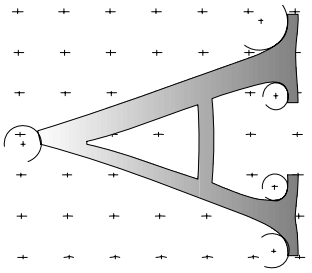
№ 11 (516)
1–15 июня 2006



# ИНФОРМАТИК

Методическая газета для учителей информатики

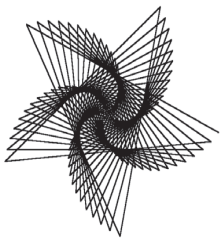
«Жаркое лето-2006»

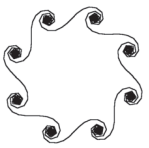
Алгоритмизация и программирование

## СОДЕРЖАНИЕ НОМЕРА

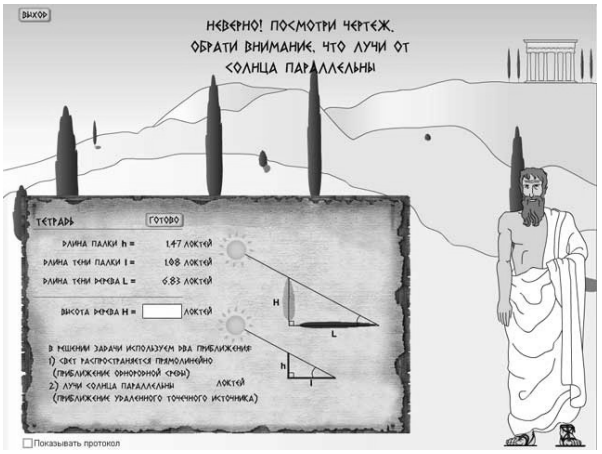











ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ «ПЕРВОЕ СЕНТЯБРЯ»
ГАЗЕТА «ИНФОРМАТИКА»
ОТДЕЛЕНИЕ ПЕДАГОГИЧЕСКОГО ОБРАЗОВАНИЯ ФГП МГУ ИМ. М.В. ЛОМОНОСОВА

# ОБЪЯВЛЯЮТ НАБОР СЛУШАТЕЛЕЙ НА КУРСЫ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

## НА 2006/2007 УЧЕБНЫЙ ГОД, ПЕРВЫЙ ПОТОК

Курсы проводятся в режиме дистанционного обучения (взаимодействие со слушателями производится посредством обычной или, при наличии у слушателя возможности, электронной почты). Продолжительность обучения — 7 месяцев, нормативный срок освоения учебного материала — 72 часа. Лекционный материал (8 лекций) и контрольные (2 работы) будут публиковаться на страницах газеты «Информатика» (для курса 07-007) или отправляться по почте (для остальных курсов). Итоговую работу слушатели будут выполнять в своих учебных заведениях.

После успешного окончания курсов слушатели получат удостоверение установленного образца о прохождении курсов повышения квалификации от Педагогического университета «Первое сентября» и Отделения педагогического образования ФГП МГУ им. М.В. Ломоносова.

Стоимость обучения составляет 790 рублей за один курс при оплате до 30 июня 2006 г. (990 рублей при оплате с 1 июля до 30 октября).

| **В 2006/2007 учебном году мы предлагаем четыре курса по вашей специальности:** | |
|---|---|
| **Код** | **Курс** |
| 07-001 | *И.Г. Семакин*. **Информационные системы в базовом и профильном курсах информатики** |
| 07-002 | *Е.В. Андреева*. **Методика обучения основам программирования на уроках информатики** |
| 07-006 | *А.А. Дуванов*. **Основы web-дизайна и школьного «сайтостроительства»** |
| 07-007 | *И.Н. Фалина, В.Ф. Бурмакина*. **Как готовиться к тестированию по проверке ИКТ–компетенции школьников** |
| **Мы также предлагаем один общепедагогический курс, предназначенный для всех работников образования:** | |
| 21-001 | *С.С. Степанов*. **Теория и практика педагогического общения** |

Для зачисления на курсы необходимо прислать в Педагогический университет «Первое сентября» заявку. Пожалуйста, используйте только приведенный ниже бланк или его ксерокопию. Регистрация слушателей производится с 1 апреля по 30 сентября 2006 г. После регистрации вам будет выслан комплект документов с правилами обучения и счетом для оплаты. Вы оплатите счет лишь в том случае, если вас устроят предлагаемые условия (факт подачи заявки ни к чему не обязывает).

**ЗАЯВКА** Прошу выслать мне комплект документов для зачисления на курсы повышения квалификации. 07-11

**ФАМИЛИЯ**

**ИМЯ**

**ОТЧЕСТВО**

**ИНДЕКС** **АДРЕС**

Телефон (с кодом города): (_____) _____

Электронный адрес (если есть): _____

Место работы: _____

Должность: _____ Стаж работы по специальности: _____

***ВНИМАНИЕ!*** *К обучению на курсах повышения квалификации допускаются сотрудники образовательных учреждений, работающие по соответствующей специальности.*

**Я хочу пройти обучение по курсам (укажите коды выбранных вами курсов):**

__ – ___ __ – ___ __ – ___ __ – ___

Если вы обучались в 2005/2006 году на наших курсах, укажите, пожалуйста, ваш идентификатор: ____

**Заявки следует направлять по адресу: ул. Киевская, д. 24, г. Москва, 121165, Педагогический университет «Первое сентября». Справки по тел.: (495) 249-47-82**

# VBA в приложениях к Excel, Word и Power Point

**О.А. ЖИТКОВА, Т.И. ПАНФИЛОВА,**
*Москва*

Окончание. Начало в № 1—5, 7, 9, 10/2006.

## Занятие № 9. VBA в приложении к Word

Пакет Word в сочетании с VBA предоставляет широкие возможности для работы с текстом или фрагментами текста. Можно вызвать для обработки какой-либо абзац, предложение и т.д.

Для доступа к тем или иным фрагментам документа используют такие объекты, как **Sections** (разделы), **Paragraphs** (абзацы), **Sentences** (предложения), **Words** (слова) и др.

*Пример:*

| | |
|---|---|
| Words(6) | дает возможность перейти к шестому слову в документе |
| Sentences(11) | переход к одиннадцатому предложению |

После перехода к определенной части документа можно использовать объект **Font**, позволяющий изменять свойства шрифта. Этот объект содержит методы, находящиеся в панели меню "Формат | Шрифт".

Объект **Range** является базовым в операции редактирования. Его можно рассматривать как непрерывный фрагмент текста (никак не связанный с выделенной областью), который не зависит от положения курсора. **Range** включает в себя такие объекты, как **Words** (слова) или **Sentences** (предложения); при обращении к ним сам объект **Range** указывать не нужно, тогда как для **Sections** (разделы) или **Paragraphs** (абзацы) это делать необходимо.

Объект **Range** поддерживает основные методы, которыми пользуются при форматировании текста (такие, как "Копировать", "Вставить", "Вырезать", команды меню "Формат").

Для выполнения заданий потребуется применение функций, работающих с символьными переменными.

**Стандартные функции VBA для символьных переменных**

| | | |
|---|---|---|
| 1 | `ASC(X$)` | переводит двоичный код символа в десятичное число |
| 2 | `LEN(X$)` | находит длину символьной переменной, включая конечные пробелы |
| 3 | `MID$(X$,Y,Z)` | выдает последовательность `Z` символов из строки `X$` с позиции `Y` |
| 4 | `STR$(X)` | преобразует значение `X` из числовой формы в символьную |
| 5 | `CHR$(X)` | преобразует код в символ |
| 6 | `LEFT$(A$,N)` | выделяет `N` левых символов строки `A$` |
| 7 | `RIGHT$(X$,N)` | выделяет `N` правых символов строки `X$` |

*Пример:*

| | |
|---|---|
| `Primer.Sentences(2).Font.ColorIndex = wdGreen` | в документе *Primer* во 2-м предложении устанавливается зеленый цвет шрифта |
| `Primer.Sentences(2).Font.Size = 16` | в документе *Primer* во 2-м предложении устанавливается шрифт размером 16 пунктов |
| `Primer.Sentences(2).Font.Underline = wdUnderlineDouble` | в документе *Primer* во 2-м предложении устанавливается шрифт с двойным подчеркиванием |
| `Primer.Sentences(2).Font.Name = "Arial"` | в документе *Primer* во 2-м предложении устанавливается шрифт "Arial" |

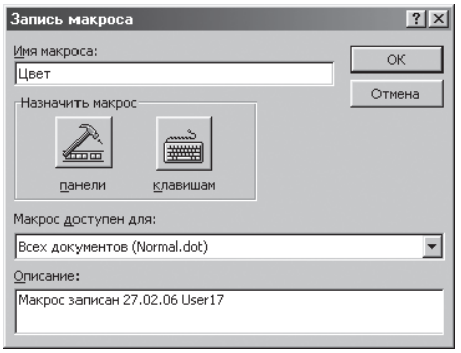
*Пример:*

| | |
|---|---|
| `Primer.Range(1,15).Bold = True` | символы с 1-го по 15-й выделяются жирным шрифтом |
| `Primer.Paragraphs(5).Range.InsertAfter "ПРИВЕТ"` | в документ *Primer* после 5-го абзаца вставляется текст "ПРИВЕТ" |
| `Primer. Sentences(5).InsertAfter "ПРИВЕТ"` | текст "ПРИВЕТ" вставляется после 5-го предложения |
| `Primer. Words(5).InsertAfter "ПРИВЕТ"` | текст "ПРИВЕТ" вставляется после 5-го слова |

Для знакомства с примерами кодов целесообразно воспользоваться приемом, когда вы форматируете часть документа в режиме записи макроса, а потом анализируете код макроса и редактируете его.

Для записи макроса необходимо:

1. Выбрать в меню "Сервис | Макрос | Начать запись".




2. Ввести в окне "Запись макроса" в поле "Имя макроса" имя нового макроса.

3. Выбрать в списке "Макрос доступен для" шаблон или документ, в котором будет храниться макрос.

4. Ввести описание макроса в поле "Описание".

5. Ввести в поле "Назначить макрос" кнопку "Панели" (для назначения макросу кнопки панели инструментов или команды меню); кнопку "Клавишам" (для назначения макросу сочетания клавиш); если макросу назначение не требуется, нажмите кнопку "ОК". Если в процессе работы вы часто используете какой-либо макрос, то целесообразно назначить ему кнопку или сочетание клавиш.

6. Выполнить действия, которые нужно включить в макрос.

Следует помнить, что для записи действий надо использовать клавиатуру, поскольку действия, произведенные с помощью мыши, не записываются.

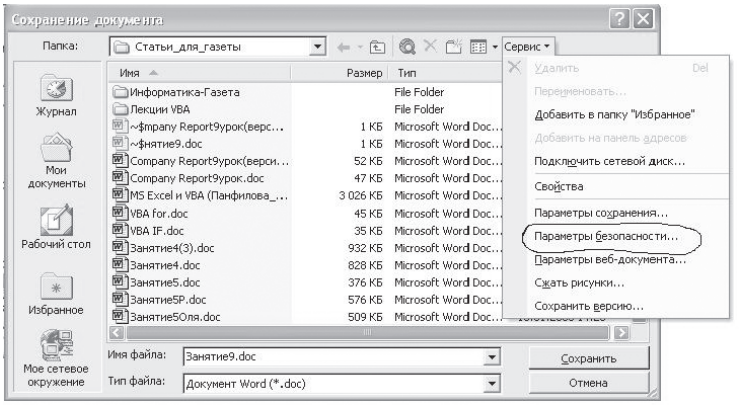
7. Нажать после записи макроса кнопку "Остановить запись".

Существует одна особенность при сохранении готового документа Word, в котором находятся макросы и процедуры. Если вы вспомните работу с макросами в Excel, то при открытии готового документа там всегда задается вопрос: "Отключать макросы при открытии или нет?". И если вы укажете, что макросы надо отключить, то подготовленные кнопки работать не будут.

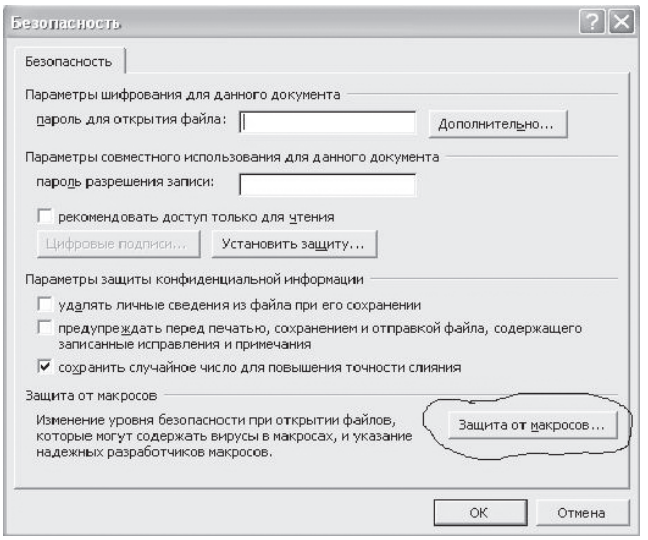
В редакторе Word, в отличие от Excel, установлена сильная защита от макросов (макросы рассматриваются как вирусы), и при повторном открытии одного и того же документа, в котором все программы и макросы работали, запроса на включение или отключение макросов не будет, и работать также ничего не будет.

Поэтому при сохранении документа в Word вы должны выполнить следующую последовательность действий:

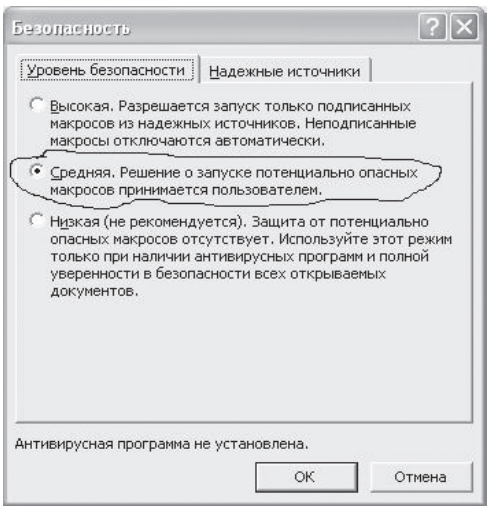
1. Выбрать "Файл | Сохранить как | Сервис". Появится диалоговое окно "Сохранение документа".



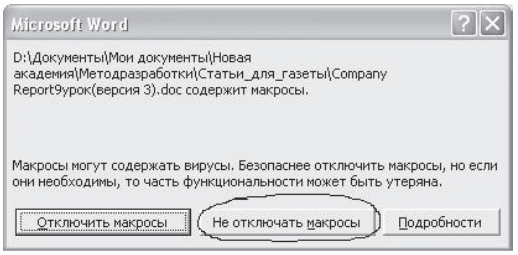

2. Выбрать в меню "Сервис" пункт "Параметры безопасности". Появится диалоговое окно "Безопасность".




3. Щелкнуть по кнопке "Защита от макросов", чтобы изменить уровень безопасности файла (по умолчанию установлен *Высокий* уровень безопасности, поэтому при повторном открытии файла ваши программы выполняться не будут).

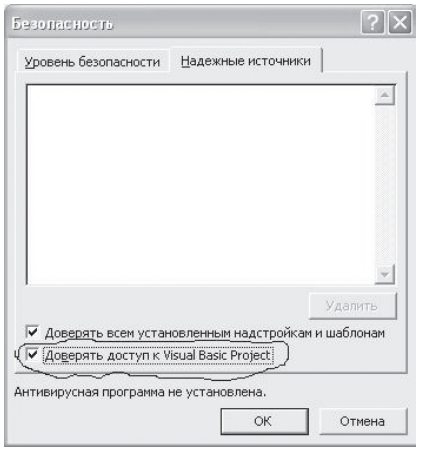


4. Выбрать в появившемся диалоговом окне на закладке "Уровень безопасности" *Средний* или *Низкий* уровень безопасности. При выборе *Среднего* уровня безопасности при открытии документа будет появляться запрос на включение или отключение макросов. Для того чтобы программы и макросы в документе работали, надо выбрать "Не отключать макросы".




При выборе *Низкого* уровня безопасности диалоговое окно появляться не будет, и макросы будут подключаться автоматически.

5. Перейти на закладку "Надежные источники" и включить флажок "*Доверять доступ к Visual Basic Project*".




6. Сохраните документ.

## Практическая работа № 9–1 "Работа в Word"

Спроектировать кнопки, при помощи которых можно провести разбор предложения. При установке курсора на найденное слово и нажатии на соответствующую клавишу слово должно окрашиваться в определенный цвет и подчеркиваться.

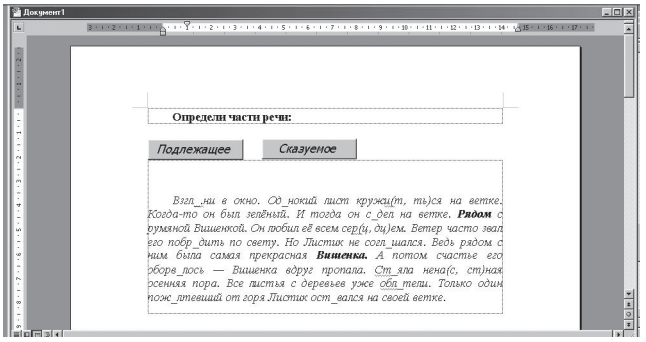
Порядок работы.

1. Набрать текст в редакторе WORD.
2. Выбрать в меню "Сервис | Макрос | Начать запись".
3. Дать макросу имя (например, "Имя_Существительное").
4. Ввести описание макроса в поле "Описание".
5. Определить в поле "Назначить макрос" способ, с помощью которого будет вызываться макрос (кнопка или сочетание клавиш).
6. Выполнить действия, которые нужно включить в макрос.

В нашем случае последовательность действий будет такая: установить курсор перед любым словом; выделить это слово; установить цвет шрифта — красный, одинарное подчеркивание.

*Не забывайте, что для записи действий надо использовать клавиатуру!*

7. Нажать после записи макроса кнопку "Остановить запись".
8. Выполнить те же действия для определения сказуемого (только цвет шрифта установить зеленый и задать двойное подчеркивание).
9. Подготовить кнопку "CommandButton1" на рабочем листе выше текста, назвать ее "Подлежащее".
10. Просмотреть записанную программу, скопировать ее и переписать через режим написания программы *Исходный текст* в кнопку "Подлежащее".
11. Проделать те же действия и для кнопки "Сказуемое".
12. Задать уровень безопасности документа.
13. Сохранить работу. Она будет выглядеть примерно так:

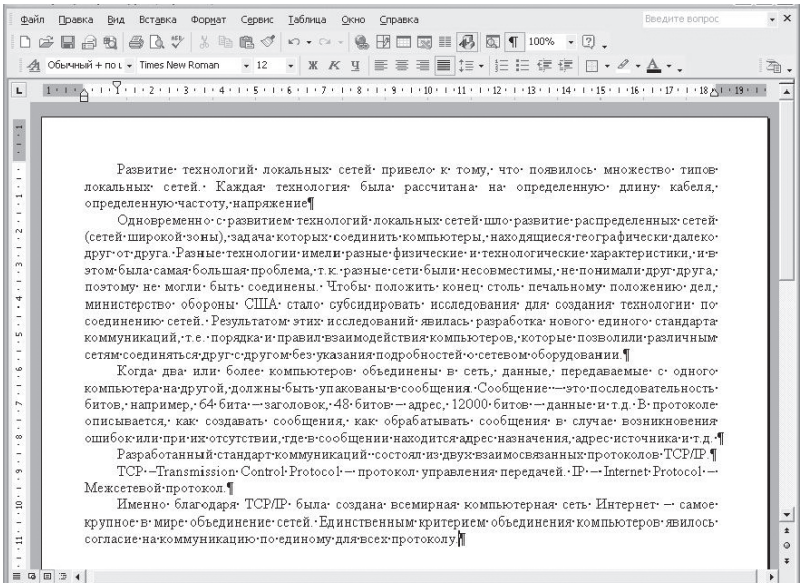


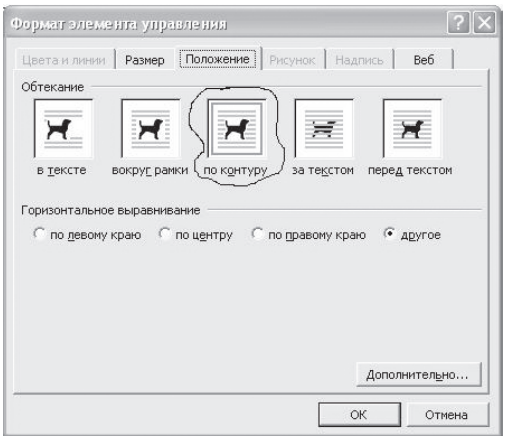

## Практическая работа № 9–2 "Количество абзацев в документе"

Рассмотреть документ, который состоит из нескольких абзацев. Задача заключается в том, чтобы программно определить количество абзацев в документе; озаглавить текст; в конце документа вывести сообщение о количестве абзацев.

Мы знаем, что в программе Word количество абзацев определяется по символу ¶, который появляется каждый раз после нажатия на клавишу Enter. Но проверять и подсчитывать количество этих символов нецелесообразно, поскольку существуют стандартные методы. Вспомним, что документ, с которым мы работаем, называется активным документом, и в программе к нему будем обращаться *ActiveDocument*.

Возьмем для примера текст об Интернете, а именно — о передаче информации и протоколах. Текст содержит 6 абзацев, судя по количеству специальных символов.



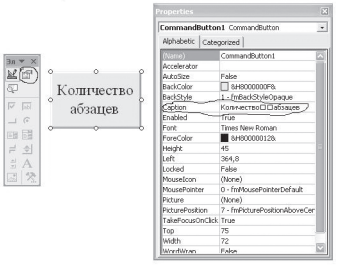




*Порядок работы.*

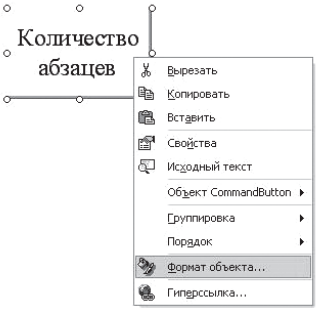
1. Вызвать панель элементов управления ("Вид | Панели инструментов | Элементы управления").

2. Подготовить кнопку "CommandButton1" на рабочем листе ниже текста.

3. Переименовать кнопку в "Количество абзацев". Для этого на панели элементов управления вызовите "Свойства". Появится панель "Properties". Выберите свойство "Caption" и в правой колонке наберите название.




4. Придать кнопке свойство обтекания по контуру, чтобы кнопку можно было перемещать по листу. Для этого на объекте кнопка следует вызвать контекстно-зависимое меню и выбрать "Формат объекта". Появится диалоговое окно "Формат элемента управления". На вкладке "Положение" выбрать вариант "По контуру".




5. Выбрать в контекстно-зависимом меню этого объекта команду "Исходный текст" или два раза щелкнуть по кнопке. Откроется редактор режима написания программы.

*Алгоритм выполнения задания:*

1. Объявить переменные:

a) `k` — переменная для подсчета количества абзацев в документе;

b) `Text` — переменная для формирования текста-ответа о количестве абзацев;

c) `ZAGOLOVOK` — переменная, которая определит область для вывода заголовка.

2. Определить количество абзацев с помощью метода `Count` объекта `Paragraphs`.

3. Сформировать текст вывода сообщения.

4. Вставить дополнительный абзац после последнего абзаца документа для вывода сообщения.

5. Вывести текст-сообщение в подготовленный абзац.

6. Сформировать область для вывода заголовка "Протоколы".

7. Вывести название "Протоколы".

8. Установить шрифт и размер заголовка.

9. Вставить символ конца абзаца для того, чтобы название и начало текста были на разных строках.

Можно дополнить эту задачу. Например, сообщение о количестве абзацев пусть выводится красным цветом и 14-м размером шрифта. Этот фрагмент учащиеся могут выполнить самостоятельно. Обратите внимание, что место этого фрагмента программы очень важно: он должен располагаться только в этой части программы, а не после работы с заголовком.

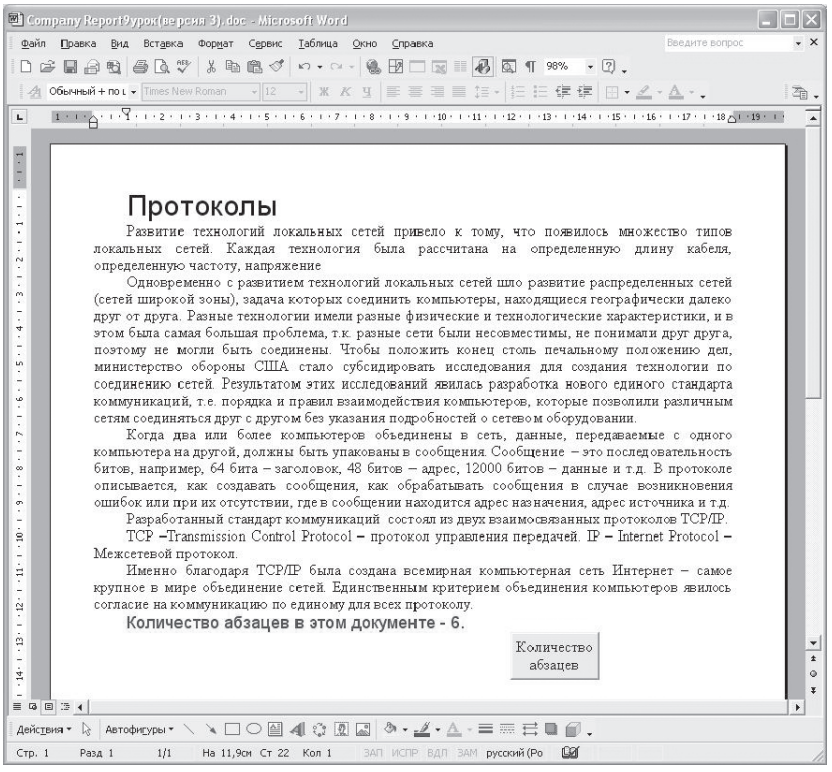
Результат выполнения программы представлен на рисунке (см. с. 7).

## Практическая работа № 9–3 "Количество букв *а* (русских) в абзаце"

Эта практическая работа выполняется с тем же документом, что и работа № 9—2 (с документом, который состоит из нескольких абзацев). Задача за-

*Текст программы.*

| Строка программы | Пункт алгоритма |
|---|---|
| `Public Sub CommandButton1_Click()` | Начало |
| `Dim k As Integer` | 1a |
| `Dim Text As String` | 1b |
| `Dim ZAGOLOVOK As Range` | 1c |
| `Dim REZULTAT As Range` | Эта часть для выполнения дополнительного задания |
| `k = ActiveDocument.Paragraphs.Count` | 2 |
| `Text = "Количество абзацев в этом документе - " & k & "."` | 3 |
| `ActiveDocument.Paragraphs(k).Range.InsertParagraphAfter` | 4 |
| `ActiveDocument.Paragraphs(k + 1).Range.InsertBefore Text` | 5 |
| `Set REZULTAT = ActiveDocument.Paragraphs(k + 1).Range With REZULTAT .Font.Name = "Arial" .Font.Size = 14 .Font.ColorIndex = wdDarkRed End With` | Эта часть для выполнения дополнительного задания |
| `Set ZAGOLOVOK = ActiveDocument.Range(Start:=0, End:=0)` | 6 |
| `With ZAGOLOVOK` | Начало структуры With |
| `.InsertBefore "Протоколы"` | 7 |
| `.Font.Name = "Arial"` | 8 |
| `.Font.Size = 24` | 8 |
| `.InsertParagraphAfter` | 9 |
| `End With` | Конец структуры With |
| `End Sub` | Конец |




ключается в том, чтобы программно определить номер абзаца, в котором будет происходить подсчет букв *а*; подсчитать их количество; в конце документа (или после абзаца, в котором происходил подсчет) вывести сообщение о количестве букв.

*Порядок работы.*

1. Вызвать панель элементов управления “Вид | Панели инструментов | Элементы управления”.

2. Подготовить кнопку на рабочем листе ниже текста “CommandButton2”.

3. Переименовать кнопку в “Количество букв *а* в абзаце”.

4. Придать кнопке формат обтекания по контуру, чтобы ее можно было перемещать по листу.

5. Выбрать в контекстно-зависимом меню этого объекта пункт “Исходный текст” или два раза щелкнуть по кнопке (мы попадем в редактор в режим написания программы).

Алгоритм выполнения задания следующий:

1. Объявить переменные:

a) `b` — строковая переменная для размещения в ней выбранного абзаца в документе;

b) `k` — переменная для хранения количества абзацев в документе;

c) `dl` — переменная для хранения длины абзаца;

d) `Text` — переменная для формирования текста-ответа о количестве символов *a* в абзаце;

e) `Nab` — переменная для ввода номера абзаца, в котором будет происходить подсчет;

f) `I` — переменная для организации цикла;

g) `kol` — переменная для подсчета символов *a*;

h) `REZULTAT` — переменная, которая определит область для вывода заголовка.

2. Ввести номер абзаца, используя диалоговое окно.

3. Определить количество абзацев в документе.

4. Проверить существование введенного номера абзаца и выдать сообщение об ошибке, если такого нет.

5. В переменную `b` занести текст всего абзаца.

6. Определить длину абзаца.

7. Организовать цикл по нахождению символа *a*.

8. Сформировать текст вывода сообщения.

9. Вставить дополнительный абзац после последнего абзаца документа для вывода сообщения.

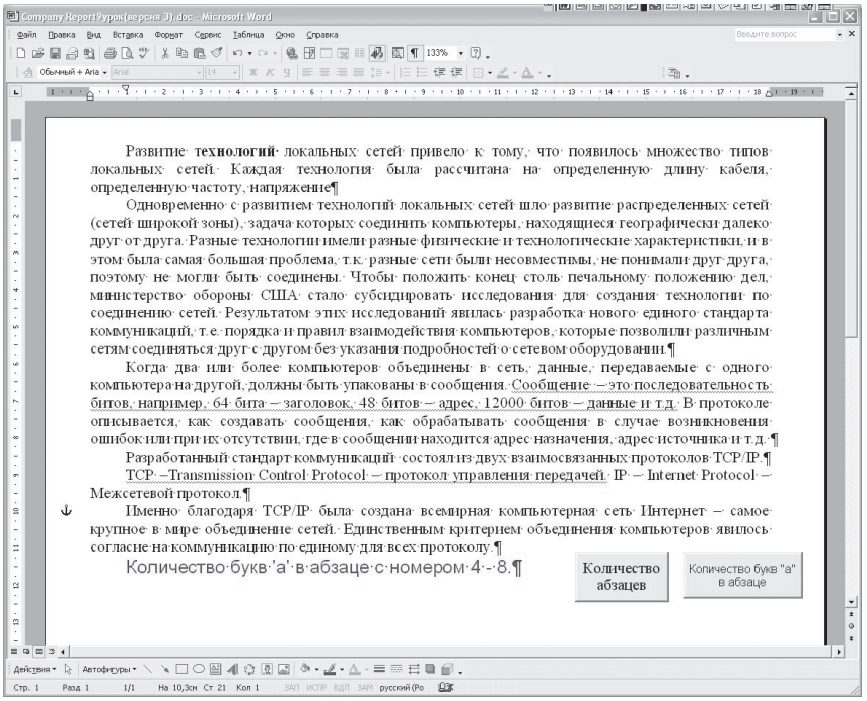
10. Вывести текст-сообщение в подготовленный абзац.

11. Установить определенный шрифт, размер и цвет текста-сообщения.

*Текст программы:*

| **Строка программы** | **Пункт алгоритма** |
|---|---|
| `Private Sub CommandButton2_Click()` | Начало программы |
| `Dim b As String` | 1a |
| `Dim k As Integer` | 1b |
| `Dim dl As Long` | 1c |
| `Dim Text As String` | 1d |
| `Dim Nab As Integer` | 1e |
| `Dim i As Long` | 1f |
| `Dim REZULTAT As Range` | 1g |
| `kol = 0` | |
| `Nab = InputBox("Введите номер абзаца", "Подсчитаем буквы a")` | 2 |
| `k = ActiveDocument.Paragraphs.Count` | 3 |
| `If Nab > k Then` | 4 |
| `MsgBox "В тексте нет такого абзаца", 48, "Предупреждение"` | 4 |
| `End` | 4 |
| `End If` | 4 |
| `b = ActiveDocument.Paragraphs(Nab).Range` | 5 |
| `dl = Len(b)` | 6 |
| `For i = 1 To dl` | 7 |
| `If Mid(b, i, 1) = "a" Or Mid(b, i, 1) = "A" Then kol = kol + 1` | 7 |
| `Next i` | 7 |
| `MsgBox kol` | Отладочная печать |
| `Text = "Количество букв a в абзаце с номером " & Nab & " - " & kol & "."` | 8 |
| `ActiveDocument.Paragraphs(k).Range.InsertParagraphAfter` | 9 |
| `Set REZULTAT = ActiveDocument.Paragraphs(k + 1).Range` | |
| `With REZULTAT` | |
| `.InsertBefore Text` | 10 |
| `.Font.Name = "Arial"` | 11 |
| `.Font.Size = 14` | 11 |
| `.Font.ColorIndex = wdDarkRed` | 11 |
| `End With` | |
| `End Sub` | Конец программы |

Результат выполнения представлен на рисунке.




b) `kol` — переменная для хранения количества предложений в абзаце;

c) `i` — переменная для организации цикла;

d) `Mas()` — массив для занесения количества предложений в абзацах;

e) `Otvet` — переменная для формирования текста-ответа с результатами;

f) `Max` — переменная для хранения максимального количества предложений в абзаце;

g) `ind` — переменная для хранения номера абзаца с максимальным количеством предложений;

h) `REZULTAT` — переменная, которая определит область абзаца, к которому надо применить элементы форматирования.

При объявлении массива нам неизвестно количество абзацев в документе. Поэтому объявление массива производим в два этапа: 1-й этап — объявление без размера `Dim Mas() As Integer`; 2-й этап — после определения количества абзацев в документе переопределяем размерность массива `ReDim Mas(k) As Integer`. Такие массивы называются динамическими.

Кроме этого, чтобы индекс массива начинался с 1, а не с 0 (для удобства работы), в редакторе в начале листа с процедурой укажите команду “Option Base 1”.

2. Определить количество абзацев в документе.

3. Переопределить размерность массива.

4. Организовать цикл по определению количества предложений в абзацах с занесением их размерности в массив и сохранением порядкового номера в массиве, что соответствует номеру абзаца.

5. Организовать цикл по нахождению наибольшего количества предложений в абзаце (работа со сформированным массивом) по стандартному алгоритму.

6. Сформировать текст вывода сообщения.

7. Вывести в диалоговом окне сообщений результат.

8. Определить область абзаца для форматирования.

9. Установить определенный шрифт, размер и цвет текста-абзаца с максимальным количеством предложений.

## Практическая работа № 9–4 “Предложения в абзаце”

Эта практическая работа выполняется с тем же документом, что и работа № 9—2 и 9—3 (или с любым текстом, который состоит из нескольких абзацев). Задача заключается в том, чтобы программно определить номер абзаца, в котором наибольшее количество предложений; выделить сообщение о номере абзаца и о количестве предложений, используя диалоговое окно; выделить этот абзац другим цветом.

*Порядок работы.*

1. Вызвать панель элементов управления “Вид | Панели инструментов | Элементы управления”.

2. Подготовить кнопку “CommandButton3” на рабочем листе ниже текста.

3. Переименовать кнопку в “Предложения в абзаце”.

4. Придать ей формат обтекания по контуру, чтобы кнопку можно было перемещать по листу.

5. Выбрать в контекстно-зависимом меню этого объекта пункт “Исходный текст” или два раза щелкнуть по кнопке (попадаем в редактор в режим написания программы).

*Алгоритм выполнения задания следующий:*

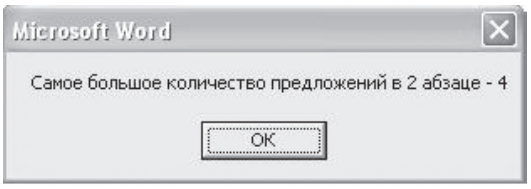
1. Объявить переменные:

a) `k` — переменная для хранения количества абзацев в документе;

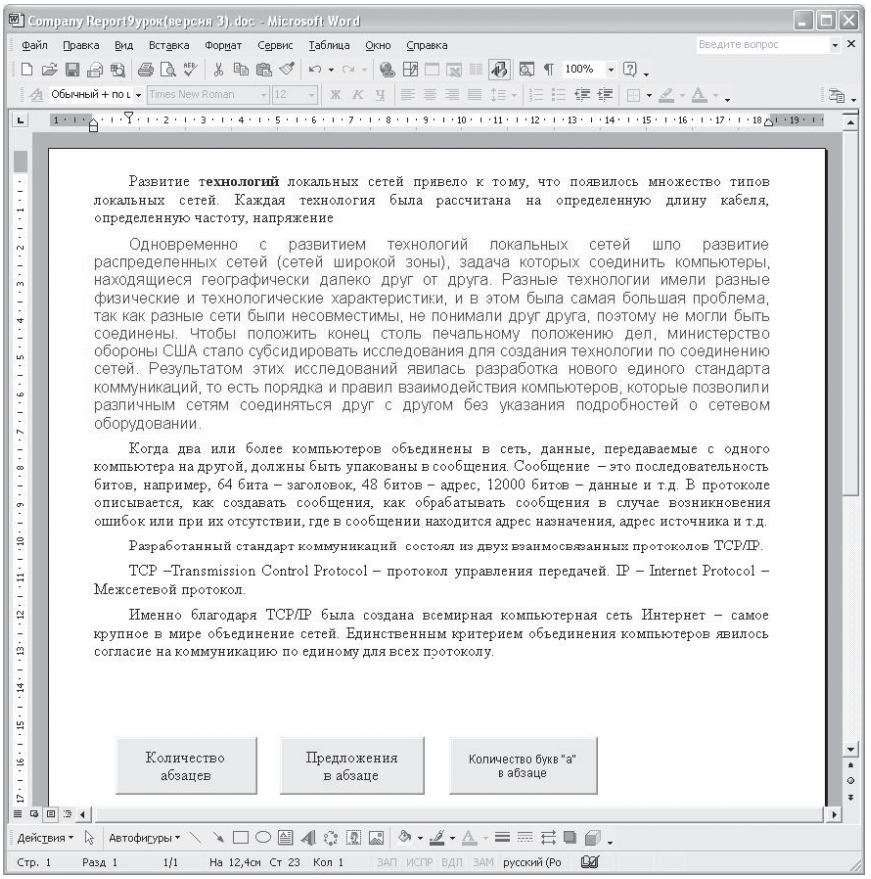
*Текст программы:*

| **Строка программы** | **Пункт алгоритма** |
|---|---|
| `Private Sub CommandButton3_Click()` | Начало программы |
| `Dim k As Integer` | 1a |
| `Dim kol As Integer` | 1b |
| `Dim i As Integer` | 1c |
| `Dim Mas() As Integer` | 1d |
| `Dim otvet As String` | 1e |
| `Dim Max As Integer` | 1f |
| `Dim ind As Byte` | 1g |
| `Dim REZULTAT As Range` | 1h |
| `kol = 0: k = 0` | Обнуление переменных |
| `k = ActiveDocument.Paragraphs.Count` | 2 |
| `ReDim Mas(k) As Integer` | 3 |
| `Rem Начало программы` | Комментарий |
| `For i = 1 To k` | 4 |
| `kol = ActiveDocument.Paragraphs(i).Range.Sentences.Count` | 4 |
| `Mas(i) = kol` | 4 |
| `Next i` | 4 |
| `Max = Mas(1)` | Подготовительная часть перед циклом |
| `ind = 1` | |
| `For i = 2 To k` | Начало цикла |
| `If Mas(i) > Max Then` | 5 |
| `Max = Mas(i)` | 5 |
| `ind = i` | 5 |
| `End If` | 5 |
| `Next i` | Конец цикла |
| `Rem Вывод результатов` | Комментарий |
| `otvet = "Самое большое количество предложений в " & ind & " абзаце - " & Max` | 6 |
| `MsgBox otvet` | 7 |
| `Set REZULTAT = ActiveDocument.Paragraphs(ind).Range` | 8 |
| `With REZULTAT` | Начало структуры With |
| `.Font.Name = "Arial"` | 9 |
| `.Font.Size = 12` | 9 |
| `.Font.ColorIndex = wdDarkRed` | 9 |
| `End With` | Конец структуры With |
| `End Sub` | Конец программы |

Диалоговое окно с результативным сообщением будет выглядеть так:




Результат выполнения задания представлен на рисунке.




## Практическая работа № 9–5 “Слова-палиндромы”

Задача заключается в том, чтобы программно определить, есть ли в тексте слова-палиндромы и выделить эти слова другим цветом.

*Порядок работы.*

1. Подготовить текст (можно придумать рассказ или стихотворение со словами-палиндромами ШАЛАШ, КАЗАК, НАГАН, МАДАМ и т.д.).
2. Вызвать панель элементов управления “Вид | Панели инструментов | Элементы управления”.
3. Подготовить кнопку “CommandButton1” на рабочем листе ниже текста.
4. Переименовать кнопку в “Палиндромы”.
5. Придать созданной кнопке формат обтекания по контуру, чтобы ее можно было перемещать по листу.
6. Выбрать в контекстно-зависимом меню этого объекта пункт “Исходный текст” или два раза щелкнуть по кнопке.

*Алгоритм выполнения задания следующий:*

1. Объявить переменные:

a) `Slovo` — переменная для последовательного хранения выделенных слов;

b) `L` — переменная для хранения длины слова;

c) `M` — переменная, определяющая половину длины выделенного слова;

d) `K` — переменная для формирования 1-й половины слова;

e) `D` — переменная для формирования 2-й половины слова;

f) `Kol` — переменная для хранения количества абзацев в документе.

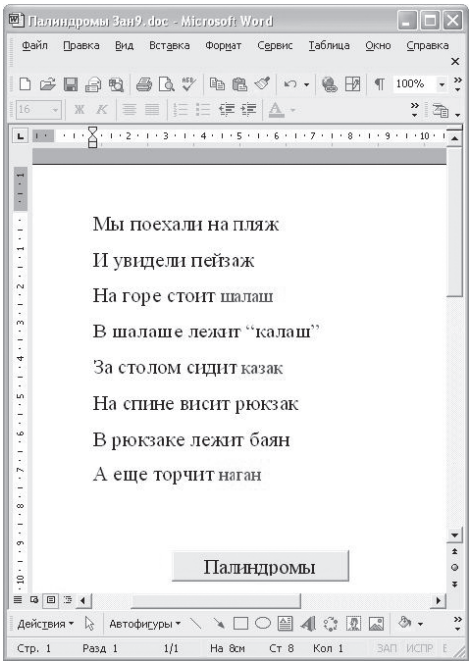
2. Определить количество абзацев в документе.
3. Определить область, в которой надо искать слова-палиндромы, — это все абзацы текста.
4. Организовать цикл по выделению слов в области определения.
5. Присвоить выделенное слово строковой переменой.
6. Определить длину выделенного слова.
7. Проверить длину слова. Если она равна 1, то слово проверять нет смысла.
8. Организовать цикл, чтобы определить, является ли слово палиндромом.
9. Изменить цвет букв у слова-палиндрома.

*Текст программы:*

| Строка программы | Пункт алгоритма |
|---|---|
| `Private Sub CommandButton1_Click()` | Начало программы |
| `Dim slovo As String` | 1a |
| `Dim L As Byte` | 1b |
| `Dim M As Byte` | 1c |

| | |
|---|---|
| `Dim K As String` | 1d |
| `Dim D As String` | 1e |
| `Dim kol as byte` | 1f |
| `kol = ActiveDocument.Paragraphs.Count` | 2 |
| `Set myRange = ActiveDocument.Range (ActiveDocument.Paragraphs(1).Range.Start, ActiveDocument.Paragraphs(kol).Range.End)` | 3 |
| `For Each aword In myRange.Words` | 4 (Начало внешнего цикла) |
| `slovo = aword.Text` | 5 |
| `L = Len(slovo)` | 6 |
| `If L <= 1 Then GoTo M1` | 7 |
| `M = Int(L / 2)` | Подготовка к проверке на палиндром |
| `For I = 1 To M` | 8 (Начало вложенного цикла) |
| `K = Mid$(slovo, I, 1)` | 8 |
| `D = Mid$(slovo, L - I + 1, 1)` | 8 |
| `If K <> D Then GoTo M1` | 8 |
| `Next I` | 8 (Конец внутреннего цикла) |
| `aword.Font.Size = 14` | 9 |
| `aword.Font.ColorIndex = wdDarkBlue` | 9 |
| `M1: Next aword` | 4 (Конец внешнего цикла) |
| `End Sub` | Конец программы |

В результате выполнения программы слова-палиндромы будут выделены синим цветом.



## Занятие № 10. VBA в приложении Power Point

В приложении Power Point с помощью VBA можно автоматизировать множество задач. Рассмотрим применение пользовательских форм для проектирования тестов и различных задач в PowerPoint. Кроме того, имеется возможность не только ввести информацию, но и использовать элементы управления. Для этого требуется следовать двум простым правилам:

1. Макросы необходимо назначить кнопкам панелей инструментов.

2. Элементы управления необходимо разместить в диалоговых окнах.

## Практическая работа № 10–1 “Умеешь ли ты считать”

Создайте презентацию, с помощью которой можно проверить и оценить навыки устного счета.

Порядок работы:

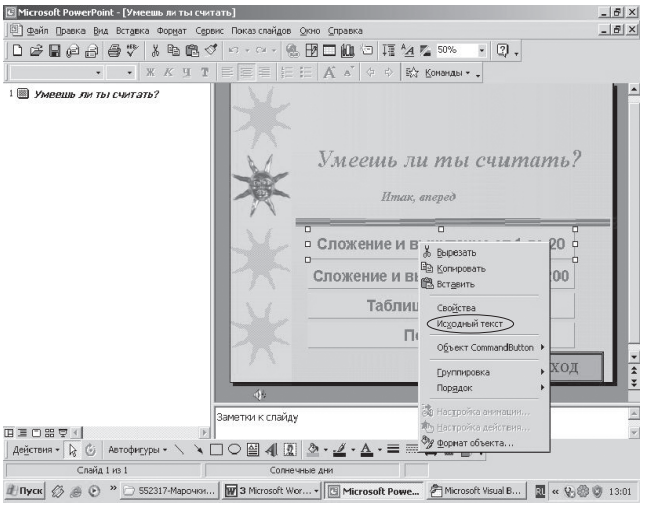
1. Открыть Power Point и создать титульный слайд; оформить на нем пять кнопок:

• Посредством Панели элементов:

| Сложение и вычитание от 1 до 20 |
|---|
| Сложение и вычитание от 1 до 200 |
| Таблица умножения |
| Посчитай!!! |

• Через панель рисования:

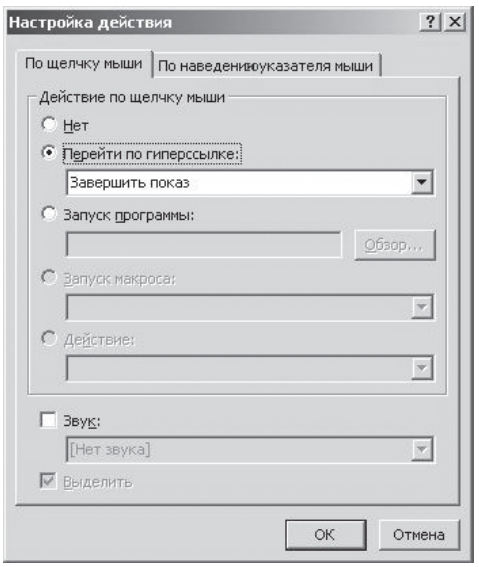

2. Записать программы вызова пользовательских форм (через контекстное меню “Исходный текст”, выделив тематические кнопки).



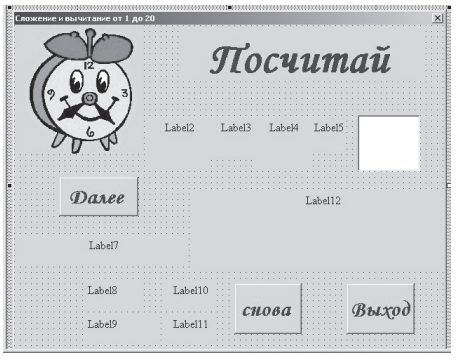

*Тексты программ кнопок:*

```
Private Sub CommandButton1_Click()
UserForm1.Show
End Sub

Private Sub CommandButton2_Click()
UserForm4.Show
End Sub

Private Sub CommandButton3_Click()
UserForm3.Show
End Sub

Private Sub CommandButton4_Click()
UserForm2.Show
End Sub
```

3. Настроить кнопку “Выход” на завершение показа.




4. Спроектировать в редакторе Visual Basic пользовательскую форму UserForm1 “Сложение и вычитание от 1 до 10”.




При проектировании формы *Свойство Caption* для меток *Label* сделать пустым; отформатировать все элементы.

Далее следует описать событийные процедуры при нажатии на кнопки.

Объявление глобальных переменных

```
Public a As Integer
Public b As Integer
Public R As Integer
Public v As Integer
Public n As Integer
Public f As Integer
```

Программа для кнопки "Далее"

```
Private Sub CommandButton1_Click()
' a — переменная для первого числа
' b — переменная для второго числа
' R — переменная для получения результата
' v, n — переменные для подсчета верных
' и неверных ответов
' f — переменная оператора выбора
 f = f + 1
Select Case f
 Case 1
' Присваиваем значения переменным
' a и b через RND в интервале (0;9)
b = Int(10 * Rnd())
a = Int(10 * Rnd())
' Формируем в надписях - или + и =
Label3.Caption = "-"
Label5.Caption = "="
```

Определяем большее из чисел, и это — первое число, меньшее — второе число:

```
If a > b Then
Label2.Caption = a
Label4.Caption = b
Else
Label2.Caption = b
Label4.Caption = a
End If
' Вычисляем результат
R = Abs(a - b)
Case 2
```

Сравниваем результат и вводимый ответ, подсчитываем количество

```
If Val(R) = Val(TextBox1) Then
v = v + 1
Label12.Caption = "Верно"
Else
n = n + 1
Label12.Caption = "Неверно"
End If
Case 3
CLS
Label12.Caption = ""
b = Int(10 * Rnd())
a = Int(10 * Rnd())
Label2.Caption = a
Label4.Caption = b
Label3.Caption = "+"
Label5.Caption = "="
R = a + b
```

Написать аналогичные программы для следующих 9 примеров.

```
' Проверка результата в примере 10
Case 20
If Val(R) = Val(TextBox1) Then
v = v + 1
Label12.Caption = "Верно"
Else
n = n + 1
Label12.Caption = "Неверно"
End If
' Вводим в надписи количество верных
и неверных ответов, даем напутствия
Label7.Caption = "Ваш результат"
Label8.Caption = "Верно"
Label10.Caption = Str(v)
Label9.Caption = "Неверно"
Label11.Caption = Str(n)
If v = 10 Then Label12.Caption =
" Молодец!!!" Else Label12.Caption =
"Еще поработай над счетом!!!"
Case 21
UserForm1.Hide
End Select
End Sub
```

Программа кнопки "Снова"

```
Private Sub CommandButton2_Click()
```

Набираем программный код для кнопки снова (для этого щелкнем по кнопке)

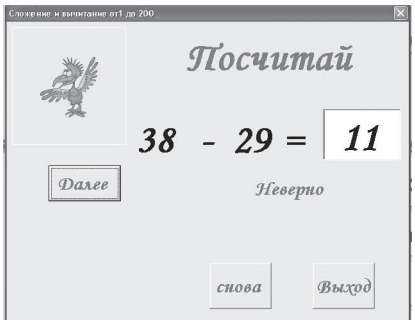
```
CLS
'Обнуляем переменные, подсчитывающие
результат
n = 0
v = 0
'В надписях стираем информацию
Label10.Caption = ""
Label11.Caption = ""
Label12.Caption = ""
Label2.Caption = ""
Label4.Caption = ""
Label3.Caption = ""
Label5.Caption = ""
Label7.Caption = ""
Label8.Caption = ""
Label9.Caption = ""
f = 0
End Sub

Sub CLS()
TextBox1.Text = ""
End Sub
```

В рабочем состоянии форма выглядит так:

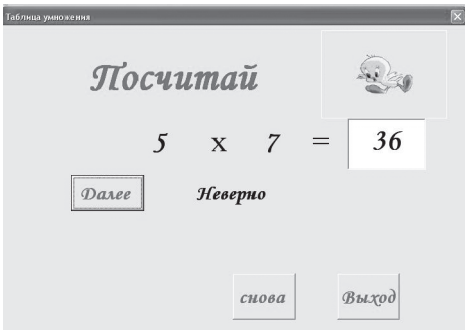

5. Спроектировать пользовательские формы UserForm2, UserForm3, UserForm4.

6. Записать программы в кнопках.

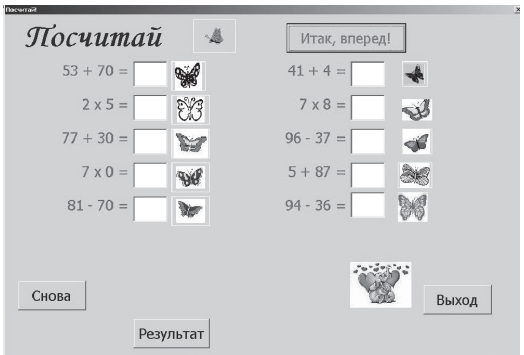
Вид формы UserForm2 "Сложение и вычитание от 1 до 200" может быть таким:




Примерный вид формы UserForm3 "Таблица умножения":




А вот такой может быть форма UserForm4 "Посчитай":



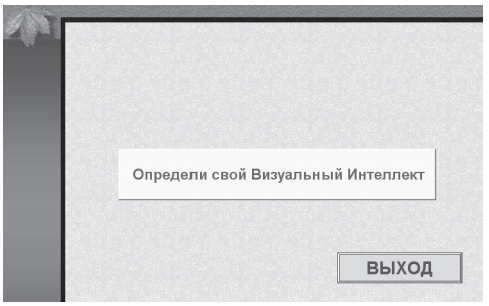

7. Проверить работоспособность проекта и сохранить работу.

## Практическая работа № 10–2 "Тест на Визуальный Интеллект"

Создайте презентацию, с помощью которой можно проверить и оценить визуальный интеллект.

Порядок работы:

1. Открыть Power Point, создать титульный слайд.



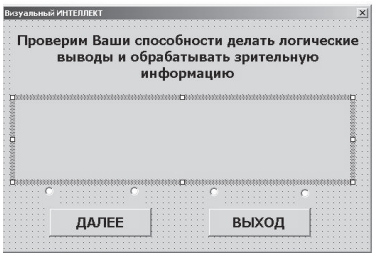

2. Создать посредством панели элементов кнопку "Определи свой Визуальный Интеллект" и записать (через контекстное меню "Исходный текст") программу вызова пользовательской формы UserForm1:

```
Private Sub CommandButton1_Click()
UserForm1.Show
End Sub
```

3. Создать через панель рисования кнопку "Выход" и настроить ее на завершение показа.

4. Спроектировать в редакторе Visual Basic пользовательскую форму со следующими элементами управления:

| Элемент управления | Назначение |
|---|---|
| Label1 | Надпись |
| OptionButton1<br>OptionButton2<br>OptionButton3<br>OptionButton4 | Выбор вариантов ответа |
| CommandButton1<br>CommandButton2 | Кнопки |




Программа кнопки "Далее"

```
Public i As Integer
Public x As Integer
' Переменная x подсчитывает количество
правильных ответов
Private Sub CommandButton1_Click()
i = i + 1
```

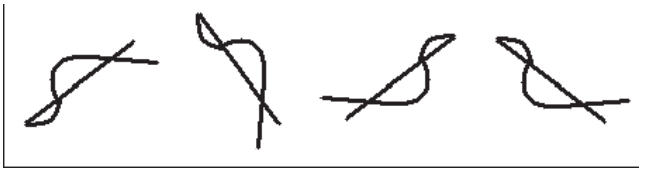
```
Select Case i
Case 1
' При первом нажатии на ДАЛЕЕ меняется
надпись и рисунок
Label1.Caption = "Какая из фигур не
соответствует общему ряду?"
UserForm1.Image1.Picture =
LoadPicture("C:\тест1.JPG")
Case 2
' При втором нажатии на ДАЛЕЕ
подсчитывается правильный ответ,
скидываются флажки в OptionButton
и меняется рисунок
If OptionButton4.Value = True
Then x = x + 1
OptionButton1.Value = False
OptionButton2.Value = False
OptionButton3.Value = False
OptionButton4.Value = False
UserForm1.Image1.Picture =
 LoadPicture("C:\тест2.jpg")
Case 3
' При третьем нажатии на ДАЛЕЕ
подсчитывается правильный ответ,
скидываются флажки в OptionButton
и меняется рисунок
If OptionButton1.Value = True Then x = x + 1
OptionButton1.Value = False
OptionButton2.Value = False
OptionButton3.Value = False
OptionButton4.Value = False
UserForm1.Image1.Picture =
LoadPicture("C:\тест3.jpg")
Case 4
' При четвертом нажатии на ДАЛЕЕ
подсчитывается правильный ответ,
скидываются флажки в OptionButton
и меняется рисунок
If OptionButton3.Value = True Then x = x + 1
OptionButton1.Value = False
OptionButton2.Value = False
OptionButton3.Value = False
OptionButton4.Value = False
UserForm1.Image1.Picture =
LoadPicture("C:\тест4.jpg")
Case 5
' При пятом нажатии на ДАЛЕЕ
подсчитывается правильный ответ,
скидываются флажки в OptionButton
и меняется рисунок
If OptionButton3.Value = True Then x = x + 1
OptionButton1.Value = False
OptionButton2.Value = False
OptionButton3.Value = False
OptionButton4.Value = False
UserForm1.Image1.Picture =
LoadPicture("C:\тест5.jpg")
Case 6
' При шестом нажатии на ДАЛЕЕ
подсчитывается правильный ответ,
скидываются флажки в OptionButton
и выдается результат
If OptionButton4.Value = True Then x = x + 1
OptionButton1.Value = False
OptionButton2.Value = False
OptionButton3.Value = False
OptionButton4.Value = False
Label1.Caption = "Ваша оценка " & x
Case 7
UserForm1.Hide
End Select
End Sub
```

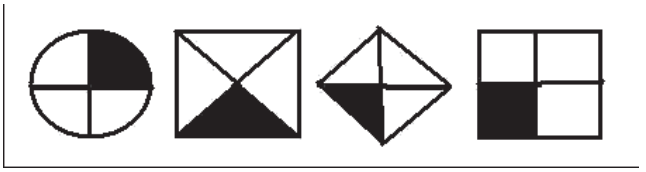
Программа кнопки “Выход”

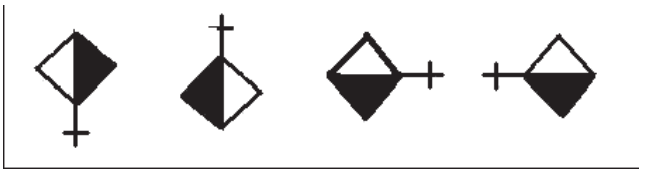
```
Private Sub CommandButton2_Click()
UserForm1.Hide
End Sub
```

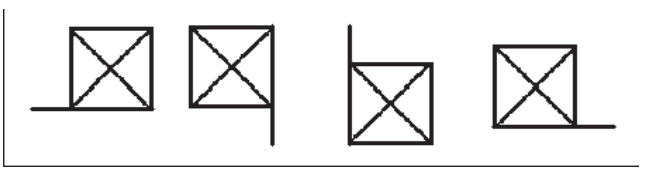
При нажатии на кнопку “Далее” форма будет выглядеть примерно так:


5. Выбрать рисунки-тесты.

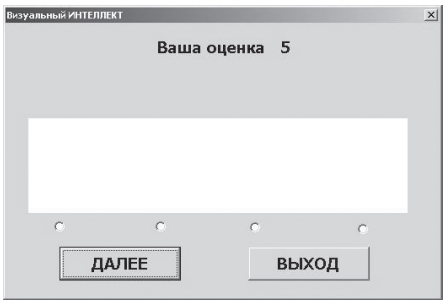








6. Вывести результат после прохождения всех пяти вопросов.




## Практическая работа № 10–3 “Твой соционический тип”

Создайте презентацию, с помощью которой можно определить соционический тип.

*Порядок работы:*

1. Спроектировать титульный слайд; предусмотреть кнопки вызова пользовательских форм для определения соционического типа: “экстраверт — интраверт”; “сенсорик — интуит”; “логик — этик”; “иррационал — рационал”; “выход”; “узнай свой социотип” (определяются все типы).


2. Спроектировать пользовательские формы для определения социотипа:

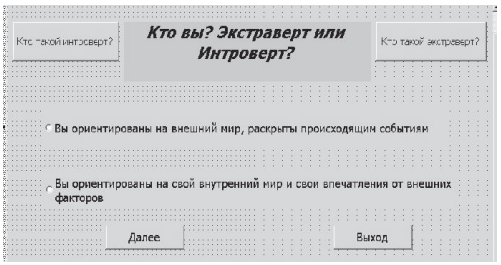
• разработать программы для кнопок перехода на пользовательские формы описания социотипов (справочные);

• создать “OptionButton” для выбора варианта ответа;

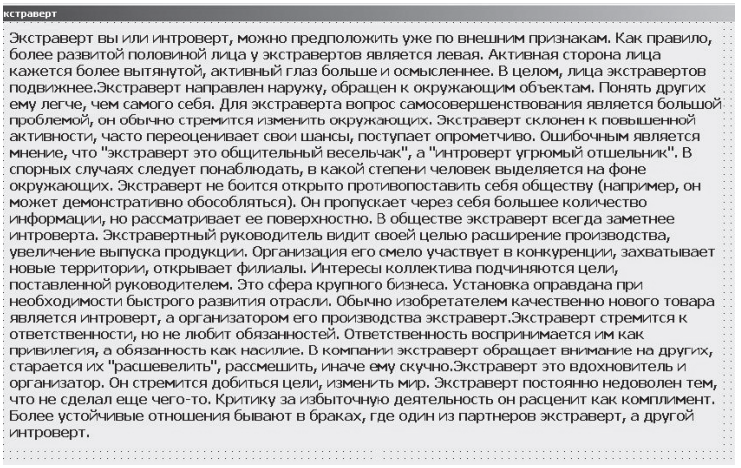
• создать кнопки “Далее” и “Выход”;

• предусмотреть переход на формы, которые содержат справочный материал;

• создать форму для определения социотипа;

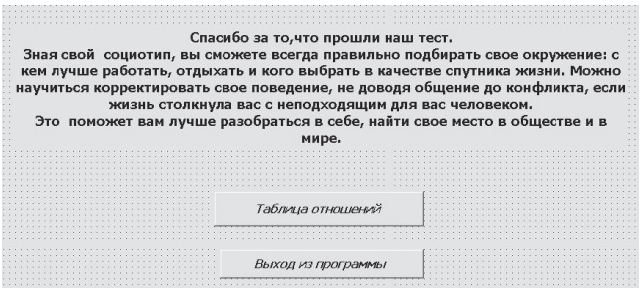



• создать форму (вызываемую при нажатии на клавишу “Кто такой экстраверт?”), которая будет содержать справочный материал;




• предусмотреть подсчет ответов при нажатии на кнопку “Далее”; при последнем нажатии должен выдаваться результат: “Вы — ЛОГИК” и т.д.;

• предусмотреть после прохождения всех вопросов и ответов вызов формы, из которой, в свою очередь, можно попасть в таблицу отношений социотипов или совсем выйти из вопросника.

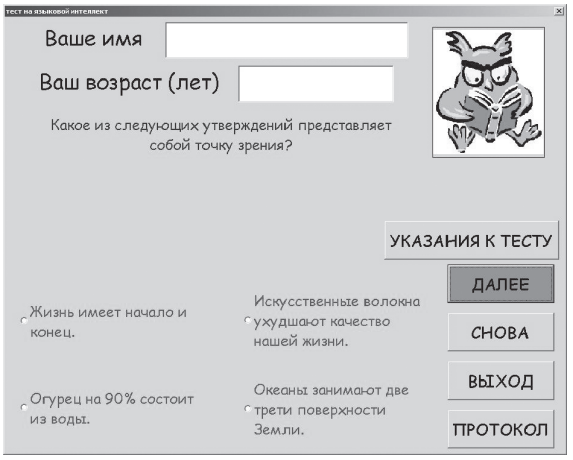



### Дополнительные задания

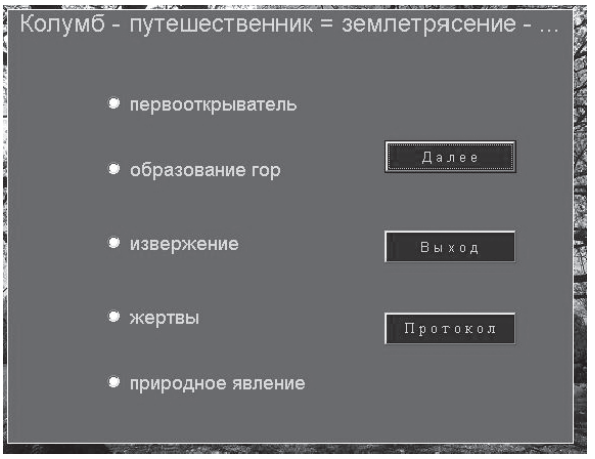
Подберите материал для создания теста-вопросника. Это может быть:

• Тест по предмету:

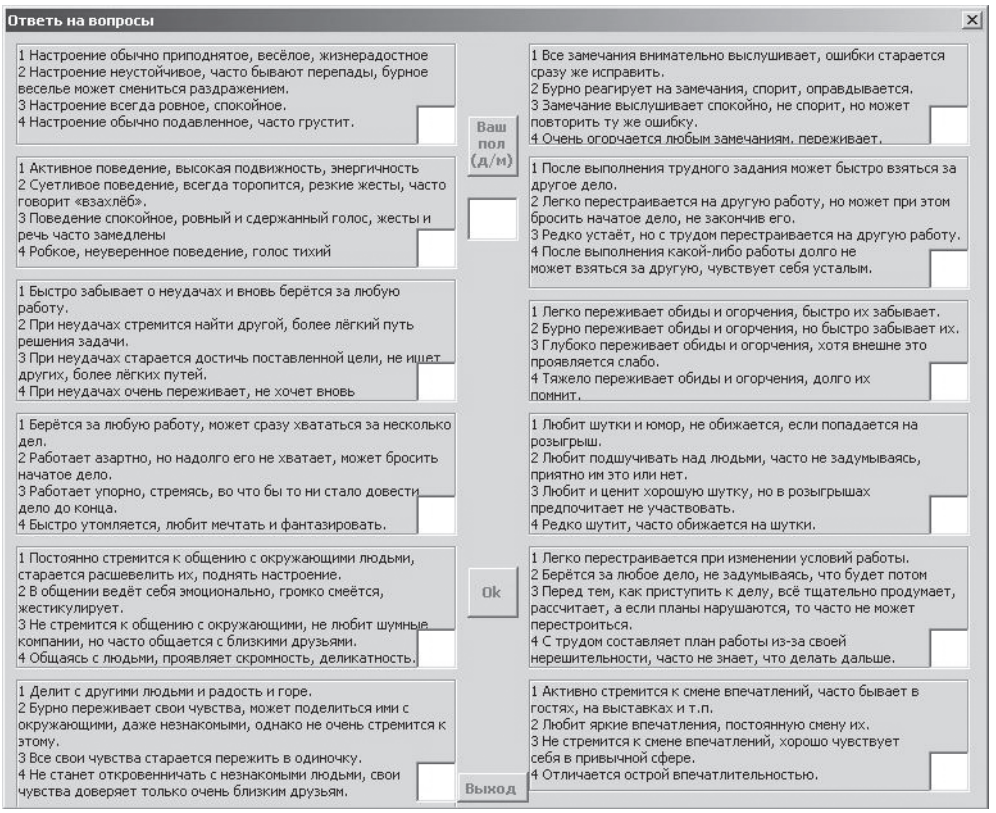
- Тест для определения IQ:




- Тест ШТУР (школьный тест умственного развития):




- Тест для определения темперамента:
- Тест профессий и т.д.




Надеемся, что вам был полезен изложенный материал. Желаем успехов!

# Практикум по информатике в среде ЛогоМиры

**А.Г. ЮДИНА,**
*Москва*

## Предисловие

Приглашаем вас познакомиться с компьютерной средой ЛогоМиры и с одним из языков программирования — Лого. Вы научитесь сами создавать программы — графические, игровые, диалоговые.

Язык Лого был разработан в конце 60-х гг. американским исследователем в области искусственного интеллекта С.Пейпертом. Название языка происходит от греческого "логос" — "слово, мысль, смысл, идея".

Во многих странах Лого признан лучшим языком для тех, кто начинает знакомиться с миром компьютеров, а также прекрасным средством для помощи в учебе (особенно по таким предметам, как математика и физика).

## ГЛАВА 1
## Графические команды. Простые циклы-повторения

### § 1. Перемещение черепашки

Запустив программу ЛогоМиры, вы сначала увидите заставку, а затем — окно с черепашкой в центре. Это рабочее поле. В самом верху окна — название файла. Вы еще файл не сохраняли (то есть не записывали на диск и не давали файлу имя), поэтому он, бедный, пока без имени — Безымянный.

В рабочем поле среды ЛогоМиры можно рисовать, помещать окна с текстами, кнопки и регуляторы, но для нас сейчас главное, что в этом поле будет двигаться черепашка, послушно выполняя написанные вами программы и оставляя при этом рисунки.

Слева от рабочего поля находится панель инструментов, а внизу — поле команд. Можно сказать, что это поле проб и ошибок. Вы будете записывать в это поле команды для черепашки: теперь в вашем распоряжении почти настоящий робот.

Самая первая команда — опустить перо: `PD` (***p**en **d**own*). Ее нужно выполнить, чтобы черепашка при перемещении оставляла за собой след — рисунок. Напечатайте `PD` в поле команд. Черепашка выполнит команду после того, как вы нажмете ввод — клавишу Enter.

Для перемещения черепашки вперед и назад служат команды `FD` и `BK` (по-английски *forward* — вперед, *back* — назад); после команды нужно указать, на сколько шагов должна продвинуться черепашка. Например, `FD 50` или `BK 70` (шаг у черепашки очень маленький). Между командой и числом необходим пробел. Команды можно набирать и строчными, и прописными буквами (например, `fd 100` или `FD 100`).

Кроме перемещения вперед-назад, черепашка способна поворачиваться вокруг своей оси на указанное число градусов. Для поворота по часовой стрелке служит команда `RT`; против часовой стрелки — команда `LT` (от слов *right* и *left* — направо и налево). Например, `RT 90` или `LT 60`.

Команда `CG` (***c**lear **g**raphics* — сотри рисунки) позволит очистить экран и вернуть черепашку в исходное положение.

**Задания**

*1.1. Получите на экране квадрат.*

*1.2. Расчертите экран в косую линеечку.*

*1.3. Измерьте длину и ширину экрана (в черепашьих шагах).*

*1.4. Начертите треугольник.*

Если черепашка уже доползла до границы экрана и должна двигаться дальше, то она не исчезнет, а появится с другой стороны (как будто края экрана склеены).

Прежде чем закончить работу, не забудьте сохранить ваш файл.

### § 2. Последовательность команд — программа для черепашки

Можно управлять черепашкой, нажимая ввод после каждой команды. Если же записать подряд несколько команд (отделяя их друг от друга пробелом) и только после этого нажать ввод, то черепашка быстро выполнит команды одну за другой.

Представьте себе, что черепашка должна выполнить следующую программу:

```
FD 40 LT 90 FD 40 LT 90 FD 40
```

Какой рисунок получится на экране? Попробуйте сначала нарисовать картинку на бумаге, а потом проверьте результат на компьютере.

Дополните эту последовательность команд так, чтобы на экране вышел полный квадрат.

Если присмотреться к получившейся программе, то можно заметить, что в ней четыре раза повторяется пара команд:

```
FD 40 LT 90
```

Новая команда `REPEAT` (повтори) избавит нас от скучных повторов:

```
REPEAT  сколько раз  [ что повторять ]
```

Например, чтобы нарисовать квадрат, можно дать команду:

```
REPEAT 4 [FD 40 LT 90]
```

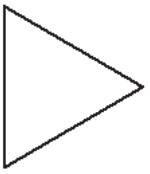
Попробуйте изменять набор команд в квадратных скобках и число повторений.

Приемы редактирования командной последовательности: стереть ошибку можно клавишей Del (если курсор стоит на удаляемом символе) или клавишей BackSpace (тогда курсор нужно поставить справа от ошибки); чтобы сделать вставку, достаточно поставить курсор в нужное место и набрать пропущенное.

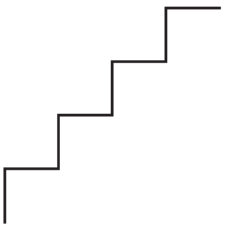
**Задания**

*Напишите программы для следующих рисунков, используя команду повтора.*

*2.1. Равносторонний треугольник*



*2.2. Лесенка*



## § 3. Движение черепашки без следа

Черепашка может двигаться и без следа, это необходимо для получения на экране нескольких отдельных изображений. Для перемещения без следа нужна новая команда:

PU (*pen up*) — подними перо (черепашка при движении не будет оставлять след).

Для того чтобы черепашка снова могла рисовать, надо дать команду PD (*pen down*) — опусти перо.

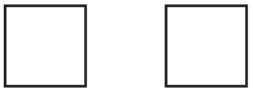
**Задания**

*Напишите программы для следующих рисунков.*

*3.1. Пунктир*

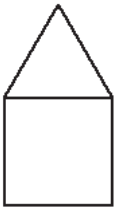


*3.2. Квадраты*



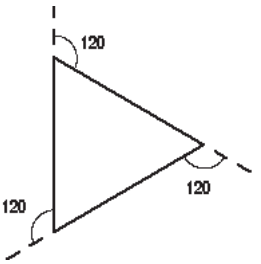
*3.3. Пила*

*3.4. Домик*



## § 4. Правильные многоугольники

Все внутренние углы в равностороннем треугольнике равны 60°.




Чтобы нарисовать треугольник и вернуться в исходное положение, черепашка должна три раза повернуться на угол, смежный с внутренним, т.е. на 120°. Например:

```
REPEAT 3 [FD 60 RT 120]
```

Вы уже умеете с помощью черепашки рисовать на экране равносторонний треугольник и равносторонний четырехугольник (квадрат). Какая геометрическая фигура должна быть следующей в этом ряду?


(правильный треугольник, правильный четырехугольник, ...)

Дальше должен быть правильный пятиугольник. А как написать для него программу?

Так можно нарисовать правильный треугольник:

```
REPEAT 3 [FD 40 RT 120]
```

Черепашка трижды повернулась на 120° и вернулась в исходную точку.

3 × 120° = 360° — полный оборот.

Это программа для правильного четырехугольника:

```
REPEAT 4 [FD 40 RT 90]
```

4 × 90° = 360° — полный оборот.

Видимо, чтобы нарисовать правильный пятиугольник, черепашка должна пять раз выполнить пару действий: пройти некоторое расстояние и повернуться на пятую часть полного оборота (360° / 5 = 72°):

```
REPEAT 5 [FD 40 RT 72]
```

Получился правильный пятиугольник.

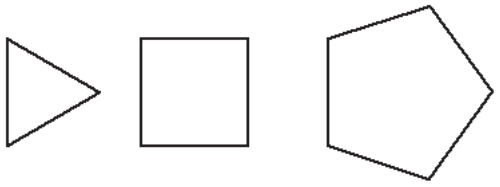
Арифметические действия можно поручить выполнять компьютеру; знаки арифметических операций в LogoWriter нужно отделять с двух сторон пробелами: например, `RT 360 / 5`.

**Задания**

*4.1. Рассуждая аналогичным образом, напишите программы для рисования правильных многоугольников с числом сторон, равным 6, 7, 9, 12, 20, 100, 360. Почему, если мы хотим, чтобы фигуры были примерно одного размера, приходится изменять длину стороны?*

*Напишите программы для рисования следующих картинок.*

*4.2. Фигуры*



*4.3. Треугольники*

*4.4. Знак*

## ГЛАВА 2
## Выделение вспомогательных алгоритмов

### § 5. Описание процедур

При выполнении заданий к предыдущему параграфу вам пришлось несколько раз переписывать одну и ту же последовательность команд для рисования треугольника. Оказывается, программу можно записать один раз (как в записную книжку) в специальном месте — на обратной стороне листа. Записанной программе нужно дать имя и затем, сколько бы раз она ни потребовалась, достаточно вызвать ее по имени. Таким образом, мы как бы обучаем черепашку новым командам. Этот процесс называется описанием процедур.

Например:

```
ЭТО КВАДРАТ
REPEAT 4 [FD 40 RT 90]
END
```

Имена (названия) процедур должны отличаться от имен других процедур и от команд, чтобы не было путаницы. Записав процедуру на изнанке листа, вы можете переворачивать лист и использовать новую, описанную вами команду наравне с теми, которые черепашка "знает" с самого начала.

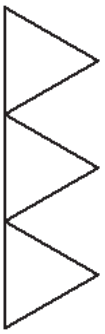
**Задания**

*5.1. Опишите процедуру рисования правильного треугольника.*

*5.2. Используя эту новую команду, нарисуйте на экране 5—6 треугольников, расположенных в разных местах.*

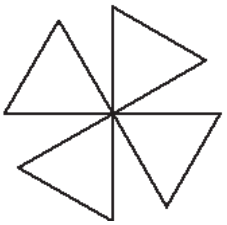
*Для рисования следующих картинок напишите программы с использованием процедуры.*

*5.3. Флаги.*



*5.4. Треугольники.*

*5.5. Напишите программу, которая рисует цветок с четырьмя треугольными лепестками.*



*5.6. Опишите процедуры, рисующие квадратный, пятиугольный, круглый лепесток.*

*5.7. Напишите программы для рисования цветов с разным количеством и различной формой лепестков.*

### § 6. Команды работы с цветом

До сих пор черепашка рисовала белым по черному. Можно изменить цвет черепашки, и тогда она будет рисовать цветные линии. Чтобы увидеть палитру цветов, откройте в нижнем окне вместо Поля команд Графический редактор (нажмите на кнопку с кисточкой).

Кроме черного и белого цвета, ЛогоМиры позволяют использовать 14 цветов, у каждого из них есть 10 оттенков. Каждый из них имеет свой номер.

Обратите внимание на стрелки с правой стороны палитры. Если щелкнуть на верхнюю стрелку, цвет станет темнее, если щелкнуть на нижнюю стрелку — светлее.

Укажите мышкой на нужный вам цвет, и под стрелкой появится окно с номером этого цвета. Черный цвет имеет номер 9, белый — 0. Оттенки одного цвета обозначаются числами одного десятка. Например, все красные цвета имеют номера от 10 до 19, а все синие — от 100 до 109. А номера с 1-го по 8-й — это оттенки серого.

Чтобы изменить цвет черепашки (и цвет ее пера), нужно дать команду SETC (***set color*** — установи цвет) и через пробел указать номер цвета. Например, после выполнения команды SETC 15 черепашка станет красной и будет оставлять такой же след.

Используя команду установки цвета, можно получить на экране разноцветные контуры и линии.

Для закраски частей экрана служит команда FILL (заполни). Черепашка, получив эту команду, заливает экран той же краской, какой покрашена сама (если перо черепашки опущено). Краска будет растекаться, пока не встретит препятствие — какой-либо замкнутый контур.

Чтобы черепашка закрасила контур, необходимо выполнить следующие действия:

а) поднять перо;

б) переместить черепашку внутрь контура;

в) опустить перо;

г) дать команду FILL.

**Задания**

*6.1. Начертите несколько разноцветных геометрических фигур.*

*6.2. Начертите и закрасьте окружность.*

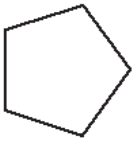
*6.3. В полученном круге нарисуйте треугольник и закрасьте его другим цветом.*

## § 7. Выделение составных частей в задаче

Возможность обучать черепашку новым командам (записывая процедуры на изнанке листа) очень полезна при решении достаточно сложных задач.

Например, чтобы нарисовать цветок с четырьмя пятиугольными лепестками, можно сначала создать процедуру рисования одного лепестка:

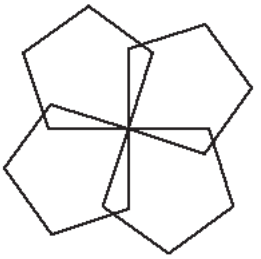
```
ЭТО* ЛЕПЕСТОК
REPEAT 5 [FD 60 RT 72]
END
```



— а затем уже записать, как получается цветок (лепесток — поворот, лепесток — поворот...):

```
ЭТО ЦВЕТОК
REPEAT 4 [ЛЕПЕСТОК RT 90]
END
```

ЦВЕТОК — это процедура второго уровня, в ней используется ранее описанная процедура.



Можно, конечно, написать программу рисования цветка и без предварительного описания лепестка:

```
REPEAT 5 [FD 60 RT 72] RT 90
REPEAT 5 [FD 60 RT 72] RT 90
REPEAT 5 [FD 60 RT 72] RT 90
REPEAT 5 [FD 60 RT 72] RT 90
```

Или короче:

```
REPEAT 4 [REPEAT 5 [FD 60 RT 72] RT 90]
```

Результат получится тот же, но запись процедуры ЦВЕТОК выглядит более понятно, ошибки в таких записях отыскивать и исправлять легче.

Еще одно преимущество заключается в том, что процедуру ЦВЕТОК можно использовать и для других цветков — стоит лишь изменить форму лепестка.

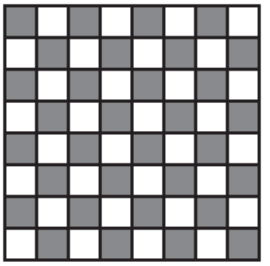
Чем сложнее задача, тем труднее добиться правильного результата, предварительно не планируя работу.

При обдумывании решения задачи надо стараться разбить ее на составные части, которые можно было бы описать как отдельные процедуры. В этих более простых задачах тоже могут выделиться составные части и т.д.

Таким образом, общее решение получится как бы выстроенным из нескольких уровней процедур.

Для сравнения: дом можно построить из привезенных на стройку готовых блоков, которые, в свою очередь, собрали на комбинате из более мелких деталей. Так получается намного быстрее, чем, положим, при строительстве кирпичного дома.

Пусть черепашка должна нарисовать шахматную доску — 64 квадратные клеточки, половина из которых закрашена.



* Вместо русского слова ЭТО можно использовать английское TO.

При решении этой достаточно сложной задачи поможет разбиение ее на подзадачи и описание более простых частей как процедур.

Предположим, мы решили нарисовать доску по строчкам (как кирпичный дом — ряд за рядом).

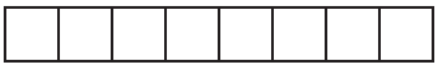
Тогда понадобятся два типа "кирпичиков" — закрашенные и незакрашенные.

Первая процедура должна рисовать просто квадратик. Пусть он будет начерчен по часовой стрелке. Напишите эту процедуру и с ее помощью попробуйте получить ряд из восьми незакрашенных квадратов.

Каждый раз после рисования квадрата черепашка должна переместиться туда, откуда она начнет рисовать следующий квадратик.


На рисунке отмечены начальное (**Н. П.**) и конечное (**К. П.**) положения черепашки после такого перемещения.

Следующий этап — научить черепашку рисовать закрашенный квадратик. Сначала черепашка должна нарисовать контур. Для этого можно воспользоваться готовой процедурой рисования квадрата (а не начинать сначала). Получится процедура второго уровня, т.е. процедура, вызывающая ранее описанную процедуру.



Затем нужно переместить черепашку внутрь контура, чтобы закрасить квадратик (с помощью команды `FILL`). А после закраски надо передвинуть черепашку на то место, откуда она начнет рисовать следующий квадратик.

Напишите процедуру для изображения закрашенного квадрата.

Теперь можно будет, чередуя вызовы первой и второй процедур и перемещения черепашки, получить первую строчку шахматной доски.

После выполнения процедуры, которая рисует строчку, опять необходимо передвинуть черепашку туда, где она начнет выполнять очередную процедуру, — в начало следующего ряда. Для такого перемещения удобно написать отдельную процедуру.

Вся шахматная доска — восемь строчек, четыре нечетные и четыре четные (в них клетки чередуются наоборот). Значит, нужно написать для строчек две процедуры — строчка нечетная и строчка четная.

Полученные процедуры для строчек — процедуры третьего уровня: в них вызывается процедура, которая сама вызывает процедуру.

И, наконец, последняя процедура — рисование шахматной доски целиком. В этой процедуре должно быть описано чередование нечетных и четных строк; после каждой строки — проход в начало следующей.

Таким образом, решая задачу, мы сначала последовательно разбили ее на подзадачи (сверху вниз):

а) шахматная доска складывается из восьми строк — четырех четных и четырех нечетных (плюс проходы из конца строки в начало следующей);

б) каждая строка состоит из восьми клеток — четырех закрашенных и четырех незакрашенных (плюс перемещения к началу следующей клетки);

в) закрашенную клетку можно получить из незакрашенной;

г) самая простая процедура — рисование незакрашенного квадрата.

Добравшись до конца, мы сначала описали самую простую процедуру, потом предыдущую и т.д. То есть описание и сборка осуществлялись в обратном порядке (снизу вверх).

**Задание**

*7.1. Напишите процедуру, которая рисует шахматную доску.*

## § 8. Конструирование изображений, состоящих из полуокружностей

Если дать черепашке задание нарисовать правильный многоугольник с большим числом сторон и с малой длиной стороны, например,

```
REPEAT 360 [FD 1 RT 1],
```

то получится почти окружность.

Чтобы получить на экране полуокружность, можно нарисовать только половину сторон соответствующего многоугольника. Например:

```
REPEAT 180 [FD 1 RT 1].
```

Каким числом регулируется размер этой полуокружности? Видимо, величиной стороны многоугольника (т.е., чтобы нарисовать вдвое большую полуокружность, нужно дать черепашке команду

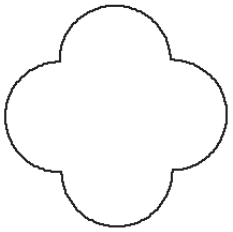
```
REPEAT 180 [FD 2 RT 1] ).
```

**Задания**

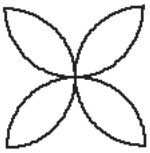
*Подготовьте необходимые процедуры для рисования полуокружностей и, используя их, напишите процедуры для следующих картинок.*

*8.1. Волны*

*8.2. Цветок*



*8.3. Цветок*

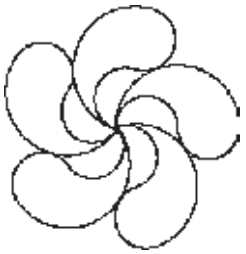


*8.4. Пружинка*

*8.5. Перо*

• *Для этой задачи потребуются три полуокружности: большая и две половинного размера.*

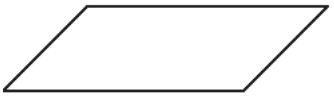
*8.6. Цветок*



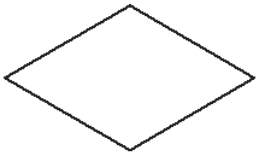
• *В решении нужно использовать результат предыдущего задания. Следовательно, должна получиться процедура третьего уровня.*

## § 9. Параллелограммы

Параллелограмм — четырехугольник, у которого попарно параллельны и равны стороны. Если все стороны параллелограмма равны, то он называется ромбом.



Параллелограмм



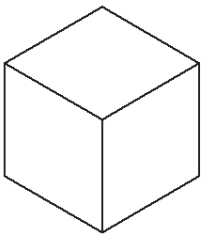
Ромб

**Задания**

*9.1. Напишите процедуру рисования ромба, в котором есть угол, равный 60°.*

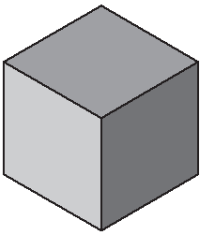
*9.2. Напишите процедуру рисования параллелограмма, в котором есть угол, равный 45°, и короткая сторона которого вдвое меньше длинной.*

*9.3. Напишите процедуру, которая изображает куб в изометрической проекции.*

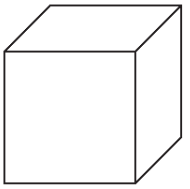


• *Используйте процедуру рисования ромба.*

*9.4. Раскрасьте грани полученного куба.*

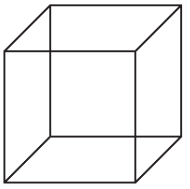


*9.5. Напишите процедуру, изображающую куб в диметрической проекции.*



• *Используйте процедуры рисования квадрата и параллелограмма.*

*9.6. Напишите процедуру, которая изображает "прозрачный" куб.*



• *Воспользуйтесь решением предыдущего задания.*

# ГЛАВА 3
# Процедуры с параметрами

## § 10. Описание процедуры с параметром

Команды языка Лого можно разделить на две группы. Одни выполняются всегда одинаково, и после команды никакой дополнительной информации указывать не нужно (например, `CG`, `PU`). Они называются командами без параметров. Другие команды требуют указания числовых или иных данных (например, `RT 80`). Это команды с параметрами (одним или несколькими).

Вы уже научились расширять список команд, которые может выполнять черепашка. Предположим, на изнанке листа описана следующая процедура:

```
ЭТО КВАДРАТ
REPEAT 4 [FD 140 RT 90]
END
```

Теперь в список команд черепашки входит команда `КВАДРАТ`, ее можно использовать наравне со всеми остальными командами. Возникает вопрос: к какой из двух групп команд ее отнести? Она всегда выполняется одинаково; чтобы ее выполнить, достаточно напечатать ее имя и нажать ввод. Поэтому команда `КВАДРАТ` — команда без параметров. К этой группе относились все процедуры, которые вы писали до сих пор.

Оказывается, можно изменить процедуру `КВАДРАТ` так, чтобы она принадлежала ко второй группе.

```
ЭТО КВАДРАТ :Z
REPEAT 4 [FD :Z RT 90]
END
```

Длина стороны квадрата теперь — переменная величина. Измененная таким образом процедура будет рисовать квадраты любого размера.

Двоеточие — не знак деления, а специальный признак перед именем переменной величины; между двоеточием и именем переменной не должно быть пробела.

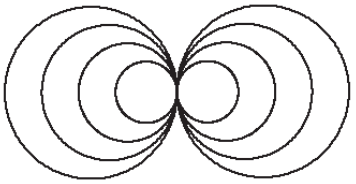
Теперь новая команда `КВАДРАТ` относится ко второй группе, и ее можно вызывать только с указанием числа. Например, получив команду `КВАДРАТ 10`, черепашка нарисует квадрат со стороной 10 шагов.

Таким же образом можно изменять и любые другие процедуры.

**Задания**

*10.1. С помощью процедуры, рисующей квадрат переменного размера, получите картинку, изображенную на рисунке.*

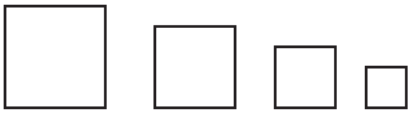
*10.2. Опишите две процедуры для рисования "правой" и "левой" окружности переменного размера. Создайте с их помощью процедуру для рисования "бабочки".*



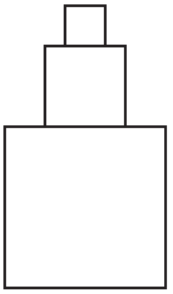
*10.3. Параметром может быть не только размер. Опишите процедуру для рисования и закрашивания окружности, в которой параметром является цвет черепашки. Нарисуйте на экране несколько кругов разного цвета.*

*При решении следующих задач используйте процедуры рисования фигур переменного размера.*

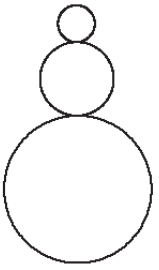
*10.4. Квадраты*



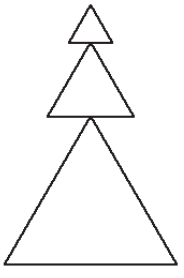
*10.5. Пирамида*



*10.6. Снеговик*



*10.7. Елочка*



### § 11. Отрицательное значение параметра

Некоторые команды Лого "понимают" отрицательные числа.

Например, можно дать черепашке команду `FD -90`, результат ее выполнения будет такой же, как результат команды `BK 90` (и наоборот);

`RT -90` = `LT 90` (и наоборот).

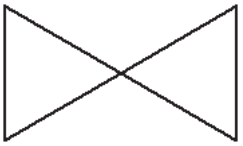
Этим свойством можно воспользоваться, чтобы сделать программы более короткими и простыми (в основном для симметричных рисунков).

**Задания**

*Получите следующие картинки, используя каждый раз только одну вспомогательную процедуру с положительными и отрицательными значениями параметров.*

*11.1. Бабочка (см. рисунок к заданию 10.2).*

*11.2. Бантик*



### § 12. Процедуры с несколькими параметрами

В процедуре может быть несколько параметров. Сколько чисел можно сделать параметрами в процедуре, которая рисует правильный многоугольник? Например:

```
ЭТО МНОГОУГОЛЬНИК
REPEAT 18 [FD 6 LT 20]
END
```

Нет, не три, а только два. И если одно из них определяет размер фигуры (длина стороны), то второе — форму (количество сторон многоугольника). Угол поворота изменять произвольно нельзя: многоугольник не получится.

Введем в процедуру два параметра и получим возможность рисовать любые правильные многоугольники:

```
ЭТО МНОГОУГОЛЬНИК :N :X
REPEAT :N [FD :X RT 360 / :N]
END
```

Изменяя первый параметр, можно получать разные фигуры: и треугольники, и пятиугольники, и другие. А при изменении второго параметра будет меняться размер фигур.

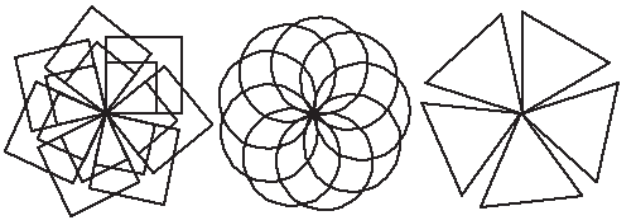
**Задания**

*Решите задачи 12.1—12.3, применяя только одну вспомогательную процедуру.*

*12.1. Получите многоугольники разных размеров и форм.*

*12.2. Нарисуйте картинку к заданию 4.4.*

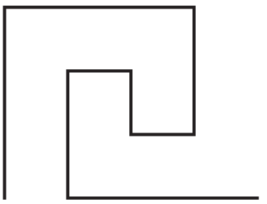
*12.3. Нарисуйте "цветы", отличающиеся друг от друга количеством и формой лепестков. Для этого напишите процедуру с тремя параметрами — для назначения формы лепестка, размера лепестка, количества лепестков. Она должна вызывать вспомогательную процедуру рисования многоугольника.*



*12.4. Введите параметр (размер клетки) в процедуры, которые использовались для рисования шахматной доски.*

*12.5. Добавьте в процедуры рисования шахматной доски второй параметр — цвет.*

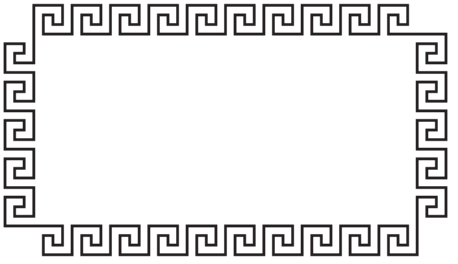
*12.6. Напишите процедуру для рисования элемента греческого орнамента — меандра.*



• *Выделите параметр, управляющий размером "завитка" меандра. Для этого можно нарисовать завиток на клетчатой бумаге и измерить все отрезки в клеточках. С изменением размера клетки будет пропорционально уменьшаться или увеличиваться вся картинка.*

*12.7. Напишите процедуру, которая рисует полоску орнамента — ряд завитков переменной длины. В этой процедуре должны быть уже два параметра: размер одного завитка и количество их в ряду.*

*12.8. Нарисуйте рамку (воспользуйтесь процедурой из предыдущего задания).*



# ГЛАВА 4
# Работа с формами. Элементы мультипликации

## § 14. Элементы мультипликации

Черепашка может поменять форму. Чтобы выбрать новую форму, нужно открыть лист форм. Он появится на месте поля команд, если нажать среднюю кнопку: 

Чтобы вернуть поле команд, нужно нажать верхнюю кнопку: 

Формы перенумерованы. Укажите мышкой на какую-нибудь из картинок, и под ней появится окошко с названием формы и с ее номером.

Изменяет форму черепашки команда SETSH (***set shape*** — установи форму), после которой нужно указать номер.

Например, в результате выполнения команды SETSH 5 черепашка превратится в дерево. Чтобы вернуть черепашке первоначальный вид, надо дать команду SETSH 0.

Черепашка с измененной формой выполняет все команды по-прежнему, но при поворотах форма не разворачивается — черепашка как бы спрятана за неподвижной маской.

Возможность изменения формы черепашки позволяет "оживить" предмет или персонаж на экране. Как мультипликаторы добиваются такого эффекта? Рисуют несколько картинок, немного отличающихся друг от друга, — несколько фаз движения. При быстрой их смене предмет или персонаж на экране как бы начинает двигаться.

В листе форм есть готовые наборы картинок с фазами движения, например, **пчела1** и **пчела2** (формы с номерами 11 и 12).

Поменяйте форму черепашки командой SETSH 11. Вместо черепашки вы увидите пчелу. Измените число 11 на 12 и нажмите клавишу. Пчелка "опустила крылышки".

Верните форму с номером 11 — и пчелка "поднимет крылышки".

Чтобы получился мультипликационный эффект движения, нужно написать программу, которая будет много раз менять форму черепашки.

Напишем простейшую программу для смены фаз:

```
REPEAT 100 [SETSH 11 SETSH 12]
```

Пробный запуск показывает, что крылья пчелы движутся слишком быстро.

Как замедлить смену картинок — форм черепашки? После каждой смены форм нужно вставить паузу.

В Лого есть специальная команда WAIT (жди). Ее используют в тех случаях, когда нужна задержка перед выполнением следующей команды. WAIT — команда с параметром, после нее надо указать число, определяющее длину паузы. Пауза измеряется в десятых долях секунды. Например, WAIT 1 — совсем короткая пауза, всего в одну десятую секунды, а WAIT 30 — задержка на три секунды.

Добавьте небольшую паузу после каждой команды смены формы и посмотрите, как это изменило работу программы.

Теперь надо сделать так, чтобы пчела полетела, а не размахивала крылышками, находясь на одном и том же месте.

Для этого за каждой сменой форм должна появиться команда продвижения. Действий в программе становится все больше, поэтому разумно разбить всю последовательность на части — процедуры:

```
ЭТО ШАГ1
SETSH 11 WAIT 1 FD 2
END
ЭТО ШАГ2
SETSH 12 WAIT 1 FD 2
END
ЭТО ПОЛЕТ
REPEAT 100 [ШАГ1 ШАГ2]
END
```

У вас должен получиться маленький мультфильм!

**Задания**

*14.1. Напишите процедуру, которая рисует линию горизонта.*

*14.2. Напишите три отдельные процедуры для осуществления вертикального подъема вертолета с линии горизонта, полета и спуска.*

*14.3. Измените процедуры так, чтобы подъем и спуск проходили по дуге (четвертой части окружности).*

*14.4. Напишите общую процедуру, в которой собраны все предыдущие.*

*14.5. Введите в общую процедуру параметр — скорость вертолета.*

Полет пчелы можно еще и озвучить — добавить в программу "жужжание" пчелы.

В ЛогоМирах есть команда для воспроизведения звуков разной высоты и длительности — NOTE (нота). Для того чтобы услышать ноты, вам нужны наушники или колонки!

NOTE — команда с двумя параметрами, после нее необходимо указать два числа: первое определяет высоту тона, второе — длительность звучания. Длительность измеряется в десятых долях секунды. А высота тона задается в соответствии со стандартом MIDI (***M**usical **I**nstrument **D**igital **I**nterface*, цифровой интерфейс музыкальных инструментов). По MIDI-стандарту ноты имеют номера от 0 до 127. Например, 60 — номер среднего ДО.

Отдайте команду NOTE 60 10 и послушайте ноту ДО длительностью в одну секунду.

**Задание**

*14.6. Добавьте звук в "полет пчелы": после команд смены формы вместо пауз вставьте ноты различной, но близкой высоты. Подберите высоту нот так, чтобы получалось "жужжание".*

## § 15. Получение изображений отпечатыванием

Во всех предыдущих графических программах рисунок возникал в результате перемещения черепашки (с опущенным пером). Существует другой способ получения изображений на экране. Черепашка может оставить след (отпечаток) на экране по команде STAMP. Перед этой командой перо черепашки должно быть опущено. Сначала нужно установить форму и цвет черепашки, а затем повторять следующие действия: поднять перо, переместить черепашку, опустить перо, оставить след.

**Задания**

*15.1. Напишите процедуру, которая рисует аллею — ряд деревьев.*

*15.2. Напишите процедуру для рисования ряда домов.*

*15.3. Соедините две предыдущие процедуры в общей, рисующей улицу.*

Используя команду STAMP, можно создавать изображения (отпечатки разных форм), расположенные регулярно или разбросанные хаотически, случайным образом. Для получения случайного числа служит датчик случайных чисел RANDOM (случайное). Случайное целое число выбирается из промежутка от нуля до значения, указанного после операции RANDOM. Выбранное датчиком случайных чисел значение нужно передать какой-либо команде. Например, для перемещения черепашки можно дать команду FD RANDOM 100 (пройди случайное число шагов — от 0 до 100). Для перемещения в случайное место необходимо выполнять два действия: продвижение на случайное число шагов и поворот на случайное число градусов.

**Задания**

*15.4. Напишите программу* ЗВЕЗДНОЕ НЕБО: *на экране в случайном порядке разбросаны желтые точки.*

*15.5. Добавьте в предыдущую процедуру "космическую музыку": после появления каждой звездочки — одну ноту случайной высоты; в начало процедуры добавьте закраску экрана синим цветом.*

*15.6. Напишите программу* ЛЕС: *на выбранном фоне в случайных местах растут деревья.*

## § 16. Анализ условия

Две последние процедуры из заданий 15.5 и 15.6 хорошо бы объединить и написать программу ПЕЙЗАЖ: экран поделен пополам, выше линии горизонта небо и звезды, ниже — лес.

Механически соединить две программы мешает то, что звезды появляются в лесу, а деревья вырастают на небе. Чтобы избежать этого, нужно сделать команду STAMP управляемой: черепашка-дерево "имеет право" оставить след только на фоне "земли", а черепашка-точка — только на фоне "неба".

Черепашка способна проверять цвет поля, на котором она находится. Для этого служит датчик COLORUNDER (два соединенных слова: "цвет под"). Результат выполнения операции COLORUNDER — номер цвета поля, оказавшегося под центром формы черепашки.

Для проверки условий и выполнения каких-то зависящих от них действий служит команда IF (если). По правилам записи команда условного ветвления IF похожа на команду REPEAT:

IF условие [что делать, если условие выполнено]

Например, нужно сделать так, чтобы черепашка оставляла отпечатки только на синем поле с номером цвета 105. Тогда надо написать команду условного выполнения:

```
IF COLORUNDER = 105 [ STAMP ]
```

То есть: если под черепашкой оказался цвет номер сто пять, то черепашка должна поставить отпечаток. Если окажется, что номер цвета поля не сто пятый, то команда STAMP выполнена не будет.

Внимание: знак равенства нужно выделять пробелами с двух сторон!

**Задания**

*16.1. Напишите процедуру, которая рисует линию горизонта и закрашивает небо и землю. В качестве заготовки воспользуйтесь процедурой из задания 14.1.*

*16.2. Напишите процедуру* ЗВЕЗДЫ.

• *За основу удобно взять решение задания 15.4 или 15.5 и сделать команду* STAMP *управляемой (звезды должны быть только на небе).*

*16.3. Напишите процедуру* ДЕРЕВЬЯ.

• *Основой для нее также может послужить решение задания 15.6. Чтобы предотвратить появляющиеся иногда на "небе" стволы деревьев, можно в нижней части экрана добавить еще одну узкую цветную полосу — дорогу.*

*16.4. Соберите предыдущие три процедуры в одной общей — для рисования пейзажа.*

*16.5. Добавьте в процедуру* ПЕЙЗАЖ *полет вертолета (задание 14.5 или 14.6).*

# ГЛАВА 5
# Вычислительные и диалоговые программы. Циклы

## § 17. Вывод на экран текста и результатов вычислений

Кроме графической информации (рисунков черепашки), на экран можно выводить тексты и числа. Для этого служат команды вывода PR (*print* — напечатай) и INSERT (вставь).

Различие в результатах выполнения команд PR и INSERT состоит в том, что после выполнения команды PR курсор переместится на новую строку, поэтому следующий фрагмент текста или число появится на экране обязательно с новой строки.

Если нужно вывести на экран какие-либо сообщения, то их следует заключать в квадратные скобки (например, PR [Назовите Ваше имя]).

**Задания**

*17.1. Напечатайте с помощью команды вывода в рабочем поле (в верхней части экрана) свои имя, отчество и фамилию.*

*17.2. Напишите процедуру, которая печатает 10 раз слово "Ура!" в строчку и 10 раз — в столбик.*

*17.3. Напишите процедуру с параметром, которая выводит на экран заданное число раз фразу "Ученье — свет, а неученье — тьма!".*

Кроме сообщений, на экран могут быть выведены:

• результаты вычислений — действий над числами. Например:

```
PR 35 * (12,37 + 5,84)
```

• значения переменных величин или результаты действий над переменными величинами.

**Правила записи арифметических выражений:**

• записывать арифметические выражения надо в строчку;

• деление обозначается знаком "/" (а не двоеточием);

• умножение обозначается знаком "*";

• скобки используются только круглые, даже когда одни скобки находятся внутри других;

• знаки арифметических действий должны быть выделены с двух сторон пробелами.

Например, нам нужно описать процедуру для умножения двух чисел:

```
ЭТО УМН :A :B
PR :A * :B
END
```

Чтобы процедурой было удобно пользоваться, надо позаботиться о комментариях к результатам (т.е. пояснить, что это за число появилось на экране). Добавим вывод сообщения:

```
ЭТО УМН :A :B
PR [Произведение чисел:]
PR :A * :B
END
```

В этой процедуре первая команда выводит сообщение, а вторая — результат действия (умножения) над переменными величинами.

Для стирания текста и чисел с экрана служит команда CT (***c**lear **t**ext* — сотри текст).

**Задания**

*17.4. Выведите на экран результаты возведения в куб чисел 3, 5, 10, 127; результат деления 127 на 73.*

*17.5. Напишите процедуру с параметрами, которая выводит на экран среднее арифметическое двух чисел.*

*17.6. Добавьте в этой процедуре вывод сообщения для пояснения того, что означает число, появившееся на экране (например: "Среднее арифметическое чисел — 6").*

*17.7. Напишите процедуру с параметрами, которая подсчитывает площадь стен комнаты (для ремонта), если известны ее длина, ширина и высота (наличие окон и дверей можно не учитывать). Вывод должен быть оформлен (например: "Площадь стен комнаты — 54 квадратных метра").*

## § 18. Команда присваивания

Для запоминания информации в компьютере имеется память. Иногда бывает необходимо поручить программе запомнить ("положить на хранение") некоторые данные. Каждый раз для этого нужно завести специальное хранилище (отвести место в памяти компьютера) и дать ему название, чтобы не спутать с другими. Вы как бы берете ящик для определенных целей и прикрепляете на нем наклейку с названием. Содержимое ящика может меняться, это переменная величина. Для заведения переменной (хранилища) служит команда присваивания MAKE (сделай).

Например, пусть выполнена команда:

```
MAKE "N 1
```

В результате выполнения команды MAKE заведена переменная (образовано хранилище) под названием *N*, и в это хранилище помещена единица. Мы можем присвоить переменной *N* другое значение:

```
MAKE "N 10
```

(сделай так, чтобы переменная N приняла значение, равное 10).

Таким образом, у каждой переменной есть имя (которое неизменно) и значение (которое может меняться). Команда MAKE заводит новую переменную или изменяет значение уже существующей переменной.

**Правила записи команды присваивания:**

• после команды надо сначала указать имя переменной, а затем значение, которое ей следует присвоить;

• перед именем переменной необходимо ставить двойные кавычки с одной стороны и без пробела.

Как узнать, каково в данный момент значение переменной с именем N? Можно напечатать его:

```
PR :N
```

В этом случае перед N стоит двоеточие, а не кавычки, так как имелось в виду значение переменной, а не ее имя. Значение переменной можно передать и другим командам, например:

```
FD :N
```

Команда присваивания оказывается необходимой в тех случаях, когда программа должна выполнять большое количество однотипных действий. Разберем задачу: надо написать программу для выведения на экран ряда целых чисел (например, от единицы до ста): 1 2 3 4 5 6 ... 100. Если бы такую работу поручили вам, то последовательность действий выглядела бы примерно так:

написать единицу,
прибавить один,
написать то, что получилось,
прибавить один,
написать то, что получилось, и т.д.

Пара действий (прибавить один; написать то, что получилось) должна быть выполнена 99 раз. Выписывая числовой ряд, вы не запутаетесь: ведь вы видите последнее написанное число (к которому нужно прибавлять единицу).

Теперь представьте себе, что эту же работу вам придется проделать в темноте. Тогда нужно будет все время хранить в памяти последнее написанное число. Предыдущие числа помнить не надо — необходимо только одно, последнее, и оно будет каждый раз меняться.

Попробуем записать эту последовательность действий с помощью команды присваивания.

```
MAKE  "K  1
```

(завести переменную K и присвоить ей значение, равное 1);

```
PR :K
```

(напечатать значение переменной K);

```
MAKE  "K  :K + 1
```

(сделать так, чтобы переменная K приняла новое значение, равное предыдущему плюс один);

```
PR :K
```

(напечатать новое значение переменной K);

```
MAKE  "K  :K + 1
```

(сделать так, чтобы переменная K приняла новое значение, равное предыдущему плюс один) и т.д.

Таким образом, первая команда выполнится один раз, а затем будет сто раз повторяться одна и та же пара команд:

```
ЭТО РЯД
MAKE  "K  1
REPEAT 100 [PR :K     MAKE "K :K + 1]
END
```

**Задания**

*18.1. Напишите процедуру, которая выводит на экран первые 100 нечетных чисел.*

*18.2. Напишите процедуру, печатающую все положительные четные числа, начиная с двухсот, в обратном порядке.*

## § 19. Цикл с изменяющейся переменной

Процедуры для печатания числовых рядов могут быть построены по единой схеме. Сначала с помощью команды присваивания MAKE заводится новая переменная и ей присваивается начальное значение. Затем начинаются повторения — многократно выполняется один и тот же набор действий. В этом наборе обязательно есть действие изменения значения переменной.

Значит, в такой программе команда присваивания MAKE встречается дважды. Первый раз при начальной установке переменной, второй раз — в цикле, при изменении значения переменной.

```
ЭТО РЯД
MAKE "K 1
REPEAT 100
       [ PR :K
         MAKE "K  :K + 1 ]
END
```

Первая команда присваивания выполняется один раз, вторая — многократно. При изменении переменной в цикле используется ее же предыдущее значение.

Вот почему в команде MAKE "K :K + 1 переменная упоминается дважды (присвоить переменной с именем K то значение, которое сейчас имеет переменная K, увеличенное на единицу). Сначала упоминается имя (на это указывает двойная кавычка), а затем — значение (поэтому поставлено двоеточие).

**Задания**

*Для следующих заданий напишите программы в виде циклов с изменяющейся переменной.*

*19.1. Нарисуйте квадратную спираль.*

*Многократно должны повториться следующие действия:*

• *черепашка продвигается на некоторое расстояние (это расстояние — переменная величина);*

• *черепашка поворачивается на 90°;*

• *значение переменной изменяется (например, увеличивается).*

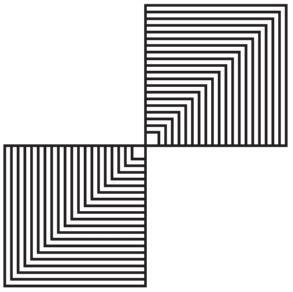
*19.2. Нарисуйте треугольную спираль.*

*19.3. Напишите программу, проигрывающую "звуковую дорожку" (ряд нот с повышающейся или понижающейся высотой).*

*19.4. Напишите программу для подсчета потомства одной амебы за двое суток, если каждая амеба через два часа делится на две.*

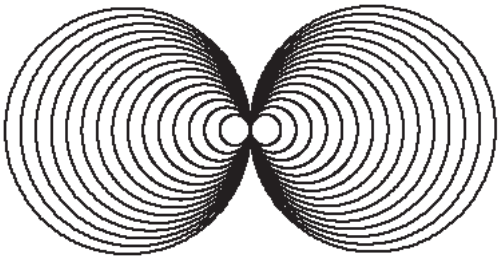
*19.5. Измените угол поворота в спиралях на 1—2°.*

*19.6. Напишите программу, рисующую палитру цветов — таблицу из квадратиков, закрашенных всеми имеющимися в среде ЛогоМиры цветами (номера цветов должны постепенно изменяться от 0 до 139).*



*19.7. Напишите программу, которая рисует увеличивающиеся квадраты. Проверьте, что получится при малом изменении переменной в цикле.*

*19.8. Напишите программу, рисующую "бабочку" с большим количеством окружностей.*



*19.9. О происхождении шахмат существует такая легенда. Мудрец, придумавший эту игру, попросил у раджи плату: за первую клетку доски — одно рисовое зерно, за вторую — два, за каждую следующую — вдвое больше, чем за предыдущую. Напишите программу для подсчета количества зерен за каждую клетку и выясните, после какой клетки числа становятся слишком большими для среды ЛогоМиры. Определите, сколько тонн весит такое количество риса (если считать, что вес одного зерна — 0,1 г).*

*19.10. Напишите процедуру с двумя параметрами для возведения заданного числа в заданную (натуральную) степень.*

## § 20. Операции ввода. Организация диалога

Диалог — это поочередные высказывания двух собеседников. Чтобы написать программу, поддерживающую пусть самый простой диалог с человеком, необходимы вывод информации на экран и ввод информации с клавиатуры.

Вам уже хорошо знакомы команды вывода PRINT и INSERT. Перед тем как работать с ними, нужно открывать текстовое окно.

Специально для диалога есть еще команды QUESTION и ANNOUNCE, а также операция ANSWER.

Давайте познакомимся с ними, напишем диалоговую программу ЗНАКОМСТВО. Сначала компьютер задает вопрос:

```
QUESTION [Как Вас зовут?]
```

Наберите эту команду. На экране должно появиться специальное диалоговое окно. В нем вопрос, а ниже — место для ввода ответа. В этом поле мигает курсор, приглашая вас напечатать ответ. Напечатайте ответ и щелкните по кнопке "OK". После этого диалоговое окно закроется.

Полученную информацию (ответ на вопрос) полезно запомнить, чтобы в дальнейшем ее можно было использовать. Запомнить введенные данные помогут команда MAKE и новая операция ANSWER (ответ):

```
QUESTION [Как Вас зовут?]
MAKE "N ANSWER
```

В переменной N теперь хранится ваш ответ на вопрос программы. Эту информацию можно использовать, чтобы запрограммировать вежливый ответ компьютера: "Приятно познакомиться, Вася!" или "Приятно познакомиться, Маша!". Для этого существует еще одна команда вывода — ANNOUNCE (сообщи).

```
QUESTION [Как Вас зовут?]
MAKE "N ANSWER
ANNOUNCE (SE [Приятно познакомиться,] :N)
```

Команда ANNOUNCE открывает окно для вывода сообщения (этим она и отличается от команд PRINT и INSERT). Прочитаете — щелкните по кнопке "OK", и окно исчезнет.

Сообщение состоит из двух частей. Первая часть — это текст, он всегда одинаковый. Вторая часть — имя — переменная величина. Чтобы "склеить" эти две части в одно предложение, понадобилась еще одна новая операция — SE (от англ. *sentence* — предложение). Операцию SE вместе со всеми частями предложения надо помещать в круглые скобки.

Полученная программа обеспечивает три хода: ход компьютера — вопрос, ход человека — ответ, опять реплика компьютера (вывод — ввод — вывод).

**Задания**

*20.1. Оформите процедуру* ЗНАКОМСТВО, *проверьте, как она работает.*

*20.2. Добавьте в процедуру* ЗНАКОМСТВО *еще один вопрос: "Ваша фамилия?". В ответе программы должны быть названы и имя, и фамилия.*

*20.3. Напишите программу, которая спрашивает, сколько вам лет, и, получив ответ, называет год вашего рождения.*

*20.4. Напишите программу, которая задает два вопроса: какой у вас вес и какой рост. После этого программа должна назвать норму веса для вас. Считается, что нормальный вес (в килограммах) — это рост (в сантиметрах) минус 110.*

*20.5. Напишите процедуру рисования закрашенного правильного многоугольника. Необходимые данные (цвет, количество и длина сторон) должны быть введены в диалоге.*

*20.6. Напишите программу рисования произвольной геометрической композиции. В программе нужно многократно повторять следующие действия:*

- *получить информацию о перемещении черепашки (поворот и продвижение);*
- *переместить черепашку;*
- *запустить процедуру из предыдущего задания.*

*Таким образом, каждый раз будут заданы пять вопросов. Для выбора номера цвета можно воспользоваться палитрой (см. задание 19.6).*

*20.7. Напишите в виде цикла с переменной программу* КАССА *для сложения десяти вводимых с клавиатуры чисел.*

## § 21. Анализ условия и разветвление в диалоге

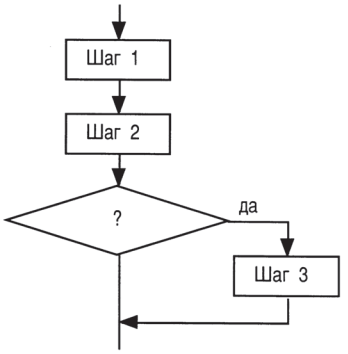
Иногда бывает нужно запрограммировать разветвление в программе. "Направо пойдешь — голову сложишь, налево пойдешь — коня потеряешь". Для таких случаев используют команды условного ветвления. Вам уже встречалась команда **IF**. Помните, в программе **ПЕЙЗАЖ** нужно было выполнять команду отпечатывания звездочки не всегда, а только при выполнении определенного условия:

```
IF COLORUNDER = 105 [ STAMP ]
```

Если под черепашкой оказался цвет номер сто пять, то черепашка должна поставить отпечаток. В противном случае команда STAMP выполнена не будет.

Такое разветвление в последовательности действий, когда в случае выполнения условия необходимо проделать некоторые действия, а в противном случае — ничего не делать, называется неполным условным ветвлением.

Его можно схематически изобразить так:



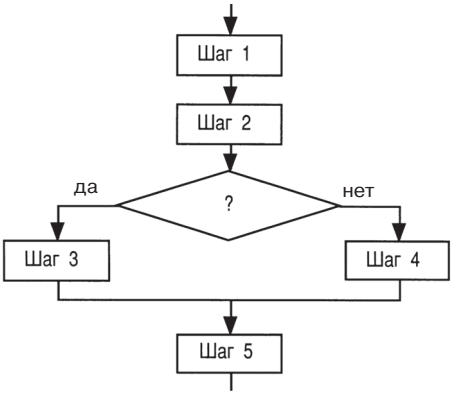

В других случаях при выполнении условия надо сделать одно действие, а при невыполнении — другое. Например, если молодец отгадает загадку царя, то получит полцарства, а не отгадает — в темницу попадет.

Для такого случая — полного условного ветвления — предназначена команда IFELSE (если—иначе). Правила ее записи такие:

```
IFELSE условие [делать, если условие выполнено]
  [делать, если условие не выполнено]
```

После условия требуются две пары квадратных скобок, в которых записаны команды.

Полное условное ветвление можно схематически изобразить так:




При записи условий можно использовать не только знак равенства, но также знаки "больше" и "меньше" (">" и "<"). Все эти знаки нужно с обеих сторон отделять пробелами.

**Задания**

*21.1. Напишите программу, которая спрашивает, какие у вас вес и рост (две отдельные переменные), и сообщает, превышает ваш вес норму или нет (нормальный вес вычисляется как разница между ростом в сантиметрах и числом 110).*

*21.2. Напишите программу* ТИР *по следующему сценарию:*

• *черепашка без следа перемещается в случайное место на экране и рисует небольшой закрашенный круг (мишень); затем она возвращается на исходную позицию;*

• *программа задает вопросы о повороте и продвижении черепашки и перемещает ее в соответствии с полученными данными;*

• *если черепашка "попала в цель" (можно использовать датчик* COLORUNDER*), то программа печатает одно сообщение, в противном случае — другое. Учтите, что черепашка "проверяет" цвет центром формы.*

*Сначала подготовьте отдельные процедуры для частей программы, а затем соберите их в общей процедуре.*

*21.3. Напишите процедуру с двумя параметрами, которая дает задание перемножить два натуральных числа, ожидает ввода ответа и затем сообщает, правильный он или нет.*

*21.4. Напишите программу-тренажер* ТАБЛИЦА УМНОЖЕНИЯ*, которая 10 раз загадывает два случайных числа от 0 до 15 (с помощью команды* MAKE *и операции* RANDOM*) и вызывает процедуру из предыдущего задания.*

*21.5. Предусмотрите в тренажере подсчет правильных ответов и выставление оценки.*

*21.6. Напишите программу, которая спрашивает у покупателя, в какой вагон нужен билет (плацкартный, купейный или общий), и называет соответствующую цену.*

## Глава 6
## Использование рекурсивных процедур

### § 22. Процедура, обращающаяся к самой себе

При решении задач из предыдущих параграфов вы часто использовали метод выделения составных частей в задаче. Программа в результате содержит обращения к процедурам, которые, в свою очередь, могут обращаться к другим процедурам, и т.д.

А что получится, если в процедуре записать обращение к самой себе? Например:

```
ЭТО A
REPEAT 4 [FD 150 RT 90]
A
END
```

Процедура A нарисует квадрат и вызовет процедуру A, которая нарисует квадрат и вызовет процедуру A, которая... и так далее. Черепашка будет двигаться по одному и тому же контуру до тех пор, пока вы не прервете программу. Для этого нужно выбрать в главном меню пункт **Редактор**, команда **Останов**.

Про такую программу говорят, что она содержит рекурсивный вызов самой себя.

Можно привести множество примеров рекурсии из жизни. На фантике конфеты "А ну-ка, отними!" нарисована девочка, в руке у нее конфета, на которой нарисована девочка, в руке у нее конфета... и т.д. Существуют "бесконечные" стихотворения типа "У попа была собака...". В интерьерах часто используется прием расположения зеркал друг против друга. Так можно добиться, например, эффекта "бесконечной" анфилады комнат.

Вернемся к процедуре A. Что изменится, если уменьшить число повторений?

```
ЭТО A
REPEAT 3 [FD 150 RT 90]
A
END
```

Легко убедиться, что результат выполнения тот же. При дальнейшем уменьшении числа повторений (2, 1) ничего не меняется. Так как команду REPEAT 1 ... можно вообще не писать, процедура упрощается:

```
ЭТО A
FD 150 RT 90
A
END
```

**Задания**

*22.1. Напишите рекурсивную процедуру, которая до прерывания двигает черепашку по треугольной, пятиугольной, шестиугольной траекториям; по окружности.*

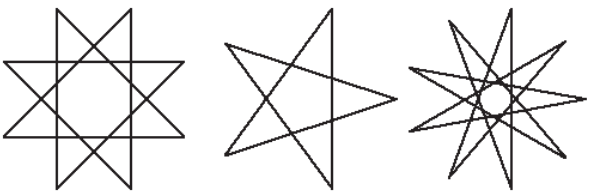
*22.2. Напишите процедуру, которая до прерывания печатает "бесконечное" стихотворение про попа и собаку или следующее:*

*У царя был двор, На дворе был кол, На колу — мочало, Начинай сначала...*

*22.3. Сформулируйте, как будет меняться траектория движения, если в процедуре* A *изменять параметр команды* FD*? Параметр команды* RT*?*

*22.4. По какой траектории будет двигаться черепашка, если угол поворота в процедуре* A *изменить с 90° на 50°? Сначала попробуйте нарисовать в тетради, а затем проверьте на компьютере.*

*22.5. Подберите угол поворота в процедуре рисования каждой из следующих звезд:*



Продолжим исследование рекурсивной процедуры A. Следующий шаг — ввести параметр, чтобы черепашка могла бегать по квадрату любого размера:

```
ЭТО A :X
FD :X RT 90
A :X
END
```

Значительно более интересный результат получится, если при рекурсивном вызове изменить значение параметра. Например, каждый раз увеличивать переменную *X* на три:

```
ЭТО A :X
FD :X RT 90
A :X + 3
END
```

Получается знакомая квадратная спираль, только "бесконечная".

**Задания**

*Выполните следующие задания, используя рекурсию и прерывание.*

*22.6. Получите на экране спирали различной формы, увеличивающиеся и уменьшающиеся.*

*22.7. Проиграйте "звуковую дорожку" (ряд звуков с повышающейся или понижающейся высотой).*

*22.8. Выведите на экран натуральный ряд.*

## § 23. Управляемая рекурсия

Как написать рекурсивную программу, чтобы она вовремя остановилась? Можно использовать условие и команду STOP.

```
ЭТО СПИРАЛЬ :X
IF :X > 200 [ STOP ]
FD :X RT 90
СПИРАЛЬ :X + 5
END
```

Если при очередном вызове параметр *X* оказался больше 200, то программа остановится. В противном случае последуют продвижение, поворот и новый вызов с увеличенным параметром.

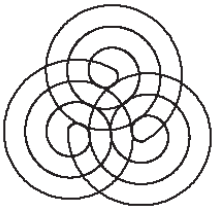
Черепашку можно сделать невидимой, если ввести команду HT (***h**ide **t**urtle* — спрячь черепашку). После этого черепашка начнет перемещаться с большей скоростью. Кроме того, она не будет мешать наблюдать рост и уменьшение спиралей. Обратная команда — ST (***s**how **t**urtle* — покажи черепашку).

**Задания**

*23.1. Напишите рекурсивные процедуры для треугольной, пяти-, шести-, десятиугольной раскручивающихся спиралей.*

*23.2. Напишите процедуру* УЛИТКА *для круглой спирали.*

*23.3. Используя решение предыдущего задания, получите следующий рисунок:*



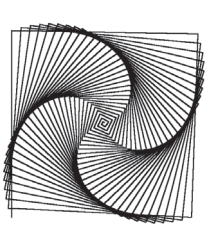
*23.4. Напишите процедуры для "скручивающихся" спиралей.*

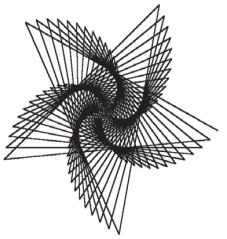
*23.5. Напишите программу, которая разворачивает и сворачивает спираль.*

*Программа должна обращаться к двум процедурам: спираль раскручивается, затем черепашка меняет направление и цвет (на черный), и спираль, скручиваясь, стирает рисунок.*

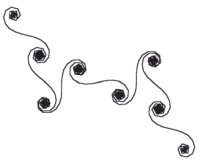
*23.6. Напишите для рисования спиралей процедуру, у которой два параметра — шаг и изменение шага.*

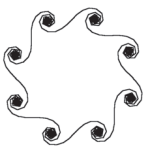
*23.7. Получите "винтовые" увеличивающиеся и уменьшающиеся спирали, немного изменив угол поворота:*





*23.8. Напишите рекурсивную процедуру, в которой каждый раз при вызове изменяется не шаг, а угол поворота. Подберите начальное значение угла и величину его изменения для следующих рисунков:*





## § 24. Косвенная рекурсия

Программа из задания 23.5, которая сначала рисует спираль, а затем ее стирает, обращается по очереди к двум процедурам. Эти две процедуры (спираль растущую и спираль уменьшающуюся) можно связать как бы крест-накрест: первая должна вызывать вторую, а вторая — первую. В результате мы получим один из примеров косвенной рекурсии, когда процедура вызывает сама себя не непосредственно, а через другую процедуру.

```
ЭТО СПИРАЛЬ_РАСТЕТ :s
FD :s RT 90 WAIT 1
IF :s > 200 [RT 90 SETC 0 СПИРАЛЬ_ИСЧЕЗАЕТ :s]
СПИРАЛЬ_РАСТЕТ :s + 3
END
ЭТО СПИРАЛЬ_ИСЧЕЗАЕТ :s
FD :s LT 90 WAIT 1
IF :s < 0 [LT 90 SETC 9 СПИРАЛЬ_РАСТЕТ :s]
СПИРАЛЬ_ИСЧЕЗАЕТ :s – 3
END
```

Запускать можно любую из двух процедур — результат будет один и тот же. Получатся бесконечные (до прерывания) колебания. (Иногда рекурсивные процедуры могут прерываться из-за того, что для продолжения процесса не хватает памяти.)

**Задания**

*24.1. Напишите систему из двух связанных процедур для проигрывания звуковых дорожек вверх и вниз.*

## § 25. Программа для игры "Угадайка"

В этой игре один из играющих загадывает число, а второй старается его отгадать. После каждой неудачной попытки первый игрок дает один из двух ответов: "Перелет" (если названное число больше задуманного) или "Недолет" (в противном случае).

Давайте напишем программу, выполняющую роль первого игрока (загадывающего число). Сначала в программе должна быть заведена переменная и в нее занесено случайное число, например, от 0 до 1000. Затем будут многократно повторяться следующие действия:

а) вывод вопроса ("Назовите число");

б) получение (ввод) ответа;

в) если число угадано, то вывод поздравления и окончание работы;

в противном случае — сравнение неправильного ответа с загаданным числом:

если ответ больше, чем число, то вывод сообщения "Перелет";

в противном случае — вывод сообщения "Недолет";

далее опять пункты а), б), в) ...

Так как нельзя ни предсказать, ни рассчитать заранее, сколько раз должен повториться этот набор действий, целесообразно описать его как рекурсивную процедуру. Таким образом, структура программы получится следующая: процедура УГАДАЙКА присваивает случайное значение переменной *N* и обращается к рекурсивной процедуре ПОПЫТКА (которая реализует описанные действия).

**Задания**

*25.1. Напишите рекурсивную процедуру* ПОПЫТКА. *Процедура должна задавать вопрос о значении некоторой переменной, принимать ответ и выдавать одну из трех резолюций: "недолет", "перелет", "угадали!".*

*25.2. Напишите процедуру* УГАДАЙКА, *которая "загадывает" случайное число и обращается к процедуре* ПОПЫТКА.

*25.3. Введите в программу счетчик попыток.*

*25.4. В программе* ТИР *(задание 21.2) играющему предоставлялась возможность только одного "выстрела". Усовершенствуйте программу: выделите и опишите рекурсивную процедуру, которая до попадания в цель многократно возвращает черепашку в исходную точку и затем перемещает ее согласно введенным с клавиатуры данным.*

## § 26. Непосредственное управление черепашкой с клавиатуры

В среде ЛогоМиры есть операция ввода символа READCHAR (***read char**acter* — прочитанный символ). Эта операция позволяет сообщить программе один символ. Ее можно использовать для написания игровых программ. Мы создадим с ее помощью программу, позволяющую двигать черепашку нажатием на различные клавиши. Пусть клавиша F продвигает черепашку на 20 шагов вперед; клавиша L поворачивает черепашку против часовой стрелки на 90°; клавиша R поворачивает черепашку по часовой стрелке на 90°.

```
ЭТО ЧЕРТЕЖНИК
MAKE "K READCHAR
IF :K = "L [LT 90]
IF :K = "R [RT 90]
IF :K = "F [FD 20]
ЧЕРТЕЖНИК
END
```

Если будет нажата любая другая клавиша, то программа ничего не сделает и снова вызовет сама себя (чтобы получить новый символ с клавиатуры, проанализировать его и выполнить закодированное действие) и т.д. Закончить работу можно только прерыванием (пункт меню **Редактор**, команда **Останов**).

Важно: когда запишете процедуру в лист программ и запустите ее в поле команд, щелкните мышкой в любом месте листа. Только после этого READCHAR начнет передавать данные в программу.

**Задания**

*26.1. Напишите программу для рисования, расширив набор управляющих клавиш, и нарисуйте любую картинку.*

*Определите не слишком большой набор действий, достаточно удобный для рисования: необходимы повороты направо/налево на 90° и на малый угол (или сразу рисование дуг различной кривизны), очистка экрана, заливка цветом, возможность поднять/опустить перо и т.д. Для изменения цвета черепашки можно назначить одну клавишу, которая будет менять цвет черепашки на случайный; нажав на эту клавишу несколько раз, вы подберете нужный цвет.*

*26.2. Добавьте в программу возможность выхода при нажатии какой-либо определенной клавиши.*

Среда ЛогоМиры может сохранить отдельно файл с картинкой. Для этого служит команда SAVEPICT (***save pict**ure* — сохрани картинку). После команды нужно указать имя (не более восьми латинских букв), а перед именем — двойную кавычку без пробела. Например:

```
SAVEPICT "CAT
```

Для обратного действия — считывания на экран записанной ранее картинки — предназначена команда LOADPICT (***load pict**ure* — загрузи картинку). Например:

```
LOADPICT "CAT
```

*26.3. Добавьте в программу (графический редактор) возможность записи и загрузки картинок.*

*Определенная для этого клавиша должна запускать процедуру, которая задает вопрос об имени графического файла, получает ответ и выполняет запись или загрузку.*

*26.4. Напишите программу* АЛЬБОМ, *которая "перелистывает" записанные картинки, загружая их по очереди на экран.*

*Чтобы картинку можно было рассмотреть, предусмотрите перед загрузкой следующей картинки управляемую паузу. Для этого можно использовать команду* ANNOUNCE.

# Анимация. Flash-технология. Автоматы

М.А. МАЗИН, А.А. ШАЛЫТО,
*Санкт-Петербург*

## 1. Введение

В настоящее время интерактивная анимация широко используется в различных приложениях, таких, как web-дизайн, электронные обучающие пособия и т.д.

Создание такой анимации возможно при непосредственном программировании на различных языках, например, C++ или *Java*. В этой области традиционно применяются графические библиотеки, из которых наиболее распространены *OpenGL* [1] и *DirectX* [2]. Первая характеризуется независимостью от платформы, а вторая, кроме средств работы с графикой, предоставляет также широкий спектр других средств разработки мультимедийных приложений.

Существуют средства, позволяющие ускорить процесс создания анимации за счет использования редакторов трехмерной графики, например, пакет *3D-Max* [3]. Обычно такие пакеты применяются совместно с указанными выше графическими библиотеками.

Однако если необходимо разрабатывать приложения для сети Интернет, то наиболее предпочтительным оказывается использование пакета *Macromedia Flash* [4], являющегося основой *Flash-технологии*. Большая популярность этой технологии в мире [5] объясняется рядом ее достоинств, среди которых следует отметить ее поддержку большинством современных браузеров и компактность исполняемого файла.

Как правило, содержание анимации задается ее сценарием. Различают линейные и нелинейные сценарии.

Если для реализации *линейных сценариев*, в которых отсутствуют разветвления, достаточно средств встроенного редактора *Macromedia Flash*, то для *нелинейных сценариев*, отличающихся наличием разветвлений, дополнительно применяется язык *ActionScript* [6]. В интерактивной анимации, в отличие от мультфильмов, обычно используются нелинейные сценарии.

Язык *ActionScript* в основном применяется для реализации достаточно простых нелинейных сценариев. В более сложных случаях его традиционное применение затруднительно. Так, например, интерпретатор не сообщает об использовании в выражениях не инициализированных переменных, что приводит к тому, что ошибку в написании имени переменной можно искать часами.

Авторами предлагается применять автоматную парадигму для формализации перехода от заданного сценария к коду программы на языке *ActionScript*. Это позволяет устранить указанные недостатки.

## 2. Разработка интерактивной анимации

Разработка анимации предполагает решение четырех подзадач:

- создание сценария;
- формализация сценария;
- “рисование”;
- программирование.

Решать указанные подзадачи будем на основе *SWITCH-технологии* [8, 9], базирующейся на парадигме автоматного программирования, которая называется также “программирование с явным выделением состояний”.

По вербальному заданию (тексту сценария) строится его математическая модель в виде графа переходов конечного автомата. Для этого сюжет разбивается на сцены — статические изображения. В каждой сцене присутствуют элементы управления, с помощью которых пользователь осуществляет ввод данных. Каждая сцена определяет состояние автомата, в котором указанное изображение формируется как выходное действие.

При этом пользовательский ввод генерирует входные переменные и порождает события, с которыми вызывается автомат. В зависимости от типа события и от значений входных переменных вызываемый автомат может осуществлять переходы.

Анимация, связанная с задачей, составляет совокупность дополнительных выходных действий, которые выполняются при переходах автомата.

Кроме того, при переходах выполняются выходные действия, связанные с управлением средой исполнения, например, таким действием является завершение работы флэш-плеера.

Изложенное позволяет построить схему связей автомата, которая описывает его интерфейс. При этом автомат управляет статическими изображениями, анимацией и средой исполнения.

По построенному графу переходов формально и изоморфно строится текст подпрограммы, который реализует его на основе конструкции *switch* языка *ActionScript*. Функции входных переменных и выходных действий на этом этапе заменяются функциями-заглушками. Обработчики событий выделяют из потока обрабатываемые события и вызывают с эти-

ми событиями автомат, который в зависимости от своего текущего состояния и значений входных переменных может осуществлять переходы и выходные действия.

После реализации графа переходов программист пишет программные модули, соответствующие входным переменным, обработчикам событий и выходным действиям. В то же время, руководствуясь схемой связей автомата, художник "рисует" статические изображения и выполняет раскадровку анимации.

Тем самым создается программа, состоящая из обработчиков событий, конструкции *switch* и функций, вызываемых из нее.

Отладка программы в рамках предлагаемого подхода осуществляется с помощью протоколов (логов), для построения которых вводятся функции протоколирования, вызываемые из конструкции *switch*, функций выходных действий, входных переменных и обработчиков событий.

## 3. Пример

Рассмотрим предлагаемый процесс на примере разработки интерактивной задачи по физике. В качестве основы для сценария используем упражнение из задачника [7]: "Ученик заметил, что палка длиной 1,2 м, поставленная вертикально, отбрасывает тень длиной 0,8 м. А длина тени от дерева оказалась в это же время равной 9,6 м. Какова высота дерева?"

Создадим по формулировке задачи сценарий. Так как в Древней Греции геометрическая оптика получила особенное развитие, в сценарии будут использованы древнегреческие образы: старец в тоге, древнегреческий храм, кипарис, высоту которого измеряем в задаче, и Гелиос на колеснице.

Приведем неформальный текст сценария.

*Ландшафт, на заднем плане — дерево. Стоит Старец. Появляется надпись: "Как можно измерить высоту дерева, используя законы геометрической оптики?"*

*Старец берет палку, втыкает ее в землю. Появляется надпись: "Запиши в тетрадь значение длины палки". Рядом с палкой появляется линейка. Крупные метки на линейке позволяют зафиксировать ее длину. Метка, соответствующая длине палки, окрашивается в другой цвет.*

*В тетради (на пергаменте) появляется надпись: "Длина палки h = _____ локтей". Тетрадь выполнена из пергаментных листов. Шрифт надписи должен быть стилизован. Если пользователь ввел неверный ответ, то появляется надпись: "Неверно! Попробуй еще раз".*

*При верном ответе Старец обращает внимание ученика на тени. Это выражается в том, что он показывает рукой на тени и предлагает измерить длину теней — от дерева и от палки.*

*Появляется надпись: "Посмотри на тени. Измерь и занеси их длины в тетрадь".*

*В тетради появляются надписи: "Длина тени палки l = _____ локтей", "Длина тени дерева = _____ локтей". Если пользователь ввел неверные длины теней, то появляется надпись: "Измерь точнее длины теней".*

*Когда пользователь ввел верный ответ, появляется надпись: "Теперь у тебя есть все данные, чтобы вычислить высоту дерева". В тетради появляется окно для ввода ответа "Высота дерева H = _____ локтей". Если введен неправильный ответ, то появляется подсказка.*

***Подсказка:** наблюдаем геометрическое построение (на пергаменте) — два подобных треугольника. Вертикальные стороны этих треугольников составлены для большего треугольника — из дерева, а для меньшего — из палки. Появляется надпись: "Неверно! Посмотри чертеж. Обрати внимание, что лучи от солнца параллельны".*

*Если пользователь опять вводит неправильное число, то выводится решение — описанный чертеж и вывод формулы для вычисления высоты дерева. Возникает надпись: "В решении этой задачи используется правило подобия треугольников".*

Перейдем к формализации сценария. Первоначально введем элементы управления, а также выделим входные переменные, события и выходные действия.

Для управления анимацией введем следующие кнопки и поля:

- кнопка "Выход" — по ее нажатию флэш-плеер прекращает работу;
- кнопка "Следующая задача" — по ее нажатию начинается загрузка следующей задачи. В примере рассматривается одна задача, и поэтому следующей задачей также является рассматриваемая;
- кнопка "Повторить задачу" — по ее нажатию задача запускается с новыми значениями величин, фигурирующих в ее условии, например, новыми длиной палки и высотой дерева;
- четыре текстовых поля для ввода длины палки, длин теней палки и дерева, а также высоты дерева. В тексте сценария полям соответствуют подчеркнутые пробелы;
- кнопка "Готово" — пользователь нажимает эту кнопку после окончания ввода ответа в соответствующие поля. По ее нажатию проверяется правильность введенных значений;
- кнопка "Дальше" — ее нажатие инициирует переход от одной статической сцены к другой.

Несмотря на то что при запуске автомат уже находится в начальном состоянии, его необходимо инициализировать событием *e0* для того, чтобы ав-

томат смог сформировать статическое изображение в начальном состоянии.

Во время загрузки файла с анимацией для комфорта пользователя на экран выводится "предзагрузчик" (короткая циклическая анимация) — изображение Гелиоса, пересекающего небосклон.

Выделим сцены, каждой из которых присвоим номер, соответствующий вершине графа переходов. Переходы обозначим двойным номером $(i, j)$, где $i$ — номер сцены, из которой осуществляется переход, а $j$ — сцена, в которую осуществляется переход. Введем также обозначение событий $e$, инициирующих переходы. Входные переменные обозначим $x$, если условием перехода является истинное значение переменной, и $!x$ — в противном случае. Кроме того, введем обозначения выходных действий, которые могут быть трех типов: статические изображения $zs$, анимация $za$ и системные действия $z$. При этом статические изображения формируются в вершинах, а анимация и системные действия — на переходах.

Приведем формальный текст сценария.

*0. Предзагрузчик: Гелиос пересекает небосклон на колеснице (zs0).*

*(0, 1) По завершению загрузки (e3) появляется надпись: "Как можно измерить высоту дерева, используя законы геометрической оптики?" (za0), происходит инициализация значений начальных условий задачи (z2).*

*1. Ландшафт, на заднем плане — дерево. Стоит Старец (zs1).*

*(1, 2). По нажатию кнопки "Дальше" (e2) Старец берет палку и втыкает ее в землю. Появляется надпись: "Запиши в тетрадь значение длины палки" (za1).*

*2. Рядом с палкой появляется линейка. Крупные метки на линейке позволяют зафиксировать ее длину. Метка, соответствующая длине палки, окрашивается в красный цвет. В тетради (на пергаменте) появляется надпись: "Длина палки h = _____ локтей". Шрифт надписи стилизован — используется шрифт "капитальное письмо" (zs2).*

*(2, 2). Если пользователь ввел (e1) неверный ответ (!x1), то появляется надпись: "Неверно! Попробуй еще раз", Старец мотает головой (za3). Ответ вводится в поле "Длина палки h = ____". Для подтверждения ввода пользователь нажимает кнопку "Готово".*

*(2, 3). В случае ввода (e1) верного ответа (x1) появляется надпись: "Посмотри на тени с линейками. Измерь и занеси их длины в тетрадь" (za2).*

*3. Старец в тоге обращает внимание пользователя на тени. Это выражается в том, что он показывает рукой на тени и предлагает измерить длину теней — от дерева и от палки. В тетради появляются надписи: "Длина тени палки l = _____ локтей", "Длина тени дерева L = _____ локтей" (zs3).*

*(3, 3). Если пользователь ввел (e1) неверный ответ — ошибочные значения длин теней (!x2), то появляется надпись: "Измерь точнее длины теней" (za4).*

*(3, 4). Когда пользователь ввел (e1) верный ответ (x2), появляется надпись: "Теперь у тебя есть все данные, чтобы вычислить высоту дерева", Старец кивает (za5).*

*4. В тетради возникает окно для ввода ответа "Высота дерева H = _____ локтей" (zs4).*

*(4, 5). В случае ввода (e1) неправильного ответа — высота дерева введена ошибочно (!x3), появляется надпись: "Неверно! Посмотри чертеж. Обрати внимание, что лучи от солнца параллельны" (za6).*

*(4, 7). Если пользователь ввел (e1) верный ответ (x3), появляется надпись об успешном решении задачи: "Задача решена", Старец поднимает руку (za7).*

*5. Подсказка: "Чертеж на пергаменте — два подобных прямоугольных треугольника. Катет одного из треугольников составлен из дерева, а второго — из палки (zs5).*

*(5, 6). Если опять вводится (e1) неверный ответ (!x3), то появляется надпись: "В решении этой задачи используется правило подобия треугольников". Старец сердит (za8).*

*(5, 7). Если пользователь ввел (e1) верный ответ (x3) после подсказки, появляется надпись об успешном решении задачи. Старец поднимает руку (za7).*

*6. Выводится верное решение. Оно представляет собой описанный выше чертеж с треугольниками, но дополнительно к нему приводится процесс получения ответа. Возникает кнопка "Дальше" (zs6).*

*(6, 7) Нажав на кнопку "Дальше" (e2), переходим к заключительной сцене.*

*7. Заключительная сцена: "Ответ получен". Старец улыбается. Появляются кнопки "Следующая задача", "Повторить задачу" (zs7).*

*(7, 0) Нажатие на кнопку "Следующая задача" (e5) приводит к выгрузке текущей задачи и загрузке следующей (z1).*

*(7, 1) По нажатию на кнопку "Повторить задачу" (e6) генерируются новые значения начальных условий (z2). Вновь выводится надпись: "Как можно измерить высоту дерева, используя законы геометрической оптики?" (za0).*

**Замечание.** На экране постоянно присутствует кнопка "Выход", нажатие на которую *(e4)* завершает работу флэш-плеера.

Для написания программы проведем дальнейшую формализацию задачи. Построим схему связей автомата, определяющую его интерфейс (*рис.* 1).

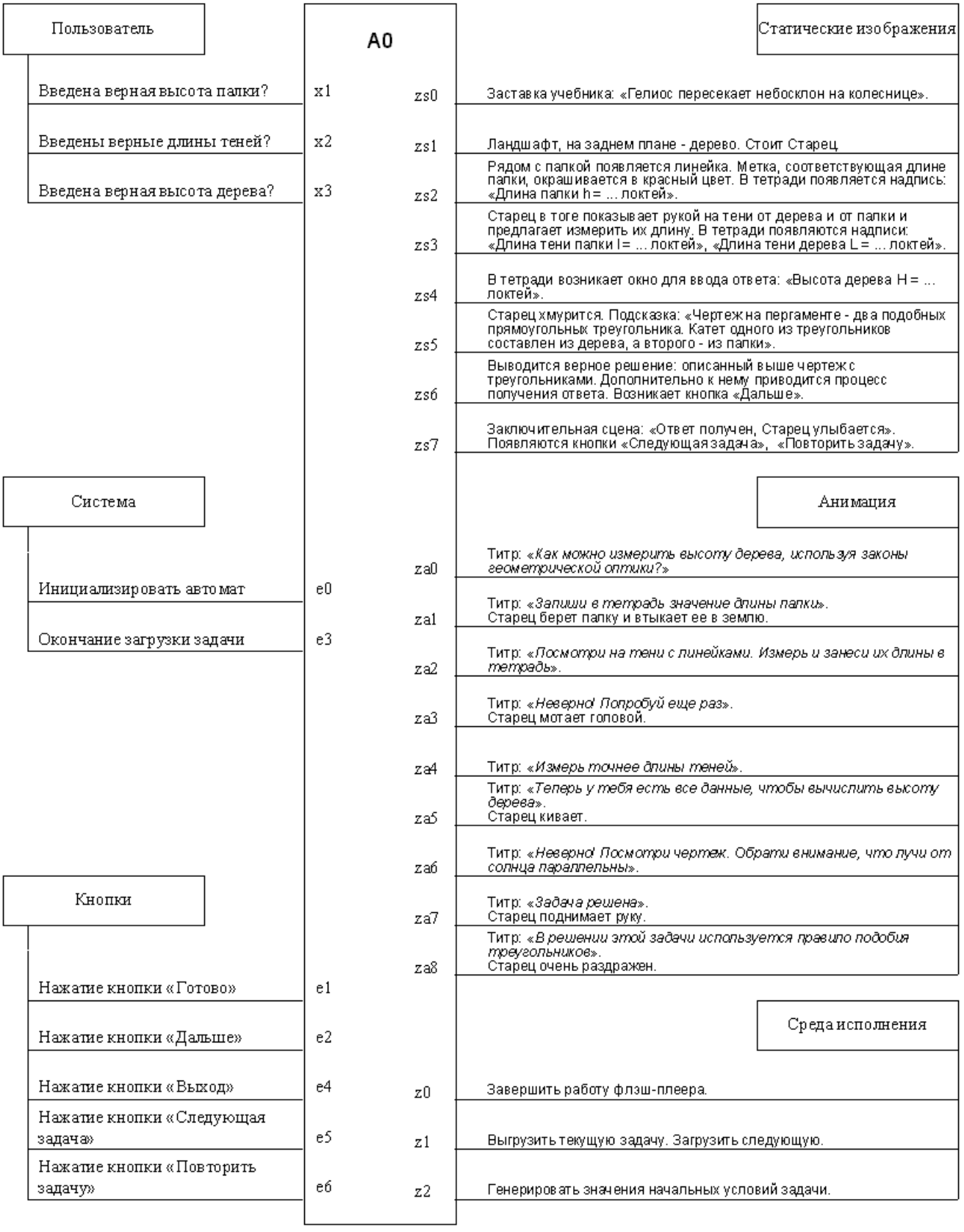



*Рис.* 1. Схема связей, описывающая интерфейс автомата

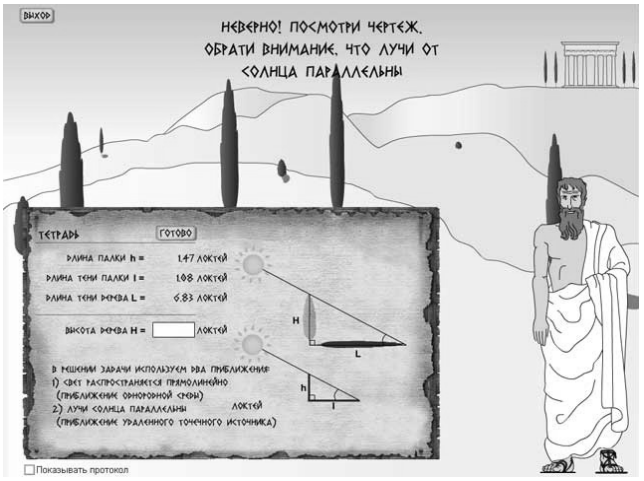



*Рис.* 2. Изображение после перехода из состояния 4 в состояние 5

В качестве примера приведем изображение, формируемое выходным действием *zs5* в состоянии 5 и выходным действием (*za6*) при переходе из состояния 4 в состояние 5 (*рис.* 2).

Граф переходов, построенный с помощью шаблона (*stencil*), описанного в работе [10], приведен на *рис.* 3.

Обратим внимание, что граф переходов является не "картинкой", а математической моделью, по которой текст фрагмента программы, реализующего логику задачи, строится формально и изоморфно. Этот текст на языке *ActionScript* приведен в Листинге 1 (fla-файл и полный комплект упоминаемых листингов размещены на сайте "Информатики" ***http://inf.1september.ru*** в разделе ***"Download"***).

Для окончательного построения программы, реализующей автомат, необходимо разработать тексты

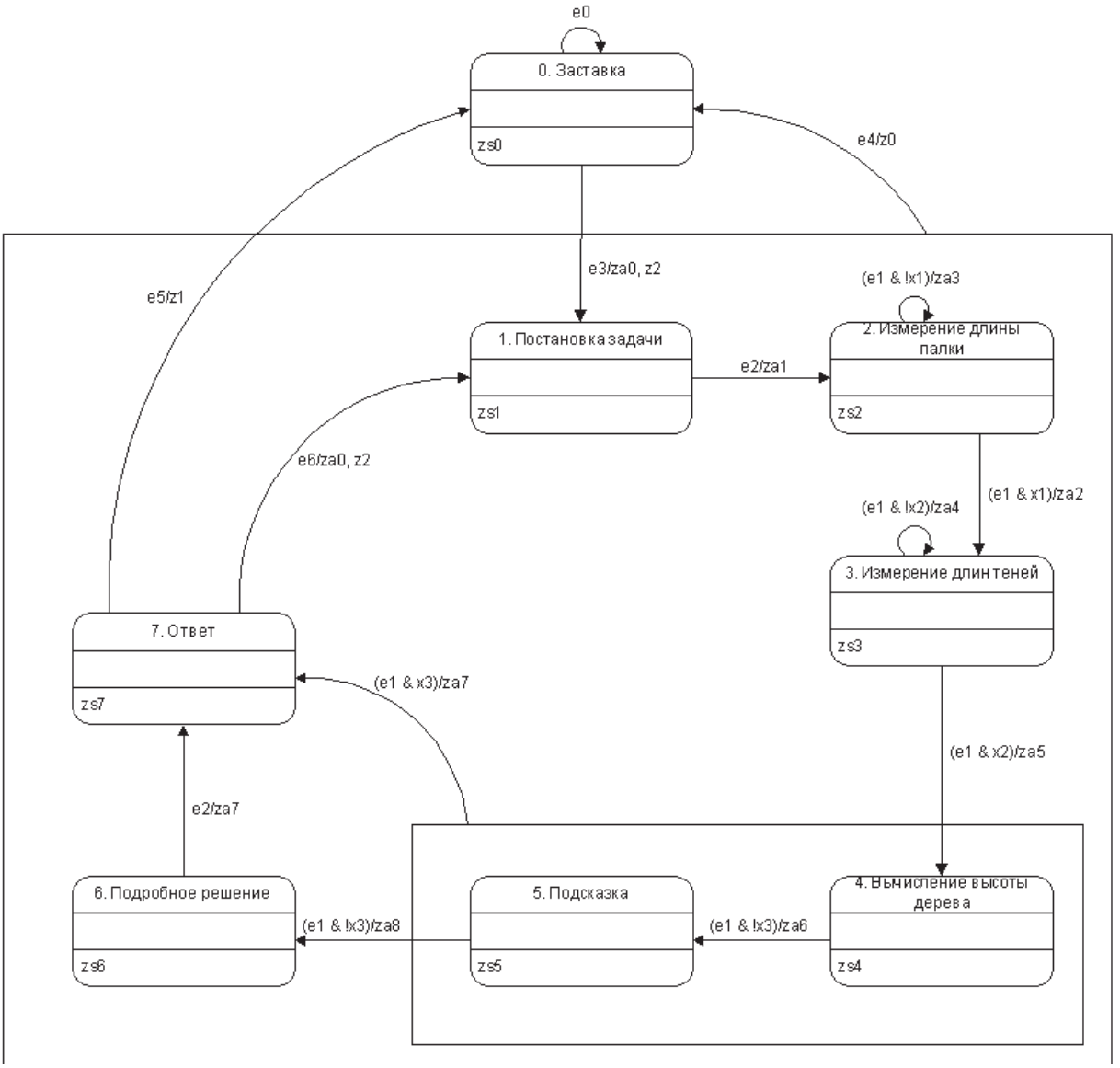



*Рис.* 3. Граф переходов, реализующий логику задачи

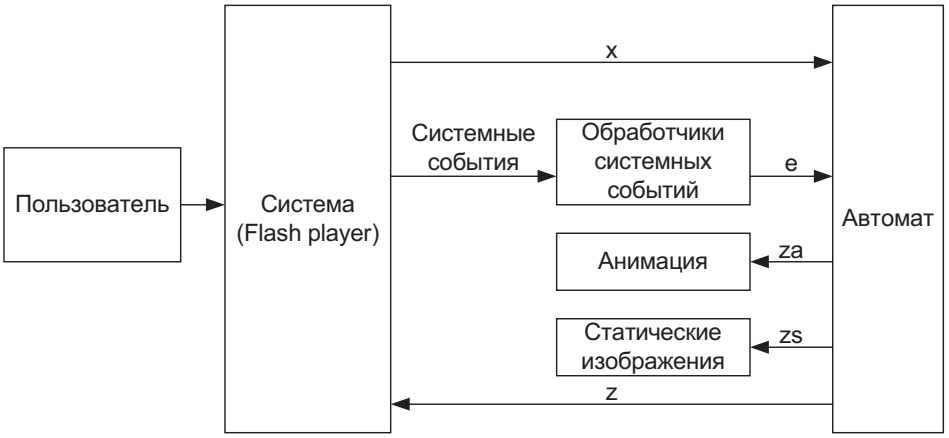



*Рис.* 4. Структурная схема программы

подпрограмм входных переменных, обработчиков событий и выходных действий. Так как в перечисленных подпрограммах практически отсутствует логическая обработка, они реализуются традиционным путем (Листинги 2, 3).

Структурная схема программы приведена на *рис.* 4.

Обработчик системного события запускается в связи с этим событием, например, от устройства "мышь" или клавиатуры, выделяет это событие и вызывает с ним (в качестве аргумента) автоматную функцию.

Из изложенного следует, что при применении предлагаемого подхода логика в отличие от традиционного подхода не рассредоточена по обработчикам событий, а централизована, что резко упрощает отладку, которую в работе [9] предлагается проводить с помощью протоколирования, выполняемого в терминах автоматов. Листинг 4 содержит модуль программы, обеспечивающий протоколирование, а пример протокола приведен в Листинге 5. Этот модуль, вызываемый из автомата, обработчиков событий и функций выходных действий, для упрощения структурной схемы на *рис.* 4 не показан. При отладке возможен вывод протоколов на экран (*рис.* 5). Для отключения и включения вывода протоколов на экран можно использовать флажок в левом нижнем углу экрана.

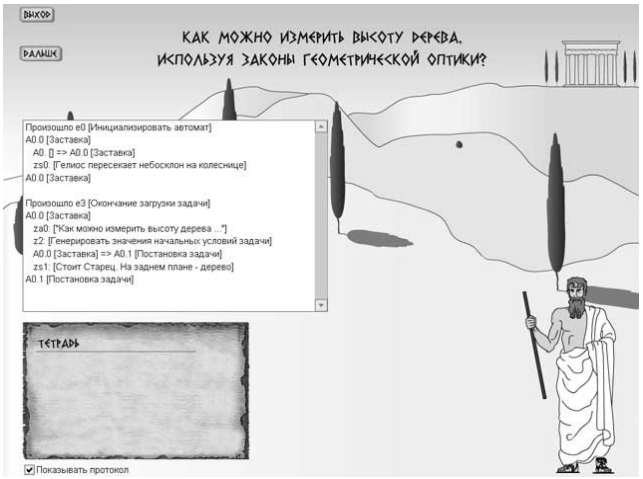



*Рис.* 5. Вывод протоколов на экран

## 4. Заключение

Подход, предлагаемый в настоящей работе, позволяет использовать автоматы при спецификации, в тексте программы и при протоколировании.

Таким образом, в настоящей работе предложена новая технология программирования анимации на языке *ActionScript*.

Указанные выше листинги программ приведены ниже.

Работа выполнена в лаборатории *Технологии программирования*, организованной *СПбГУ ИТМО* и центром разработки корпорации *Borland*.

## Литература

1. *Тарасов И.А.* Основы программирования в OpenGL. М.: Горячая линия — телеком, 2001.

2. *Гончаров Д., Салихов Т.* Книга DirectX 7.0 для программистов. СПб.: Питер, 2001.

3. *Маэстри Дж.* Компьютерная анимация персонажей. СПб.: Питер, 2001.

4. *Рейнхардт Р., Ленц Дж.* Flash 5. Библия пользователя. М.: Вильямс, 2001.

5. *Туйкин М.* CeBIT 2002 // Программист, 2002, № 4.

6. *Сандерс Б.* Flash ActionScript. СПб.: Питер, 2001.

7. *Степанова Г.Н., Степанов А.П.* Сборник вопросов и задач по физике. СПб.: Основная школа, 2001.

8. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

9. *Шалыто А.А., Туккель Н.И.* SWITCH-технология — автоматный подход к созданию программного обеспечения "РЕАКТИВНЫХ" систем // Программирование, 2001, № 5. ***http://is.ifmo.ru.*** Раздел "Статьи".

10. *Головешин А.* Конвертер Visio2Switch. ***http://is.ifmo.ru/progeny/visio2switch/***.

## Листинг 1. Реализация логики задачи

**Файл** *automat.as*

```
#include "log.as"
#include "animation.as"
#include "static.as"
#include "out.as"
#include "var.as"
iProblem2.A0 = function(e) {
    if (Y0 == undefined)
        Y0 = 0;
    else
        Yold = Y0;
    switch (Y0)
    {
        case 0: logBegin("A0", "0", "Заставка");
            if (e == 0)           {               Y0=0;}
            if (e == 3)           {za0(); z2();   Y0=1;}
        break;
        case 1: logBegin("A0", "1", "Постановка задачи");
            if (e == 2)           {za1();         Y0=2;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 2: logBegin("A0", "2", "Измерение длины палки");
            var _x1 = x1();
            if (e == 1 &&  _x1)   {za2();         Y0=3;}
            else
            if (e == 1 && !_x1)   {za3();              }
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 3: logBegin("A0", "3", "Измерение длин теней");
            var _x2 = x2();
            if (e == 1 &&  _x2)   {za5();         Y0=4;}
            else
            if (e == 1 && !_x2)   {za4();              }
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 4: logBegin("A0", "4", "Вычисление высоты дерева");
            var _x3 = x3();
            if (e == 1 &&  _x3)   {za7();         Y0=7;}
            else
            if (e == 1 && !_x3)   {za6();         Y0=5;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 5: logBegin("A0", "5", "Подсказка");
            var _x3 = x3();
            if (e == 1 &&  _x3)   {za7();         Y0=7;}
            else
            if (e == 1 && !_x3)   {za8();         Y0=6;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 6: logBegin("A0", "6", "Подробное решение");
            if (e == 2)           {za7();         Y0=7;}
            else
            if (e == 4)           {z0();          Y0=0;}
        break;
        case 7: logBegin("A0", "7", "Ответ");
            if (e == 4)          {z0();          Y0=0;}
            else
            if (e == 5)          {z1();          Y0=0;}
            else
            if (e == 6)          {za0(); z2();   Y0=1;}
        break;
    }
    if (Yold != Y0) {
        switch (Y0) {
            case 0: logTrans("A0", Yold, "0", "Заставка");
                zs0();
            break;
```

```
            case 1: logTrans("A0", Yold, "1", "Постановка задачи");
                zs1();
            break;
            case 2: logTrans("A0", Yold, "2", "Измерение длины палки");
                zs2();
            break;
            case 3: logTrans("A0", Yold, "3", "Измерение длин теней");
                zs3();
            break;
            case 4: logTrans("A0", Yold, "4", "Вычисление высоты дерева");
                zs4();
            break;
            case 5: logTrans("A0", Yold, "5", "Подсказка");
                zs5();
            break;
            case 6: logTrans("A0", Yold, "6", "Подробное решение");
                zs6();
            break;
            case 7: logTrans("A0", Yold, "7", "Ответ");
                zs7();
            break;
        }
    }
    logEnd("A0",Y0);
};
```

## Листинг 2. Реализация выходных действий

**Файл *static.as***

```
iProblem2.zs0 = function() {
    logStaticOut(0 ,"Гелиос пересекает небосклон на колеснице");
    ShowButtons(false,false,false,false,false); // Спрятать все кнопки
    _root.iPreLoader._visible = true;           // Показать предзагрузчик
    _alpha = 0;                                 // Скрыть загружаемые объекты
};
iProblem2.zs1 = function() {
    logStaticOut(1 ,"Стоит Старец. На заднем плане — дерево");
    _root.iPreLoader._visible = false;          // Скрыть предзагрузчик
    _alpha = 100;                               // Показывать загружаемые объекты
    ShowButtons(true,true,false,false,false);   // Показывать кнопки "Дальше" и "Выход"
    ShowMaster("WithStick");                    // Показать Старца с палкой
    ShowStick(false,false,false,false);         // Спрятать воткнутую палку
    ShowTree(true,true,false,false);            // Показать дерево и его тень
    ShowWorkBook("Empty");                      // Показать пустую тетрадь
};
iProblem2.zs2 = function() {
    logStaticOut(2 ,"Линейка рядом с палкой");
    ShowButtons(false,true,false,false,false);  // Показывать кнопку "Выход"
    ShowMaster("WithOutStick");                 // Показать Старца без палки
    ShowStick(true,true,true,false);         // Показать воткнутую палку с тенью и линейкой
    ShowWorkBook("StickHeight");          // Показать тетрадь с полем для ввода длины палки
};
iProblem2.zs3 = function() {
    logStaticOut(3 ,"Линейки рядом с тенями");
    ShowMaster("PointingOnShadow");           // Показать Старца, указывающего на тени
    ShowStick(true,true,false,true);            // Показать палку с тенью и линейкой у тени
    ShowTree(true,true,false,true);             // Показать дерево с тенью и линейкой у тени
    ShowWorkBook("ShadowsLengths");        // Показать тетрадь с полем для ввода длин теней
};
iProblem2.zs4 = function() {
    logStaticOut(4 ,"Линейки убраны");
    ShowMaster("WithOutStick");           // Показать Старца без палки
    ShowStick(true,true,false,false);     // Показать воткнутую палку с тенью
    ShowTree(true,true,false,false);      // Показать дерево с тенью
    ShowWorkBook("TreeHeight");   // Показать тетрадь с приглашением ввести высоту дерева
};
iProblem2.zs5 = function() {
    logStaticOut(5 ,"Старец увеличен. В тетради чертеж");
    ShowMaster("Torso");                        // Показать увеличенного Старца
    ShowWorkBook("Scheme");                     // Показать чертеж
};
```

```
iProblem2.zs6 = function() {
    logStaticOut(6 ,"Старец недоволен. Решение в тетради");
    ShowButtons(true,true,false,false,false);   // Показать кнопки "Дальше" и "Выход"
    ShowMaster("TorsoAngry");                   // Показать озлобленного Старца
    ShowWorkBook("Formula");                    // Показать вывод решения
};
iProblem2.zs7 = function() {
    logStaticOut(7 ,"Старец радуется");
    ShowButtons(false,true,true,true,false);    // Показать кнопки
    ShowMaster("Happy");                        // Показать счастливого Старца
    ShowStick(true,true,false,false);           // Показать палку с тенью
    ShowTree(true,true,true,false);             // Показать дерево с линейкой и тенью
    ShowWorkBook("Answere");                    // Показать в тетради правильный ответ
};
```

**Файл *animation.as***

```
iProblem2.za0 = function() {
    logAnimationOut(0, "\"Как можно измерить высоту дерева ...\"");
    ShowComment("TreeMeasureProblem");          // Титр с постановкой задачи
};
iProblem2.za1 = function() {
    logAnimationOut(1,"\"Запиши в тетрадь значение длины палки\". Старец втыкает палку");
    ShowComment("StickHeight");            // Титр: "Запиши в тетрадь значение длины палки"
    iWorkBook.txfStickHeight = "";      // Очистить поле ввода длины палки
};
iProblem2.za2 = function() {
    logAnimationOut(2,"\"Посмотри на тени с линейками ...\"");
    ShowComment("ShadowsLengths");              // Титр: "Посмотри на тени с линейками ..."
    iWorkBook.txfStickShadowLength = "";        // Очистить поле ввода длины тени от палки
    iWorkBook.txfTreeShadowLength = "";         // Очистить поле ввода длины тени от дерева
};
iProblem2.za3 = function() {
    logAnimationOut(3,"\"Неверно! Попробуй еще раз\". Старец мотает головой");
    ShowComment("WrongHeight");                 // Титр: "Неверно! Попробуй еще раз".
    AnimateMaster("HeadShake");                 // Старец мотает головой
    iWorkBook.txfStickHeight = "";              // Очистить поле ввода длины палки
};
iProblem2.za4 = function() {
    logAnimationOut(4,"\"Измерь точнее длины теней\". Старец мотает головой");
    ShowComment("WrongLengths");                // Титр: "Измерь точнее длины теней".
    AnimateMaster("HeadShake");                 // Старец мотает головой
    if (!CheckData(iWorkBook.txfStickShadowLength, StickShadowLength(), 2))
        iWorkBook.txfStickShadowLength = "";    // Очистить поле ввода длины тени от палки
    if (!CheckData(iWorkBook.txfTreeShadowLength, TreeShadowLength(), 2))
        iWorkBook.txfTreeShadowLength = "";   // Очистить поле ввода длины тени от дерева
};
iProblem2.za5 = function() {
    logAnimationOut(5,"\"Теперь у тебя есть все данные ...\". Старец кивает");
    ShowComment("TreeHeight");                  // Титр: "Теперь у тебя есть все данные ..."
    AnimateMaster("HeadNod");                // Старец кивает
    iWorkBook.txfTreeHeight = "";            // Очистить поле ввода высоты дерева
};
iProblem2.za6 = function() {
    logAnimationOut(6,"\"Неверно. Посмотри чертеж ...\"");
    ShowComment("WatchScheme");              // Титр: "Неверно. Посмотри чертеж ...".
    iWorkBook.txfTreeHeight = "";            // Очистить поле ввода высоты дерева
};
iProblem2.za7 = function() {
    logAnimationOut(7,"\"Задача решена\". Старец поднимает руку");
    ShowComment("ProblemSolved");               // Титр: "Задача решена"
    AnimateMaster("HandUp");                    // Старец поднимает руку
};
iProblem2.za8 =function() {
    logAnimationOut(8,"\"В решении этой задачи используется ...\". Старец очень раздражен");
    ShowComment("WatchFormula");          // Титр: "В решении этой задачи используется ..."
    AnimateMaster("Angry");            // Старец очень раздражен.
    iWorkBook.txfTreeHeight = "";      // Очистить поле ввода высоты дерева
};
```

**Файл *out.as***

```
iProblem2.z0 = function() {
    logOut("0","Завершить работу флэш-плеера");
    fscommand("quit","");               // Передать плееру команду выход
};
iProblem2.z1 = function() {
    logOut("1","Выгрузить текущую задачу. Загрузить следующую");
    _root.NextProblem();                 // Выгрузить текущую задачу, загрузить следующую
};
iProblem2.z2 = function() {
    logOut("2","Генерировать значения начальных условий задачи");
    InitConstants();                    // Инициализировать используемые константы
    InitData();                         //   и переменные
}
```

## Листинг 3. Реализация входных переменных и обработчиков событий

**Файл *var.as*** — входные переменные

```
iProblem2.x1 = function() {
    uSH = iWorkBook.txfStickHeight;
     SH = StickHeight;
    res = CheckData(uSH,SH,2);
    logVar("1", res?1:0, "Введена верная высота палки?");
    return res;
};
iProblem2.x2 = function() {
    uSL = iWorkBook.txfStickShadowLength;
     SL = StickShadowLength();
    uTL = iWorkBook.txfTreeShadowLength;
     TL = TreeShadowLength();
    res = CheckData(uSL, SL, 2) && CheckData(uTL, TL, 2);
    logVar("2", res?1:0, "Введены верные длины теней?");
    return res;
};
iProblem2.x3 = function() {
    uTH = iWorkBook.txfTreeHeight;
     TH = TreeHeight;
    res = CheckData(uTH, TH, 1);
    logVar("3", res?1:0, "Введена верная высота дерева?");
    return res;
};
```

**Файл *events.as*** — обработчики событий

```
automatInit = function() {
    logEvent(0,"Инициализировать автомат");
    A0(0);
}
// Загрузка задачи окончена [e3]
loadingFinished = function() {
    logEvent(3,"Окончание загрузки задачи");
    A0(3);
};
// Сообщения от графических кнопок [e1,e2,e4,e5,e6]
onMouseUp = function() {
    var xm = _root._xmouse;
    var ym = _root._ymouse;

    if (iWorkBook.iOK.hitTest(xm,ym,true) && iWorkBook.iOK._visible) {
        logEvent(1,"Нажатие кнопки \"Готово\"");
        A0(1);
    } else
    if (iButNext.hitTest(xm,ym,true) && iButNext._visible) {
        logEvent(2,"Нажатие кнопки \"Дальше\"");
        A0(2);
    } else
    if (iButExit.hitTest(xm,ym,true) && iButExit._visible) {
        logEvent(4,"Нажатие кнопки \"Выход\"");
        A0(4);
    } else
    if (iButNextProblem.hitTest(xm,ym,true) && iButNextProblem._visible) {
        logEvent(5,"Нажатие кнопки \"Следующая задача\"");
```

```
        A0(5);
    } else
    if (iButReplayProblem.hitTest(xm,ym,true) && iButReplayProblem._visible) {
        logEvent(6,"Нажатие кнопки \"Повторить задачу\"");
        A0(6);
    }
};
// Сообщения от клавиатуры [e1,e2]
onKeyUp = function () {
    switch (Key.getCode())
    {
    case Key.ENTER:
        logEvent(1,"Нажатие кнопки \"Готово\"");
        A0(1);
    break;

    case Key.SPACE:
        logEvent(2,"Нажатие кнопки \"Дальше\"");
        A0(2);
    break;
    }
};
// Обрабатывать сообщения от клавиатуры
Key.addListener(this);
```

## Листинг 4. Реализация функций протоколирования

**Файл *log.as***

```
iProblem2.logToWin          = true;
iProblem2.doLogBegin        = true;
iProblem2.doLogTrans        = true;
iProblem2.doLogEnd          = true;
iProblem2.doLogOut          = true;
iProblem2.doLogEvent        = true;
iProblem2.doLogAnimationOut = true;
iProblem2.doLogStaticOut    = true;
iProblem2.doLogVar          = true;
iProblem2.yNames = new Array();
// Для вывода протоколов может быть использовано окно отладчика редактора Macromedia Flash
// или окно, созданное средствами стандартных компонент и доступное не только во время
// отладки. Для практических применений технологии рекомендуется пользоваться окном
// отладчика (logToWin == false). Здесь используется компонентное окно, оно добавлено
// из соображений наглядности протоколирования.
iProblem2.log = function(str) {
    if (logToWin) {
        lstLog.addItem(str);
        lstLog.setScrollPosition(Math.max(0,lstLog.getLength() - lstLog.getRowCount()));
    }
    else
        trace(str);
};
iProblem2.logBegin = function(A, Y, yName) {
    if (doLogBegin) {
        log(A+"."+Y+" ["+yName+"]");
        if (yNames[Y] == undefined)
            yNames[Y] = yName;
    }
};
iProblem2.logTrans = function(A, Y_old, Y, yName) {
    if (doLogTrans) {
        log("   "+A+"."+Y_old+" ["+yNames[Y_old]+"] => "+A+"."+Y+" ["+yName+"]");
        if (yNames[Y] == undefined)
            yNames[Y] = yName;
    }
};
iProblem2.logEnd = function(A, Y) {
    if (doLogEnd) {
        log(A+"."+Y+" ["+yNames[Y]+"]");
        log("");
    }
};
```

```
iProblem2.logOut = function(z, descr) {
    if (doLogOut)
        log("   z"+z+": ["+descr+"]");
};
iProblem2.logEvent = function(e, descr) {
    if (doLogEvent)
        log("Произошло e"+e+" ["+descr+"]");
};
iProblem2.logAnimationOut = function(z, descr) {
    if (doLogAnimationOut)
        logOut("a"+z, descr);
};
iProblem2.logStaticOut = function(z, descr) {
    if (doLogStaticOut)
        logOut("s"+z, descr);
};
iProblem2.logVar = function(x, value, descr) {
    if (doLogVar)
        log("   x"+x+"="+value+": ["+descr+"] — " + (value?"ДА":"НЕТ"));
};
```

## Листинг 5. Пример протокола

Приведенный протокол соответствует случаю, когда пользователь решил задачу, не воспользовавшись ни одной подсказкой.

```
Произошло e0 [Инициализировать автомат]
A0. []
   A0. [] => A0.0 [Заставка]
   zs0: [Гелиос пересекает небосклон на колеснице]
A0.0 [Заставка]
Произошло e3 [Окончание загрузки задачи]
A0.0 [Заставка]
   za0: ["Как можно измерить высоту дерева ..."]
   z2: [Генерировать значения начальных условий задачи]
   A0.0 [Заставка] => A0.1 [Постановка задачи]
   zs1: [Стоит Старец. На заднем плане — дерево]
A0.1 [Постановка задачи]
Произошло e2 [Нажатие кнопки "Дальше"]
A0.1 [Постановка задачи]
   za1: ["Запиши в тетрадь значение длины палки". Старец втыкает палку]
   A0.1 [Постановка задачи] => A0.2 [Измерение длины палки]
   zs2: [Линейка рядом с палкой]
A0.2 [Измерение длины палки]
Произошло e1 [Нажатие кнопки "Готово"]
A0.2 [Измерение длины палки]
   x1=1: [Введена верная высота палки?] — ДА
   za2: ["Посмотри на тени с линейками ..."]
   A0.2 [Измерение длины палки] => A0.3 [Измерение длин теней]
   zs3: [Линейки рядом с тенями]
A0.3 [Измерение длин теней]
Произошло e1 [Нажатие кнопки "Готово"]
A0.3 [Измерение длин теней]
   x2=1: [Введены верные длины теней?] — ДА
   za5: ["Теперь у тебя есть все данные ...". Старец кивает]
   A0.3 [Измерение длин теней] => A0.4 [Вычисление высоты дерева]
   zs4: [Линейки убраны]
A0.4 [Вычисление высоты дерева]
Произошло e1 [Нажатие кнопки "Готово"]
A0.4 [Вычисление высоты дерева]
   x3=1: [Введена верная высота дерева?] — ДА
   za7: ["Задача решена". Старец поднимает руку]
   A0.4 [Вычисление высоты дерева] => A0.7 [Ответ]
   zs7: [Старец радуется]
A0.7 [Ответ]
Произошло e4 [Нажатие кнопки "Выход"]
A0.7 [Ответ]
   z0: [Завершить работу флэш-плеера]
   A0.7 [Ответ] => A0.0 [Заставка]
   zs0: [Гелиос пересекает небосклон на колеснице]
A0.0 [Заставка]
```

# В ближайших номерах «Жаркого лета»

### Портретная галерея для кабинета информатики

Следующий номер будет целиком (!) составлен из 44 портретов, которыми вы сможете оформить кабинет информатики. Все портреты выполнены специально для обеспечения наилучшего качества сканирования и ксерокопирования (вам не придется разрывать газету). Все портреты снабжены короткими информационными блоками.

### "Началка": информатика в стихах

Иногда говорят: номер — просто песня. А этим летом мы предложим подписчикам не песню, а целую поэму! И не одну! Надеемся, этот замечательный сборник станет незаменим для всех, кто учит информатике маленьких детей.




**ИЗДАТЕЛЬСКИЙ ДОМ «ПЕРВОЕ СЕНТЯБРЯ»**
**главный редактор – А.С. Соловейчик**

ГАЗЕТЫ ИЗДАТЕЛЬСКОГО ДОМА

**Первое сентября**
гл. ред. — Е.В. Бирюкова, индекс подписки — 32024;
**Английский язык**
гл. ред. — Е.В. Громушкина, индекс подписки — 32025;
**Библиотека в школе**
гл. ред. — О.К. Громова, индекс подписки — 33376;
**Биология**
гл. ред. — Н.Г. Иванова, индекс подписки — 32026;
**География**
гл. ред. — О.Н. Коротова, индекс подписки — 32027;
**Дошкольное образование**
гл. ред. — М.С. Аромштам, индекс подписки — 33373;
**Здоровье детей**
гл. ред. — Н.В. Сёмина, индекс подписки — 32033;
**Информатика**
гл. ред. — С.Л. Островский, индекс подписки — 32291;
**Искусство**
гл. ред. — М.Н. Сартан, индекс подписки — 32584;
**История**
гл. ред. — А.Л. Савельев, индекс подписки — 32028;
**Литература**
отв. сек. — С.Ф. Дмитренко, индекс подписки — 32029;
**Математика**
и. о. гл. ред. — Л.О. Рослова, индекс подписки — 32030;
**Начальная школа**
гл. ред. — М.В. Соловейчик, индекс подписки — 32031;
**Немецкий язык**
гл. ред. — М.Д. Бузоева, индекс подписки — 32292;
**Русский язык**
гл. ред. — Л.А. Гончар, индекс подписки — 32383;
**Спорт в школе**
гл. ред. — О.М. Леонтьева, индекс подписки — 32384;
**Управление школой**
гл. ред. — Я.А. Сартан, индекс подписки — 32652;
**Физика**
гл. ред. — Н.Д. Козлова, индекс подписки — 32032;
**Французский язык**
гл. ред. — Г.А. Чесновицкая, индекс подписки — 33371;
**Химия**
гл. ред. — О.Г. Блохина, индекс подписки — 32034;
**Школьный психолог**
гл. ред. — И.В. Вачков, индекс подписки — 32898.

**Гл. редактор** С.Л. Островский
**Зам. гл. редактора** А.И. Сенокосов
**Редакция** Е.В. Андреева, Д.М. Златопольский (редактор вкладки "В мир информатики"), Л.Н. Картвелишвили, С.Б. Кишкина, Н.П. Медведева, Ю.А. Первин (редактор вкладки "Началка")
**Корректор** Е.Л. Володина **Дизайн и верстка** Н.И. Пронская




**Адрес редакции и издателя:**
Киевская, 24, Москва, 121165
тел. 249-48-96
Отдел рекламы: 249-98-70

**Учредитель: ООО "Чистые пруды"**

Зарегистрировано в Министерстве РФ по делам печати. ПИ № 77-7230 от 12.04.2001.
Отпечатано в ОИД "Медиа-Пресса", ул. Правды, 24, Москва, ГСП-3, А-40, 125993
Тираж 6500 экз.
Срок подписания в печать по графику 12.05.2006.
Номер подписан 12.05.2006.
Заказ № 615511
Цена свободная

**ИНДЕКС ПОДПИСКИ**
**для индивидуальных подписчиков 32291**
**комплекта изданий 32744**

**Тел.: (095) 249-31-38, 249-33-86. Факс (095)249-31-84**

**Internet: inf@1september.ru**
**WWW: http://www.1september.ru**

**ИЗДАТЕЛЬСКАЯ ПОДПИСКА Тел.: (495) 249-47-58 E-mail: podpiska@1september.ru**