Уважаемые студенты групп!

Вашему вниманию представлена лекция на тему «Программирование в системе Delphi с использованием подпрограмм».

Задание

1. Прочитать внимательно лекцию.
2. Законспектировать лекцию в рабочую тетрадь не менее 3-6 страницы рукописного текста. В конспекте лекции обязательно должно быть приведены примеры.
3. Ответить письменно в рабочей тетради на контрольные вопросы.
4. Дата предоставления фотоотчета лекции до 27.01

С уважением Ганзенко Ирина Владимировна
!!! Если возникнут вопросы обращаться по телефону 0721134803 (вацап), +79591134803 (телеграмм)
disobuch.ganzenko2020@mail.ru

Лекция. Программирование в системе Delphi с использованием подпрограмм

Целью данного раздела является изучение структур процедур и функций, механизма взаимодействия программы с подпрограммами, рекурсивного метода организации вычислений, получение навыков программирования алгоритмов средней сложности с использованием процедур и функций.

План

Две формы подпрограмм

Если в программе встречаются повторяющиеся фрагменты, то целесообразно эти фрагменты оформить как подпрограмму. Подпрограммы представляют собой относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именем. Упоминание этого имени в тексте программы называется вызовом подпрограммы. Подпрограмма обычно

содержит описания идентификаторов: констант, типов, переменных и вложенных в нее подпрограмм. Все идентификаторы, описанные внутри подпрограммы, локализуются в ней. В разделе описания подпрограмм могут встретиться описания подпрограмм низшего уровня, в них – описания других подпрограмм и т. д. Все имена в пределах подпрограммы, в которой они объявлены, должны быть уникальными. При входе в подпрограмму низшего уровня становятся доступными не только объявленные в ней имена, но и сохраняется доступ ко всем именам верхнего уровня. При взаимодействии подпрограмм одного уровня иерархии вступает в силу основное правило *Object Pascal*: любая подпрограмма перед её использованием должна быть описана.

Существуют две формы подпрограмм – процедуры и функции. Основное функциональное отличие процедур от функций заключается в том, что процедура позволяет получить один или более результатов, а функция – всегда только один результат.

Использование подпрограмм позволяет уменьшить объем памяти, занимаемый кодом программы в целом. Однако время выполнения программы при этом несколько возрастает, так как появляются временные издержки на вызов подпрограмм и возврат из подпрограмм.

Процедуры

Каждую свою процедуру программисту необходимо описать в разделе описаний, то есть указать её заголовок и тело.

В *заголовке* объявляются *имя процедуры* и *формальные параметры*, если они есть.

Формат: Procedure <имя>(формальные параметры);
Procedure <имя>;

Пример: Procedure A (K: integer; B: real);
Procedure B;

Тело процедуры представляет собой локальный блок, по структуре аналогичный программе:

```
Procedure <имя> (формальные параметры);
//Раздел объявления меток
Label met1, met2;
//Раздел объявления констант
Const A=5; B=2.8;
//Раздел описания типов
Type tx=array [0..4] of real;
//Раздел описания переменных
Var y:integer; x:tx;
//Раздел описания встроенных процедур и функций
//Раздел операторов
begin
<Операторы языка программирования>
```

end;

Разделы описаний и объявлений могут использоваться в произвольном порядке и встречаться несколько раз. При этом нужно соблюдать правило: все описания и объявления элементов программы должны быть сделаны до того, как они будут использованы.

Вызов процедуры осуществляется простым упоминанием имени процедуры в операторе вызова процедуры. Оператор вызова процедуры состоит из имени (идентификатора) процедуры и списка фактических параметров, отделённых друг от друга запятыми и заключённых в круглые скобки. Список фактических параметров может отсутствовать, если процедуре не передаётся никаких значений.

Формат: <имя процедуры> (параметр 1, ……, параметр N);
<имя процедуры>;

Пример:

A(k,c);
B; {фактических параметров нет}

Параметры обеспечивают механизмы, которые позволяют выполнять процедуру с начальными данными.

Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливаются взаимнооднозначные соответствия в результате их перебора слева направо, то есть формальных и фактических параметров должно быть одинаковое количество; порядок следования формальных и фактических параметров должен быть один и тот же; тип каждого фактического параметра должен совпадать с типом соответствующего ему формального параметра.

Функции

Описание функции в основном аналогично описанию процедуры. Для функции, кроме того, указывается тип возвращаемого ею результата. Функция состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово *Function*, идентификатор (имя) функции, необязательный список формальных параметров, заключённый в круглые скобки, и тип возвращаемого функцией значения:

Function Pr (x, y: integer):real;
Function Z: boolean;

Отличие вызова функции от вызова процедуры заключается в том, что обращение к функции можно использовать в соответствующих выражениях наряду с переменными и константами.

Тело функции представляет собой локальный блок, по структуре аналогичный программе:

Function<имя>(формальные параметры):<тип результата>;

<разделы описаний>;
begin
<раздел операторов>
end;

Для каждой функции создается по умолчанию переменная *Result*, которая является синонимом имени функции. В разделе операторов должен использоваться хотя бы один раз оператор присваивания, в левой части которого записывается имя функции или переменная *Result*.

Параметры

Список *формальных* параметров необязателен и может отсутствовать. Если же он есть, то в нём должны быть перечислены имена *формальных* параметров и их типы, например:

Procedure SB(a: real; b: integer; var c: char);

Любой из *формальных* параметров подпрограммы может быть либо *параметром-значением*, либо *параметром-переменной*, либо *параметром-константой*.

Если параметры определяются как *параметры-переменные*, перед ними необходимо ставить зарезервированное слово ***var***, а если это *параметры-константы* – слово ***const***, например:

Procedure SA(var a:real; b:real; const c:string);
Здесь: a – *параметр-переменная*;
b – *параметр-значение*;
c – *параметр-константа*.

Определение формального параметра тем или иным способом существенно в основном только для вызывающей программы: если формальный параметр объявлен как *параметр-переменная*, то при вызове подпрограммы ему должен соответствовать фактический параметр в виде переменной нужного типа.

Если параметр определен как *параметр-значение*, то перед вызовом подпрограммы это значение вычисляется, полученный результат копируется во временную память и передается подпрограмме. Важно учесть, что даже если в качестве фактического *параметра-значения* указано простейшее выражение в виде переменной или константы, все равно подпрограмме будет передана лишь копия значения переменной (константы). Любые возможные изменения в подпрограмме *параметра-значения* никак не воспринимаются вызывающей программой, так как в этом случае изменяется копия фактического параметра.

Если параметр определен как *параметр-переменная*, то при вызове подпрограммы передается сама переменная, а не ее копия (фактически в этом случае подпрограмме передается адрес переменной). Изменение *параметра-переменной* приводит к изменению самого фактического параметра в вызывающей программе.

В случае *параметра-константы* в подпрограмму также передается адрес области памяти, в которой располагается переменная или вычисленное значение.

Однако компилятор блокирует любые присваивания *параметру-константе* нового значения в теле подпрограммы.

Итак, *параметры-переменные* используются как средство связи алгоритма, реализованного в подпрограмме, с внешним миром: с помощью этих параметров подпрограмма может передавать результаты своей работы вызывающей программе. Разумеется, в распоряжении программиста всегда есть и другой способ передачи результатов – через *глобальные* переменные. Однако злоупотребление *глобальными* связями делает программу, как правило, запутанной, трудной в понимании и сложной в отладке. В соответствии с требованиями хорошего стиля программирования рекомендуется там, где это возможно, использовать передачу результатов через *параметры- переменные*.

Существует еще одна разновидность параметров в *Object Pascal* – *нетипизированные* параметры. Параметр считается *нетипизированным*, если тип *формального параметра-переменной* в заголовке подпрограммы не указан, при этом соответствующий ему *фактический* параметр может быть переменной любого типа. Заметим, что *нетипизированными* могут быть только *параметры-переменные*:

```
Procedure MyProc( var aParametr );
```

Нетипизированные параметры обычно используются в случае, когда тип данных несущественен.

Типом любого параметра в списке *формальных* параметров может быть только стандартный или ранее объявленный тип. Если мы хотим передать какой-то элемент массива, то проблем не возникает, но если в подпрограмму передается весь массив, то следует первоначально описать его тип.

Пример

```
Type
aType = array [1..10] of real;
/Заголовок процедуры:
Procedure S(var a: aType);
```

Поскольку короткая строка является фактически символьным массивом, её передача в подпрограмму осуществляется аналогичным образом:

```
Type
InType = String [15];
OnType = String[30];
//Заголовок функции
Function St (S: InType): OnType;
```

Object Pascal поддерживает открытые массивы, легко решающие проблему передачи подпрограмме одномерных массивов переменной длины. Открытый массив представляет собой *формальный* параметр подпрограммы, описывающий базовый тип элементов массива, но не определяющий его размерности и границы:

```
Procedure My (OpenArray: array of Integer);
```

Рекурсия

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих ее операторов обращается сама к себе. Слово «рекурсия» означает «возвращение».

При выполнении правильно организованной рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока не будет получено тривиальное решение поставленной задачи.

Главное требование к рекурсивным подпрограммам заключается в том, что вызов рекурсивной подпрограммы должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным.

Структура рекурсивной процедуры может принимать три разных формы:

1. Форма с выполнением действий до рекурсивного вызова (с выполнением действий на рекурсивном спуске):

```
Procedure Rec;
begin
S;
if условие then Rec;
end;
```

2. Форма с выполнением действий после рекурсивного вызова (с выполнением действий на рекурсивном возврате).

```
Procedure Rec;
begin
if условие then  Rec;
S;
end;
```

3. Форма с выполнением действий как до, так и после рекурсивного вызова (с выполнением действий как на рекурсивном спуске, так и на рекурсивном возврате):

а)
```
Procedure Rec;
   begin
S1;
if условие then  Rec;
S2;
   end;
```

б)
```
Procedure Rec;
   begin
if условие then
   begin
     S1;
     Rec;
     S2;
   end;
```

```
end;
```

Структура рекурсивной функции аналогична структуре рекурсивной процедуры.

Косвенная рекурсия

Рекурсивный вызов может быть косвенным. В этом случае программа обращается к себе непосредственно, а путем вызова другой подпрограммы, в которой содержится обращение к первой. Если строго следовать правилу, согласно которому каждый идентификатор перед употреблением должен быть описан, вводится опережающее описание:

```
Procedure B (var k : Byte); Forward;
Procedure A (var c : Byte);
begin
      B(k);
end;
Procedure B;
begin
     A(c);
end;
```

Все формы рекурсивных процедур и функций находят применение на практике. Глубокое понимание рекурсивного механизма и умение управлять им является необходимым качеством квалифицированного программиста.

Пример программирования с подпрограммами

Задание. Вычислить функцию:

$S = \frac{\cos x}{1!} + \ldots + \frac{\cos Nx}{N!}$, где N – количество членов ряда.

Структура программы будет включать в себя два модуля: программу с именем Progect1 и модуль с именем Unit1, связанный с формой Form1.

Чтобы продемонстрировать на учебном примере особенности и отличия в использовании процедуры и функции, реализуем подпрограмму вычисления факториала тремя способами: в виде процедуры, в виде функции и рекурсивным методом.

1. Разработка алгоритма (рис. 7.1):

• Входные данные:

а) X – вещественная переменная, являющаяся аргументом функции cos(NX);

б) N – целочисленная переменная, обозначающая количество членов ряда.

• Выходные данные: S – вещественная переменная, значение которой есть сумма членов ряда.

• Промежуточные данные:

а) K – целочисленная переменная, используемая как счетчик цикла;

б) R – вещественная переменная.

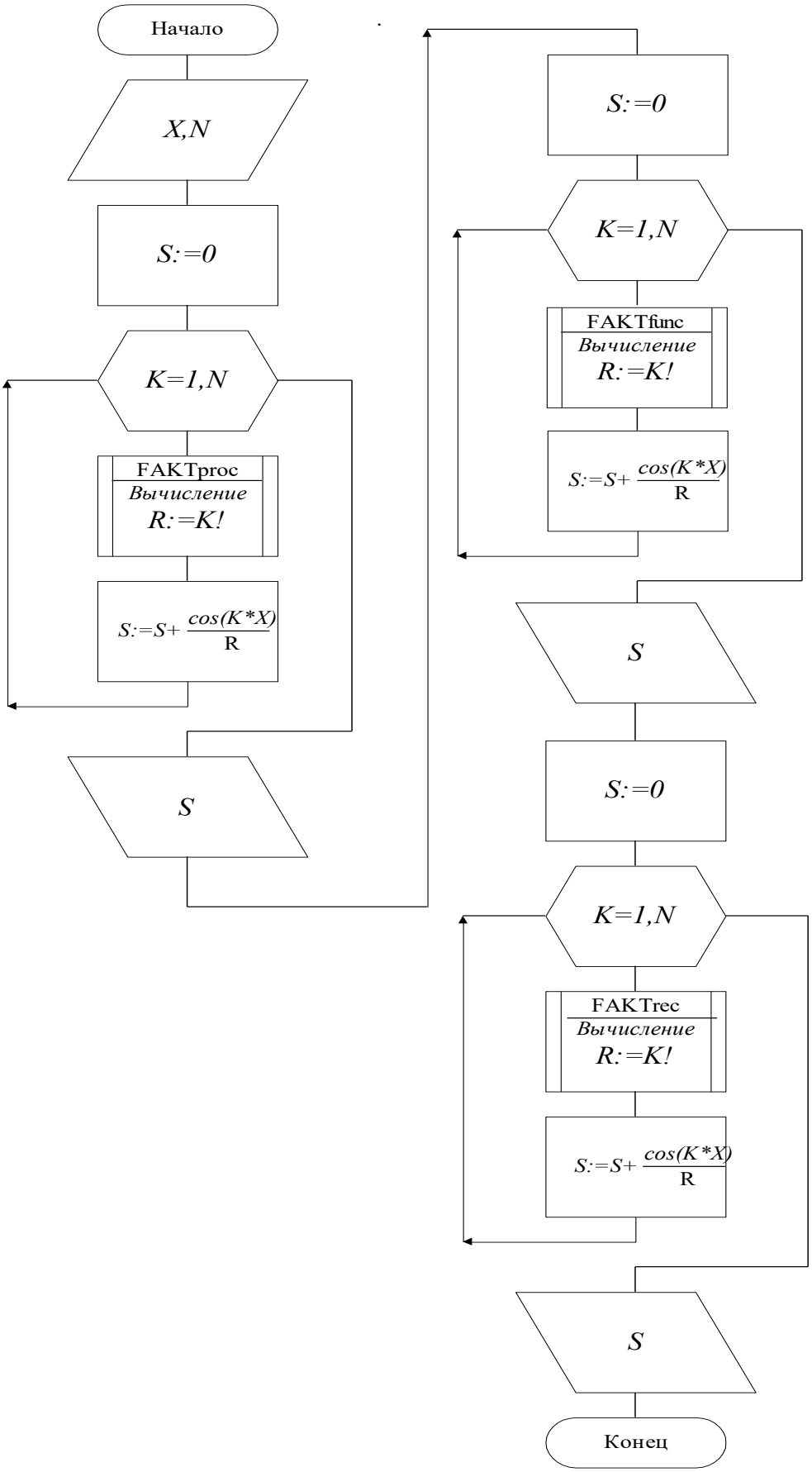

Рис. 7.1. Схема алгоритма основной программы

Процедура FAKTproc (рис. 7.2). Данная процедура должна вычислить R=K! .

• Входные данные: K – целочисленная переменная.

• Выходные данные: R – вещественная переменная, являющаяся значением K! .

• Промежуточные данные: I – целочисленная переменная, используемая как счетчик цикла.

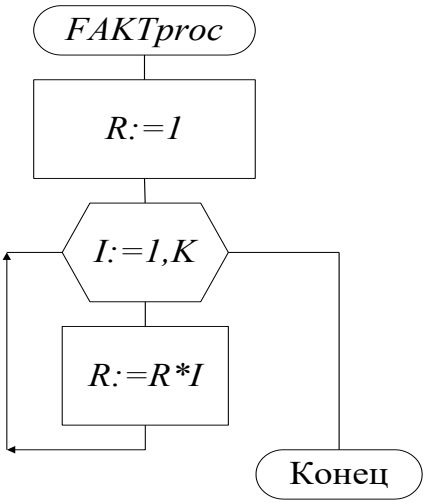

Рис. 7.2. Схема алгоритма процедуры FAKTproc

Функция FAKTfunk (рис. 7.3). Функция FAKTfunk вычисляет K! .

• Входные данные: K – целочисленная переменная.

• Выходные данные: FAKTfunk – имя функции вещественного типа.

• Промежуточные данные: I – целочисленная переменная, используемая как счетчик цикла.

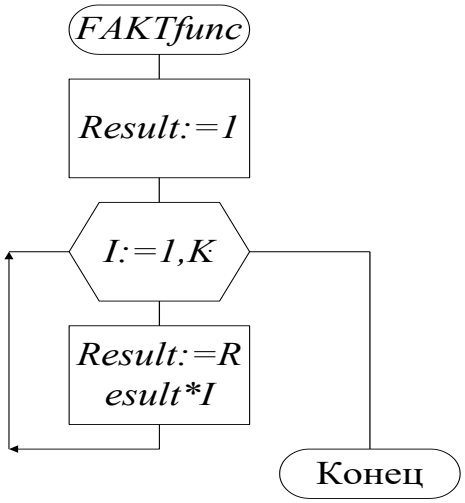

Рис. 7.3. Схема алгоритма функции FAKTfunc

Функция FAKTrec (рис. 7.4). Функция FAKTrec вычисляет K! рекурсивным методом.

• Входные данные: K – целочисленная переменная.

• Выходные данные: FAKTrec – имя функции вещественного типа.

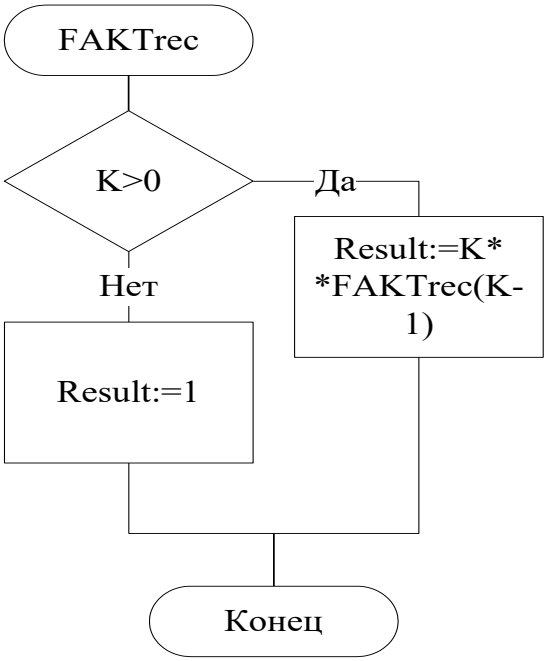

Рис. 7.4. Схема алгоритма функции FAKTrec

2. Разработка формы (табл. 7.1, рис. 7.5).

Таблица 7.1

Используемые компоненты

Имя компонента	Страница палитры компонент	Настраиваемое свойство	Значение
1. Form1	–	Caption	
2. StaticText1	Additional	Caption	Введите N
3. StaticText2	Additional	Caption	Результат S
4. Edit1	Standard	Text	
5. Edit2	Standard	Text	
6. Edit3	Standard	Text	
7. Edit4	Standard	Text	
8. Edit5	Standard	Text	
9. Button1	Standard	Caption	Вычисление функции

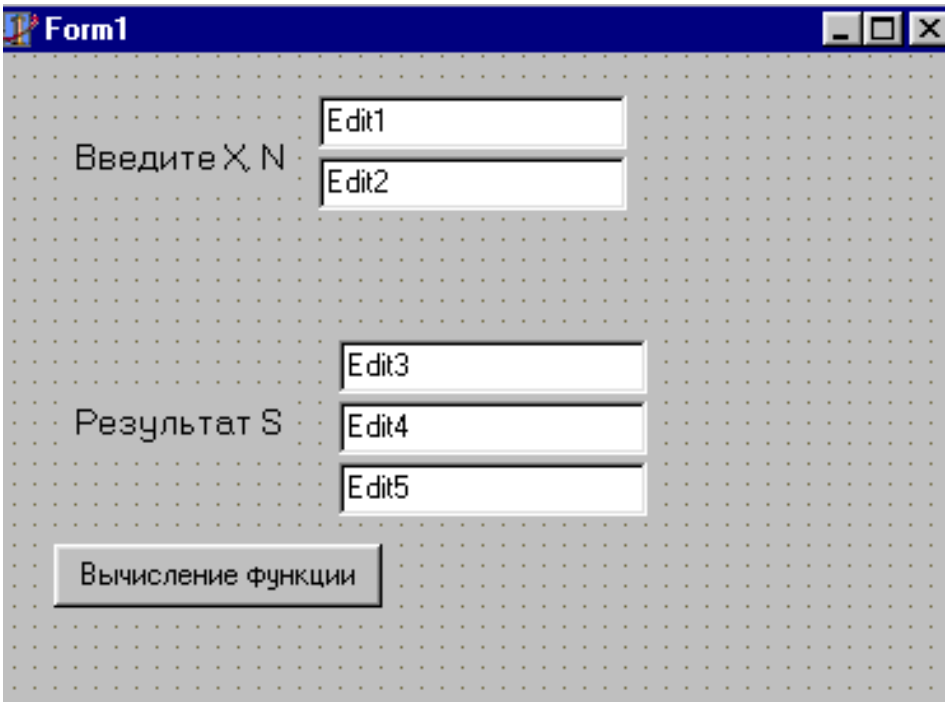

Рис. 7.5. Изображение формы

3. Пример программы:

```
program Project1;
uses
        Forms,
        Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
        Application.Initialize;
        Application.CreateForm(TForm1, Form1);
        Application.Run;
end.
unit Unit1;
interface
uses
        Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;
type
        TForm1 = class(TForm)
        Edit1: TEdit;           // поле для ввода X
        Edit2: TEdit;           // поле для ввода N
        Edit3: TEdit;           // поле для вывода S (процедура)
        Edit4: TEdit;           // поле для вывода S (функция)
        Edit5: TEdit;           // поле для вывода S (рекурсивная
                                //   функция)
```

```
        StaticText1: TStaticText;    //
        StaticText2: TStaticText;    //  комментарии к полям
        Button1: TButton;            // кнопка выполнить
        procedure Button1Click(Sender: TObject); // процедура,
        //вызываемая при щелчке левой кнопкой мыши на Button1;
        end;
var
        Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var    X,S,R:extended;
        N,K:integer;
procedure FAKTproc(K:integer;var R:extended); //  вычисление  с
// помощью процедуры
var i:integer;
begin
        R:=1;
        for i:=1 to K do
        R:=R*i;
end;
function FAKTfunc(K:integer):extended; // вычисление с помощью
        // функции
var i:integer;
begin
        Result:=1;
        for i:=1 to K do
        Result:=Result*i;
end;
function FAKTrec(K:integer):extended; // вычисление с помощью
        // рекурсии
begin
        if k>0 then
        Result:=k*FAKTrec(k–1)
        else Result:=1;
end;
begin
        X:=StrToFloat(Edit1.Text);
        N:=StrToInt(Edit2.Text);
        S:=0;
        for K:=1 to N do
        begin
        FAKTproc(K,R);
```

```
        S:=S+cos(K*X)/R;
        end;
Edit3.Text:=FloatToStr(S);
S:=0;
        for K:=1 to N do
        begin
        R:=FAKTfunc(K);
        S:=S+cos(K*X)/R;
        end;
Edit4.Text:=FloatToStr(S);
S:=0;
for K:=1 to N do
        begin
        R:=FAKTrec(K);
        S:=S+cos(K*X)/R;
        end;
Edit5.Text:=FloatToStr(S);
end;
end.
```

Контрольные вопросы к главе 7

1. Для чего предназначаются подпрограммы?
2. Что включает в себя заголовок процедуры?
3. Что включает в себя заголовок функции?
4. Какую структуру имеет процедура?
5. Какую структуру имеет функция?
6. Чем отличается процедура от функции?
7. Какая существует взаимосвязь между формальными и фактическими параметрами?
8. Какие существуют разновидности параметров подпрограмм?
9. Чем отличаются параметры-переменные от параметров-констант?
10. Чем отличаются параметры-значения от параметров-констант?
11. С какой целью используются открытые массивы?
12. В чем суть рекурсивного метода организации вычислений?
13. Какие достоинства и недостатки рекурсивного метода?
14. Каким образом реализуется косвенная рекурсия?