Процедура

Процедура — это разновидность подпрограммы. Обычно подпрограмма реализуется как процедура в двух случаях:

- когда подпрограмма не возвращает в основную программу никаких данных. Например, вычерчивает график в диалоговом окне;
- когда подпрограмма возвращает в вызвавшую ее программу больше чем одно значение. Например, подпрограмма, которая решает квадратное уравнение, должна вернуть в вызвавшую ее программу два дробных числа — корни уравнения.

Объявление процедуры

В общем виде объявление процедуры выглядит так: procedure Имя (var параметр1: тип1; ... var параметрК: типК) ; **var**

// здесь объявление локальных переменных

begin

// здесь инструкции процедуры

end;

где:

- procedure — зарезервированное слово языка Delphi, обозначающее, что далее следует объявление процедуры;
- имя — имя процедуры, которое используется для вызова процедуры;
- параметр К — формальный параметр, переменная, которая используется в инструкциях процедуры. Слово var перед именем параметра не является обязательным. Однако если оно стоит, то это означает, что в инструкции вызова процедуры фактическим параметром обязательно должна быть переменная.

Параметры процедуры используются для передачи данных в процедуру, а также для возврата данных из процедуры в вызвавшую ее программу.

В качестве примера в листинге 6.5 приведена процедура решения квадратного уравнения (которое в общем виде записывается так: ax^2 + Ьх+ с = 0). У процедуры шесть параметров: первые три предназначены для передачи в процедуру исходных данных — коэффициентов уравнения; параметры xi и х2 используются для возврата результата — корней уравнения; параметр ok служит для передачи информации о том, что решение существует.

Листинг 6.5. Процедура SgRoot

```
// решает квадратное уравнение

procedure SqRoot(a,b,c : real;

var xl,x2 : real;

var ok : boolean);

{ a,b,c — коэффициенты уравнения x1,x2 — корни уравнения ok = True
— решение есть ok = False — решения нет }

var

d : real; // дискриминант

begin

d:= Sqr(b) - 4*a*c; if d < 0 then

ok := False // уравнение не имеет решения

else

begin

ok := True;

x1 := (-b + Sqrt(d)) / (2*a) ; x2 := (b + Sqrt(d)) / (2*a);

end;

end;
```

Использование процедуры

Разработанную процедуру нужно поместить в раздел implementation, перед подпрограммой, которая использует эту процедуру.

Инструкция вызова процедуры в общем виде выглядит так:

Имя(СписокПараметров);

где:

П имя — имя вызываемой процедуры;

- списокПараметров — разделенные запятыми фактические параметры.

Фактическим параметром, в зависимости от описания формального параметра в объявлении процедуры, может быть переменная, выражение или константа соответствующего типа.

Например, инструкция вызова приведенной выше процедуры решения квадратного уравнения может выглядеть следующим образом:

```
SqRoot(StrToFloat(Edit1.Text),

StrToFloat(Edit2.Text),

StrToFloat(Edit3.Text), k1,k2,rez);
```

Если в описании процедуры перед именем параметра стоит слово var, то при вызове процедуры на месте соответствующего параметра должна стоять переменная основной программы. Использование константы или выражения считается ошибкой, и компилятор в этом случае выведет сообщение: Types of actual and formal var parameters must be identical (ТИП фактического параметра должен соответствовать типу формального параметра).

В листинге 6.6 приведена программа решения квадратного уравнения, в которой используется процедура SqRoot. Окно программы представлено на рис. 6.2.

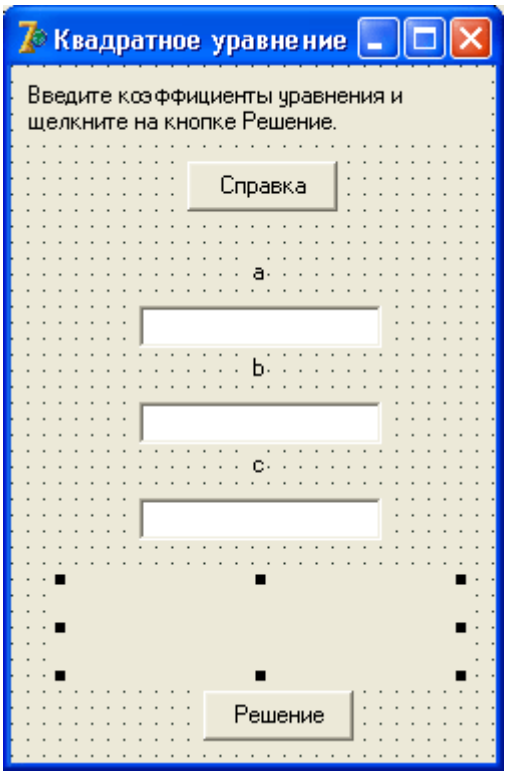

Рис. 6.2. Окно программы **Квадратное уравнение**

Листинг 6.6. Решение квадратного уравнения (использование процедуры)

```
unit SqRoot_; interface

uses

Windows, Messages, SysUtils, Variants, Classes,

Graphics, Controls, Forms, Dialogs, StdCtrls;

type

TForml = class(TForm)

Editl: TEdit;

Edit2: TEdit;

Edit3: TEdit;

Label1: TLabe1;

Label2: TLabe1;

Label3: TLabe1;

Label4: TLabe1;

Button1: TButton;

Label5: TLabel;

procedure ButtonlClick(Sender: TObject); private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R *.dfm}

// решает квадратное уравнение
```

```
procedure SqRoot(a,b,c : real; var xl, x2 : real; var ok : boolean);
{ a,b,c — коэффициенты уравнения x1,x2 — корни уравнения
ok = True — решение есть ok = False — решения нет }
var
d : real; // дискриминант begin
d:= Sqr(b) - 4*a*c; if d < 0 then
ok := False // уравнение не имеет решения
else
begin
ok := True;
xl := (-b + Sqrt(d)) / (2*a); x2 := (b + Sqrt(d)) / (2*a) ;
end;
end;
procedure TForml.ButtonlClick(Sender: TObject);
var
k1,k2: real; // корни уравнения
rez: boolean; // True —решение есть, False —решения нет mes:
string; // сообщение begin
SqRoot(StrToFloat(Editl.Text), StrToFloat(Edit2.Text) ,
StrToFloat(Edit3.Text) , k1,k2,rez);
if rez then
mes := 'Корни уравнения' + #13 +
'x1='+FloatToStrF(kl,ffGeneral,
4,2)+#13+ 'x2='+FloatToStrF(k2,ffGeneral,4,2)+#13 else
mes := 'Уравнение не имеет решения'; labels.Caption := mes;
```

end;

end.

Повторное использование функций и процедур

Разработав некоторую функцию, программист может использовать ее в другой программе, поместив текст этой функции в раздел implementation. Однако этот способ неудобен, т. к. приходится набирать текст функции заново или копировать его из текста другой программы.

Создание модуля

Delphi позволяет программисту поместить свои функции и процедуры в отдельный модуль, а затем использовать процедуры и функции модуля в своих программах, указав имя модуля в списке модулей, необходимых программе (инструкция uses).

Чтобы приступить к созданию модуля, нужно сначала закрыть окно формы и окно модуля формы (в ответ на вопрос о необходимости сохранения модуля следует выбрать No, т. е. модуль, соответствующий закрытой форме, сохранять не надо). Затем из меню **File** нужно выбрать команду New | **Unit.** В результате открывается окно редактора кода, в котором находится сформированный Delphi шаблон модуля. Его текст приведен в листинге 6.7.

Листинг 6.7. Шаблон модуля

unit Unit1;

interface implementation

end.

Начинается модуль заголовком — инструкцией unit, в которой указано имя модуля. Во время сохранения модуля это имя будет автоматически заменено на имя, указанное программистом.

Слово interface отмечает раздел интерфейса модуля. В этот раздел программист должен поместить объявления находящихся в модуле процедур и функций, которые могут быть вызваны из других модулей, использующих данный.

В раздел implementation (реализация) нужно поместить процедуры и функции, объявленные в разделе interface.

В качестве примера в листинге 6.8 приведен модуль программиста, который содержит рассмотренные ранее функции IsInt и isFioat.

Листинг 6.8. Модуль программиста

```
unit my__unit;
interface // объявления процедур и функций,
// доступных программам,
// использующим этот модуль
function IsInt(ch : char) : Boolean;
// функция Islnt проверяет, является ли символ
// допустимым во время ввода целого числа
function IsFloat(ch : char; st: string) : Boolean;
// Функция IsFloat проверяет, является ли символ допустимым
// во время ввода дробного числа
// ch — очередной символ
// st — уже введенные символы
implementation // реализация
// проверяет, является ли символ допустимым
// во время ввода целого числа
function Islnt(ch : char) : Boolean;
begin
if (ch >= '0') and (ch <= '9') // цифры
or (ch = #13) // клавиша <Enter>
or (ch = #8) // клавиша <Backspace>
then Islnt := True // символ допустим
else Islnt := False; // недопустимый символ
end;
// проверяет, является ли символ допустимым
// во время ввода дробного числа
```

```
function IsFloat(ch : char; st: string) : Boolean;

// ch — очередной символ // st — уже введенные символы

begin

if (ch >= '0') and (ch <= '9') // цифры

or (ch = #13) // клавиша <Enter>

or (ch = #8) // клавиша <Backspace>

then

begin

IsFloat := True; // символ верный

Exit; // выход из функции

end; case ch of

'-': if Length(st) = 0 then IsFloat := True; ',':

if (Pos(',',st) = 0)

and (st[Length(st)] >= '0') and (st[Length(st)] <= '9')

then // разделитель можно ввести только после цифры

// и если он еще не введен

IsFloat := True; else // остальные символы запрещены

IsFloat := False; end

// это раздел инициализации // он в данном случае не содержит инструкция end.
```

Сохраняется модуль обычным образом, т. е. выбором из меню **File** команды **Save.** Вместе с тем, для модулей повторно используемых процедур и функций лучше создать отдельную папку, назвав ее, например, **Units.**

Использование модуля

Для того чтобы в программе могли применяться функции и процедуры модуля, программист должен добавить этот модуль к проекту и указать имя модуля в списке используемых модулей (обычно имя модуля

программиста помещают в конец сформированного Delphi списка используемых модулей).

В листинге 6.9 приведен вариант программы **Поездка на дачу.** Процедура обработки события onKeyPress в полях ввода исходных данных обращается к функции IsFloat, которая находится в модуле my_unit.pas, поэтому в списке используемых модулей указано имя модуля my_unit.

Листинг 6.9. Использование функции из модуля программиста

```
unit fazenda_;

interface

uses

Windows, Messages, SysUtils, Variants,

Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, my_unit; // модуль программиста

type

TForm1 = class(TForm)

Edit1: TEdit; // расстояние

Edit2: TEdit; // цена литра бензина

Edit3: TEdit; // потребление бензина на 100 км

CheckBoxl: TCheckBox; // True — поездка туда и обратно

Button1: TButton; // кнопка Вычислить

Label4: TLabel; // поле вывода результата расчета

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

procedure EditlKeyPress(Sender: TObject;

var Key: Char);

procedure Edit2KeyPress(Sender: TObject;
```

```
var Key: Char);
procedure Edit3KeyPress(Sender: TObject;
var Key: Char);
procedure Button1Click(Sender: TObject);
private
{ Private declarations} public
{ Public declarations } end;
var
Form1: TForm1;
implementation
{$R *.dfm}
// нажатие клавиши в поле Расстояние
procedure TForml.EditlKeyPress(Sender: TObject;
var Key: Char);
begin
if Key = Char(VK_RETURN)
then Edit2.SetFocus // переместить курсор в поле Цена
else If not IsFloat(Key,Edit2.Text)
then Key := Chr(O);
end;
// нажатие клавиши в поле Цена
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
if Key = Char(VK_RETURN)
then Edit3.SetFocus // переместить курсор в поле Потребление .
```

```
else If not IsFloat(Key,Edit2.Text) then Key := Chr(0);
end;
// нажатие клавиши в поле Потребление
procedure TForm1.EditSKeyPress(Sender: TObject;
var Key: Char);
begin
if Key = Char(VK_RETURN)
then Button1.SetFocus // // сделать активной кнопку Вычислить
else If not IsFloat(Key,Edit2.Text) then Key := Chr(0);
end;
// щелчок на кнопке Вычислить
procedure TForml.ButtonlClick(Sender: TObject);
var
rast : real; // расстояние
cena : real; // цена
potr : real; // потребление на 100 км
summ : real; // сумма
mes: string;
begin
rast := StrToFloat(Editl.Text) ;
cena := StrToFloat(Edit2.Text);
potr := StrToFloat(Edit3.Text);
summ := rast / 100 * potr * cena;
if CheckBoxl.Checked then summ := summ * 2;
mes := 'Поездка на дачу';
```

```
if CheckBox1.Checked then

mes : = mes + ' и обратно' ;

mes := mes + 'обойдется в '

+ FloatToStrF(summ,ffGeneral, 4,2) + ' руб.';

Label4.Caption := mes;

end;

end.
```

После добавления имени модуля в список модулей, используемых приложением, сам модуль нужно добавить в проект. Для этого из меню **Project** надо выбрать команду **Add to Project** и в открывшемся диалоговом окне — имя файла модуля. В результате добавления модуля к проекту в окне редактора появится вкладка с текстом добавленного к проекту модуля.

Увидеть структуру проекта можно в окне **Project Manager,** которое появляется в результате выбора соответствующей команды из меню **View.** В качестве примера на рис. 6.3 приведена структура проекта **Поездка на дачу.**

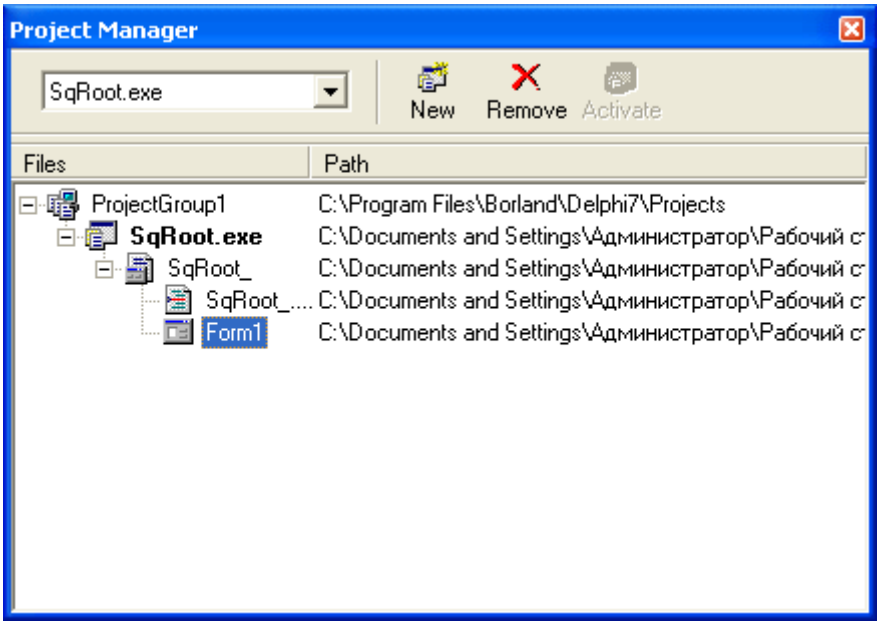

Рис. 6.3. Структура проекта отражается в окне **Project Manager**

После добавления модуля к проекту и включения его имени в список используемых модулей (инструкция uses) можно выполнить компиляцию программы.