозрачной коробочке или пакетике так, чтобы магический квадрат образовался на каждой стороне одновременно. Фишки, которые вы можете изготовить из картона, можно как угодно переставлять, поворачивать, переворачивать.

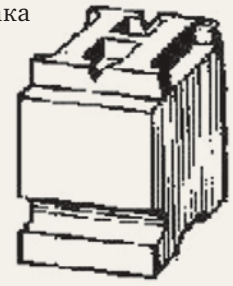
Ответ в виде двух таблиц (“виды квадрата с двух сторон”) присылайте в редакцию.

ИСТОРИЯ ИНФОРМАТИКИ

Об азбуке Морзе

Известно, что к так называемым “неравномерным” системам кодирования символов относятся системы, в которых разным символам соответствуют коды различной длины. В их числе — азбука Морзе.

Рассказывают, что, создавая свой код, Морзе отправился в ближайшую типографию и подсчитал число литер в наборных кассах (литера — металлический, пластмассовый или деревянный брусок прямоугольного сечения с рельефным изображением буквы или знака на одной из его сторон — см. рисунок; наборная касса — ящик, в котором помещались литеры. В те времена, да и в течение длительного времени потом, текст в книгах, газетах и т.п. набирался вручную из отдельных литер).



Буквам и знакам, для которых литер в этих кассах было припасено больше, он сопоставил более короткие кодовые обозначения (ведь эти буквы повторяются в тексте чаще). Так, например, в русском варианте азбуки Морзе буква *е* передается одной точкой, а редко встречающаяся буква *ц* — набором из четырех символов.

ПРЕДЛАГАЮ КОЛЛЕГАМ

О признаках делимости

Д.М. Златопольский,
Москва

Как известно, существуют простые признаки, позволяющие определить, что то или иное число делится, например, на 3, на 5, на 9 и т.п. Напомним эти признаки.

1. *Признак делимости на 3*: число делится на три, если сумма его цифр делится на три. Например, число 257 802 (сумма цифр 2 + 5 + 7 + 8 + 0 + 2 = = 24) делится на три, а число 125 831 (сумма цифр 1 + 2 + 5 + 8 + 3 + 1 = 20) на три не делится.

2. *Признак делимости на 5*: число делится на пять, если его последняя цифра есть пять или 0 (т.е. если на пять делится число единиц его последнего разряда).

3. *Признак делимости на 2* аналогичен предыдущему: число делится на два, если на два делится число единиц его последнего разряда.

4. *Признак делимости на 9* аналогичен признаку делимости на три: число делится на девять, если сумма составляющих его цифр делится на девять.

Доказательство справедливости этих признаков не представляет труда. Рассмотрим, например, признак делимости на три. Он основан на том, что единица каждого из разрядов десятичной системы (то есть числа 1, 10, 100, 1000 и т.д.) при делении на три дает остаток 1. Поэтому всякое число

$$(a_n a_{n-1} a_{n-2} \ldots a_1 a_0)_{10},$$

то есть число

$$a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2} + \ldots + a_1 \cdot 10 + a_0$$

можно записать в виде

$$(a_n + a_{n-1} + a_{n-2} + \ldots + a_1 + a_0) + B,$$

где число B делится на три без остатка. Отсюда следует, что

$$a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2} + \ldots + a_1 \cdot 10 + a_0$$

делится на три в том и только в том случае, если на три делится число $a_n + a_{n-1} + a_{n-2} + \ldots + a_1 + a_0$, т.е. сумма цифр исходного числа.

Признак делимости на пять вытекает из того, что число 10 — основание системы счисления — делится на пять, поэтому все разряды, кроме разряда единиц, при делении на пять обязательно дают в остатке ноль. На том же самом основан и признак делимости на два: число четное, если оно кончается четной цифрой.

Признак делимости на девять, как и признак делимости на три, вытекает из того, что каждое число вида 10^k при делении на девять дает в остатке 1.

Из сказанного ясно, что все эти признаки связаны с представлением чисел именно в десятичной системе и что они, вообще говоря, неприменимы, если пользоваться системой счисления с каким-либо другим основанием, отличным от 10. Например, число 86 в восьмеричной системе записывается в виде 126_8 (так как $86 = 8^2 + 2 \cdot 8 + 6$). Сумма цифр равна девяти, но 86 не делится ни на девять, ни на три.

Однако для каждой позиционной системы счисления можно сформулировать свои признаки делимости на то или иное число.

Рассмотрим несколько примеров.

Будем писать числа в двенадцатеричной системе и сформулируем для такой записи признак делимости на шесть. Так как число 12 — основание системы счисления — делится на шесть, то число, записанное в двенадцатеричной системе, делится на шесть в том и только в том случае, если на шесть делится его последняя цифра (здесь то же самое положение, что и с делимостью на пять или на два в десятичной системе).

Так как числа 2, 3 и 4 тоже служат делителями числа 12, то справедливы следующие признаки делимости: число, записанное в двенадцатеричной системе, делится на два (соответственно, на три и на четыре), если его последняя цифра делится на два (соответственно, на три и на четыре).

Задания для самостоятельной работы учащихся

1. Определите признаки делимости:

1) в двоичной системе — на 2 и на 4;

2) в троичной системе — на 2 и на 3;

3) в четверичной системе — на 2, на 3 и на 4;

4) в шестеричной системе — на 2, на 3, на 5 и на 6;

5) в восьмеричной системе — на 2, на 4, на 7 и на 8;

6) в двенадцатеричной системе — на 8, на 9 и на 11;

7) в q-ричной системе — на $(q - 1)$.

2. Некоторое десятичное число кратно семи. Установите, будет оно кратно семи:

1) в семеричной системе счисления;

2) в двоичной системе счисления?

3. В натуральном десятичном числе A переставили цифры и получили число B. Известно, что $A - B$ состоит из единиц. Найдите наименьшее возможное количество единиц в разности.

4. Из целого числа A вычли число B, полученное перестановкой цифр A. Разность $A - B$ состоит из 2013 единиц. Все указанные числа (A, B, $A - B$, 2013) даны в q-ричной системе счисления. Найдите (в десятичной системе счисления) сумму всех возможных значений q.

Ответы будут опубликованы в следующем выпуске.

журнал

Информатика – Первое сентября

ПОДПИСКА НА ОДИН ЖУРНАЛ

НА ПОЧТЕ ПО КАТАЛОГУ «РОСПЕЧАТЬ» ИЛИ НА САЙТЕ www.1september.ru

НА ПЕРИОД С 1 ЯНВАРЯ 2016 ПО 30 ИЮНЯ 2016 (I ПОЛУГОДИЕ)

Варианты подписки

- Печатная версия – **2200** р. (приходит на почтовый адрес)
- Электронная версия на CD – **800** р. (приходит на почтовый адрес)

- Электронная версия (приходит в Личный кабинет) – **500** р.

Подробности на сайте **www.1september.ru**

ПОДПИСКА НА ВСЕ ЖУРНАЛЫ ДЛЯ ВСЕХ РАБОТНИКОВ ШКОЛЫ

НА ПЕРИОД С 1 АВГУСТА 2015 ПО 30 ИЮНЯ 2016 (ВЕСЬ УЧЕБНЫЙ ГОД)

Каждому учителю доступны в Личном кабинете

- 24 журнала (включая журнал «Информатика»)
- 35 курсов повышения квалификации
- 460 брошюр по всем предметам

Стоимость участия школы в проекте

- 6 тысяч рублей от школы за весь 2015/16 учебный год независимо от количества педагогических работников

Оформление участия в проекте – круглогодично на сайте **digital.1september.ru**

Подписка на журнал и участие в проекте могут быть оформлены как от организации, так и от физического лица. При оформлении подписки **на сайте** оплата производится либо по квитанции в отделении банка, либо электронными платежами on-line

VISA MasterCard QIWI WebMoney