

89  
А. Жуков

3 00-40  
з а 175-6  
~~номер~~

# ИЗУЧАЕМ DELPHI



 ПИТЕР®

Санкт-Петербург • Москва • Харьков • Минск

2000

905/1  
АБ/1

к нижнему (*lower*) регистру, то есть станут маленьими. Соответственно в процедуре TForm1.Button1Click нужно отследить эту поправку, записав слово «привет» также с маленькой буквы:

```
if Edit1.Text = 'привет!'
then Label1.Caption := 'Прриветик!';
```

Внеся это изменение, запустите еще раз программу. Теперь «Томагочи» не будет обращать внимание на то, большими или маленькими буквами набрана обращенная к нему фраза.

### Почему «Томагочи» не реагирует на фразу: «Привет»?

Потому что в конце приветствия нет восклицательного знака. Он не прореагирует также на фразы « Привет!» (слово начинается с пробела), «Привет !» (между словом и восклицательным знаком вставлен пробел). Нам это понять трудно, но компьютер — жуткий формалист и все воспринимает буквально. Можно расширить кругозор «Томагочи», поместив, например, вместо прежнего условного предложения более универсальное:

```
if (Edit1.Text = 'привет!') or
(Edit1.Text = 'привет') or
(Edit1.Text = 'привет !') or
(Edit1.Text = 'здравствуй!') or
(Edit1.Text = 'Салют !') or
(Edit1.Text = 'Hi !') // этот ряд можно
                     // продолжать
then Label1.Caption := 'Прриветик!';
```

### Как сделать так, чтобы фраза «Прриветик!» не застревала во рту «Томагочи»?

Этот дефект устраняется с помощью еще одного таймера. Разместим в форме компонент Timer2 со стан-

дартным интервалом срабатывания 1000 миллисекунд. Предварительно отключите его, установив в паспортном свойстве Enabled значение, равное false. В программе обработчика TForm1.Button1Click рядом с оператором Timer1.Enabled := true; разместим аналогичный: Timer2.Enabled := true;

Затем в новой процедуре TForm1.Timer2Timer (ее заготовку нужно открыть, два раза щелкнув мышью на компоненте Timer2 в форме), пишем:

```
Label1.Caption := '';
// присвоение пустой строки
// приводит к очистке застрявшей
// во рту «Томагочи» фразы
Timer2.Enabled := false; // отключение
                         // таймера 2
```

После всех исправлений проверьте работоспособность программы.

### Дальнейшее совершенствование программы

Умение произносить одну единственную фразу «Прриветик!» отнюдь не лестно характеризует умственные способности «Томагочи». Прежде чем обучить его другим премудростям, нам будет удобно завести некоторую вспомогательную переменную для хранения обращенной к «Томагочи» фразы.

**Эксперимент 3.** Перед словом **begin** процедуры TForm1.Button1Click вставьте следующие строки:

```
var
  s: string;
Что это такое?
```

### Раздел описания переменных

**var** — это начальные буквы английского слова *variables* — переменные. В нашей программе **var** обозначает специальный раздел, в котором мы должны описать все новые придуманные нами переменные, с которыми пока еще не знакома система Delphi. Прочитав их, компьютер выделит для хранения значений каждой переменной некое пустое хранилище — одну или несколько ячеек памяти в зависимости от *типа* переменной, который указывается после двоеточия. Тип *string* означает, что компьютер будет иметь дело со *строкой* символов.



*Принципиально важное соглашение языка Object Pascal: каждый объект, использованный в программе, должен быть предварительно описан. До сих пор описания всех объектов, с которыми мы имели дело, заботливо брала на себя специальная система Delphi, и мы не обращали на это внимания.*

Продолжим эксперимент. После того как мы представили Delphi нашу новую переменную *s*, ее нужно инициализировать, другими словами, присвоить ей некоторое значение. Это значение мы возьмем из окна *Edit1*. В начале процедуры *TForm1.Button1Click* запишем:

```
s := Edit1.Text;
```

Теперь слегка подправим условное предложение. Вместо

```
if Edit1.Text = 'привет!'
then Label1.Caption := 'Пряриветик!';
напишем:
if s = 'привет!'
```

```
then Label1.Caption := 'Пряриветик!';
```

Добавьте далее и другие предложения, например,

```
if s = 'как тебя
```

```
зовут?' then Label1.Caption := 'Гоша';
```

```
if s = 'сколько тебе лет?'
```

```
then Label1.Caption := 'надцать';
```

```
if s = 'как поживаешь?'
```

```
then Label1.Caption := 'Ничего';
```

```
if s = 'Тебе нравится программирование?'
```

```
then Label1.Caption := 'Очень!';
```

и им подобные. Чем больше вы запрограммируете фраз, тем более сообразительным будет казаться «Томагочи».

Слегка улучшим также процедуру работы с окном *Edit1*, поместив в конце после всех условных предложений две строки:

```
Edit1.Text := ''; // очистка окна Edit1
Edit1.SetFocus; // подготовка окна
// к вводу новой строки
// символов
```

После выполнения последней команды в окне *Edit1* появится курсор, который будет подмигивать, словно приглашая ввести в это окно новый текст.

Запустите программу и проверьте её работоспособность.

**Эксперимент 4.** Как говорил Козьма Прутков: «Нельзя объять необъятное», а в нашем случае — предусмотреть весь набор фраз, с которыми кому-то заблагорассудится обращаться к несчастному «Томагочи». Сейчас мы предусмотрим небольшую «хитринку» — она поможет «Томагочи» выкручиваться из некоторых затруднительных ситуаций. Идея такая: если в конце обращенной к «То-

«Томагочи» непонятной фразы стоит восклицательный знак, то маловероятно, что на эту фразу потребуется какой-то вразумительный ответ (все же между восклицательным и вопросительным знаками есть некоторое различие). В данном случае можно отделаться какой-нибудь ничего не значащей репликой.

Но как определить, что в конце фразы стоит восклицательный знак? Представим себе содержащуюся в строке *s* фразу как поезд, в каждом вагоне которого сидит отдельный символ. Все вагоны этого поезда пронумерованы: 1, 2, 3, ... (на самом деле, в этом поезде имеется вагон и с нулевым номером, но он не предназначен для перевозки букв). Номер последнего вагона мы узнаем, обратившись к функции *length(s)* (английское слово *length* — длина), имеющейся в библиотеке функций Object Pascal. Например, если в строке *s* находится фраза «какой хороший день!», то мы имеем дело с поездом, показанным на рис. 3.2.

поезд " **к а к о й** **х о р о ш и й** **д е н ы !**  
 с 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
 номер вагона

Рис. 3.2. Фраза в строке *s*

В данном случае *length(s) = 19*.

Как прочитать символ, находящийся в пятом вагоне поезда? — очень просто: *s[5]* (там сидит буква *й*). В десятом вагоне? — *s[10]* (там сидит буква *о*). В последнем вагоне? — *s[length(s)]* (там сидит восклицательный знак). Почему мы вместо конкретного номера последнего вагона 19 пишем громоздкую конструкцию

*length(s)*? Потому что эта конструкция годится для поезда произвольной длины, — как раз то, что нам нужно, ведь мы не можем заранее предугадать, сколько букв будет во входной фразе (сколько вагонов потребуется для их размещения).

Итак, к условным предложениям программы «Томагочи» добавляем еще одно предложение:

```
if (length(s) > 0) and (s[length(s)] = '!')
then Label1.Caption := 'Ну-ну';
// или какой-нибудь другой ответ
// в этом роде
```

Запустите программу и убедитесь, что она работает правильно.

#### Как проверить, содержит ли фраза заданное слово?

Если кто-то обращается к «Томагочи» с предложением «А я...», то и наш воспитанник может ответить в том же духе: «А я умею моргать глазками», или «А у меня 64 мегабайта оперативной памяти». Распознавать некоторую группу символов в предложении лучше всего с помощью специальной функции языка *pos(s1,s2)*, которая выдает номер первого совпадающего символа, если строка *s1* содержится среди символов строки *s2*, и 0 в противном случае. Например, словосочетание «А я» в обращенном к «Томагочи» предложении можно отследить с помощью следующего кода:

```
if pos('А я', s) > 0 then // соответствующая
                           // реплика «Томагочи»
```

**Эксперимент 5.** Получая длинную вереницу непонятных ему фраз, «Томагочи» будет только усердно хлопать глазками. У его собеседника может сложиться превратное впечатление, что «Томагочи» вообще не

способен вести диалог цивилизованным образом. Чтобы этого не случилось, предусмотрим более естественную реакцию, не вызывающую кривотолков.

Если фраза не знакома «Томагочи», то она отсутствует во всех условных предложениях нашей программы. Как это проверить? Здесь может пригодиться прием, связанный с использованием флагка — некоторого логического признака. В начале выполнения процедуры TForm1.Button1Click опустим флагок. Если в каком-то условном предложении анализируемая фраза обнаружится, поднимем флагок. В самом конце проверим, поднимался флагок или нет. Если не поднимался, значит, фраза оказалась неизвестной.

А вот как выглядит программная реализация этого способа. В уже знакомом нам разделе описания переменных var процедуры TForm1.Button1Click добавим еще одну строчку:

```
f: boolean;
```

**f** — это имя переменной, которая будет играть роль логического признака или флагка; **boolean** — специальный тип, его название связано с именем английского математика Джорджа Буля (1815–1864), разработавшего теоретические основы *алгебры логики*. В нашей программе тип **boolean** определяет допустимое множество значений переменной **f**: **true** и **false**.

Первым оператором процедуры TForm1.Button1Click теперь у нас будет

```
f := false; // опустим флагок
```

Его мы разместим после слова **begin** в любом месте перед условными предложениями. Каждое условное предложение переделаем по образцу реконструкции следующего предложения:

```
if s = 'привет!' then
begin
  Label1.Caption := 'Прриветик!';
  f := true; // поднимем флагок
end;
```

Здесь вместо прежнего *простого* оператора **Label1.Caption := 'Прриветик!'**; образован *составной оператор*, объединяющий с помощью операторных скобок **begin ... end**; два простых оператора в одно целое (один блок).

После титанической работы по переделке всех условных предложений в конце добавим новое условное предложение:

```
if not f then Label1.Caption := 'Ась ?';
```

Было бы неплохо, если бы в этот момент у «Томагочи» появилось импровизированное ухо (на рис. 3.3 у нашего героя еще откуда-то появилась ворона). Попробуйте самостоятельно запрограммировать такие эффекты.

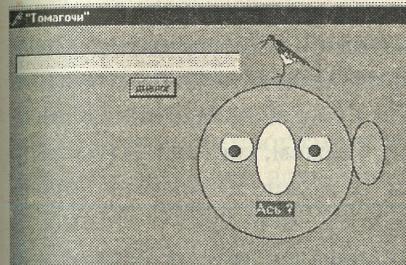


Рис. 3.3. Вид проекта после изменений

Несмотря на всю простоту одноступенчатого диалога, сценарии такого типа находят применение в серьезных коммерческих приложениях, например в автоматизированных справочных системах, предусматривающих

голосовое общение и мобильную связь. Безусловно, в таких системах применяются также специальные программы распознавания речи.

## Двухступенчатый диалог

— А ты почему не разговариваешь?  
 — Да я не знаю, о чём разговаривать, — говорит Мишка. — Это всегда так бывает: когда надо разговаривать, так не знаешь, о чём разговаривать, а когда надо разговаривать, так разговариваешь и разговариваешь...

*Н. Носов, «Телефон»*

Что должен делать компьютер, получив вопрос: «Сколько она стоит?»

Под местоимением *она* может скрываться все, что угодно: лягушка, пушка или фабрика по производству погремушек. Но если этот вопрос прозвучал в контексте предыдущего вопроса: «Где можно купить игрушку Томагочи?», то ясно, что он касается именно игрушки Томагочи, и на него можно ответить вполне конкретно. Итак, иногда приходится анализировать сразу два вопроса: данный и предыдущий. Первое, что приходит в голову при попытке организации двухступенчатого диалога, — воспользоваться уже знакомыми нам логическими флагками. Если, скажем, состоялся разговор о месте покупки игрушки, то можно поднять специальный флагок, а потом, когда речь пойдет о стоимости игрушки, — проверить, поднимался ли этот флагок или нет. Однако здесь мы сталкиваемся с одной принципиальной трудностью.

## Локальные и глобальные переменные программы

Как только происходит щелчок на кнопке Button1, операционная система отводит память для описанных в разделе var процедуры TForm1.Button1Click переменных. После этого начинают последовательно выполняться операторы процедуры, описанные в ее тексте. Завершив выполнение последнего оператора процедуры, компьютер освобождает выделенную для нее память — он «забывает» обо всех внутренних переменных процедуры, с которыми только что имел дело. Переменные, доступные только во время исполнения процедуры, называются *локальными (местными)* переменными данной процедуры. В них нельзя сохранить данные, которыми можно было бы воспользоваться при очередном запуске процедуры. Если в этом все же возникает необходимость, данные нужно «сдать на хранение» *глобальным (общим)* переменным, описанным *вне* других процедур и доступным всем другим процедурам, когда бы они ни исполнялись.

Существует несколько способов описания глобальных переменных. Давайте условимся, если это не будет оговорено особо, описывать все нужные нам глобальные переменные в разделе var, помещаемом в верхней части большого программного раздела, в котором создаются заготовки всех наших процедур. Как вы уже, наверное, успели заметить, все эти заготовки возникают в разделе implementation (исполнение).

Флагок, который может быть поднят во время одного прохода по операторам процедуры TForm1.Button1Click с тем, чтобы во время другого прохода на него обратили внимание, конечно же, должен быть глобальным признаком.

**Эксперимент 7.** В верхней части раздела `implementation` после служебного комментария `{$R *.DFM}` (сгенерированного Delphi) опишем глобальную переменную:

```
var
  flagTomagochi: boolean = false;
```

Для глобальных переменных разрешается непосредственно указывать начальное значение (в данном случае `false`). Обратите внимание, что это значение задается через знак равенства, а не с помощью оператора присваивания. Место операторов — на сцене, где выполняются какие-то действия, а сейчас пока объявляются лишь «действующие лица и исполнители».

К компании условных предложений процедуры `TForm1.Button1Click` добавим еще два:

```
if s = 'где можно купить Томагочи ?' then
begin
  flagTomagochi := true;
  Label1.Caption :=
    'В магазине «Детский мир»';
end;
if s = 'сколько она стоит ?' and flagTomagochi
then
begin
  Label1.Caption := 'Двадцать пять монет';
  flagTomagochi := false; // забываем
                        // предыдущий разговор
end;
```

Запустите программу и поэкспериментируйте с порядком задания вопросов.

Естественно ожидать, что следующим по уровню сложности будет трехступенчатый, далее — четырехступенчатый, ..., много-многоступенчатый диалог. Однако (такова уж особенность нашей речи) на практике

в большинстве случаев основную роль играет именно двухступенчатый диалог.

Обратим внимание на один «подводный камень», подкарауливающий разработчиков сценариев двухступенчатого диалога в системах событийного управления типа Delphi. Рассмотрим следующую модель диалога, в которой используются логические флаги `flagHow`, `flagWhy` (рис. 3.4).

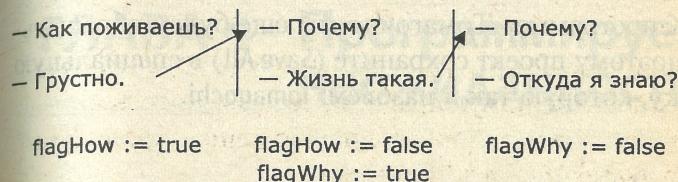


Рис. 3.4. Модель диалога с логическими флагами

Для второго ответа необходимы наличие вопроса «Почему?» и установка в `true` флага `flagHow`; для третьего ответа необходимы наличие вопроса «Почему?» и установка в `true` флага `flagWhy`. Если в программе обработку фраз расположить в естественном (приведенном на рис. 3.4) порядке, то в ответ на первый вопрос «Почему?» на экране сразу возникнет ответ «Откуда я знаю?», и складывается впечатление, что ответ «Жизнь такая» игнорируется. На самом деле здесь наблюдается эффект так называемого «квази-перескакивания».

— А видела ты Черепаху Квази?  
— Нет, — сказала Алиса. — Я даже не знаю, кто это такой.  
— Как же, — сказала Королева. — Это то, из чего делают квази-черепаший суп.

Льюис Кэрролл, «Алиса в Стране Чудес»

Ответ «Жизнь такая» выдается, но не замечается — не успеет наш «Томагочи» моргнуть глазками, как фраза тут же, в полном соответствии с логикой работы программы, заменяется на «Откуда я знаю». Чтобы этого не произошло, нужно «телегу поставить впереди лошади» — в данном случае поменять местами в тексте программы обработку третьей и второй фразы сценария.

**Эксперимент 8.** Испытайте указанные сценарии в своем проекте.

К программе «Томагочи» мы еще будем обращаться, поэтому проект сохраните (Save All) в специальную папку, которую так и назовем: Tomagochi.

## ГЛАВА 4 Программируем неожиданность

