Презентация по курсу: «Прикладное программирование»

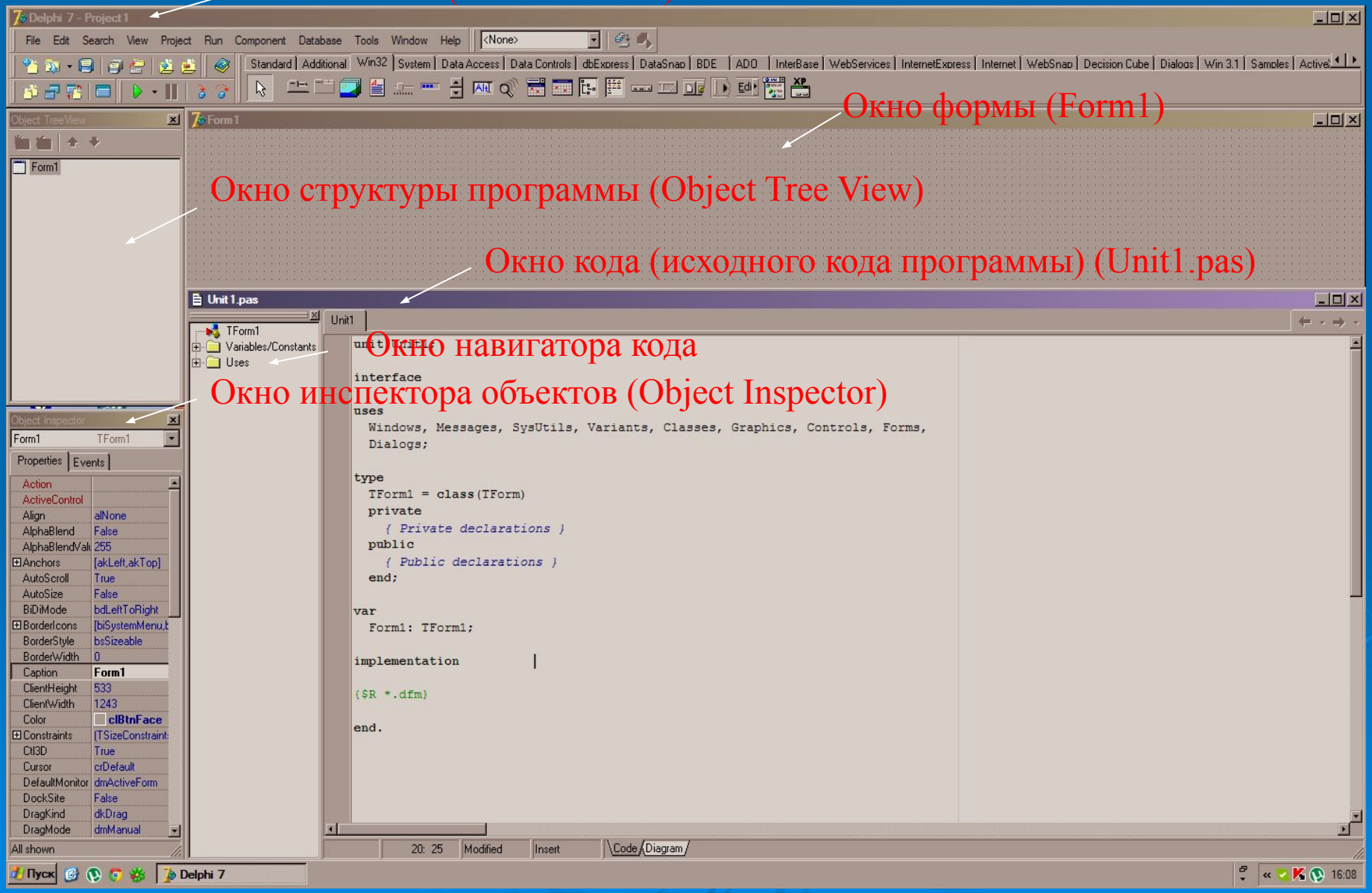
Delphi - императивный, структурированный, объектно-ориентированный язык программирования высокого уровня, диалект Object Pascal.

Delphi является потомком среды программирования Turbo Pascal. Название среды произошло от названия города в Древней Греции, где находился знаменитый Дельфийский оракул (храм Аполлона в городе Дельфы, жрецы которого занимались предсказаниями).

Система визуального объектно-ориентированного программирования Delphi позволяет:

- 1. Создавать законченные приложения для Windows самой различной направленности.
- 2. Быстро создавать оконный интерфейс для любых приложений; интерфейс удовлетворяет всем требованиям Windows и автоматически настраивается на ту систему, которая установлена, поскольку использует функции, процедуры и библиотеки Windows.
- 3. Создавать динамически присоединяемые библиотеки компонентов, форм, функций.
- 4. Создавать мощные системы работы с базами данных любых типов.
- 5. Формировать и печатать сложные отчеты, включающие таблицы, графики и т.п.
- 6. Создавать профессиональные программы установки для приложений Windows, учитывающие специфику и требования операционной системы.

Интерфейс Delphi

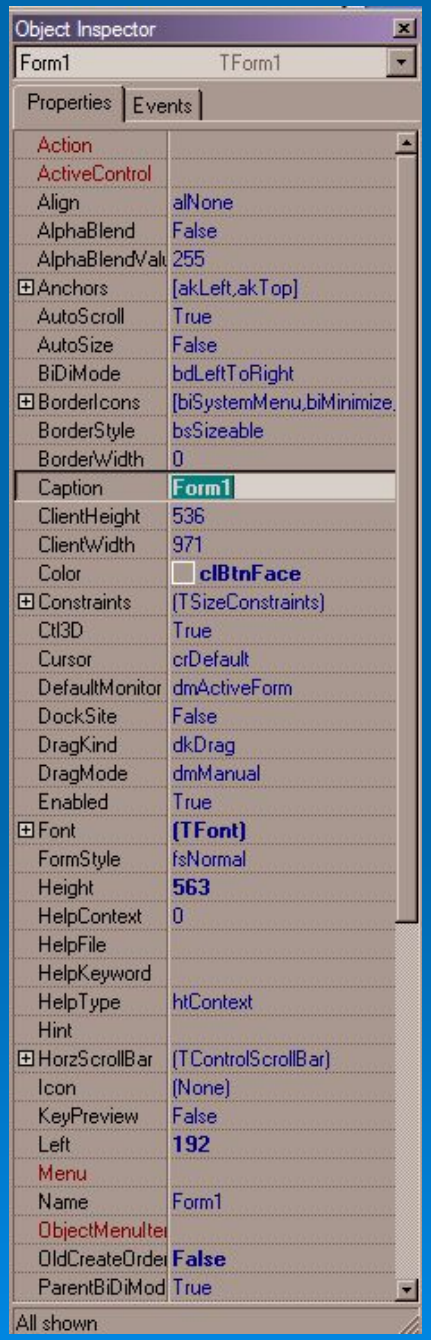


Интерфейс Delphi

- Интерфейс Delphi представляет собой совокупность нескольких одновременно открытых окон:
- 1. Главное окно – управляет проектом создания программы (всегда находится вверху рабочего стола).
- 1.1. Главное меню Delphi (Main Menu) – содержит командную строку.
- 1.2. Пиктографические кнопки, дублирующие основные опции меню.
- 1.3. Палитра компонентов, состоящая из вкладок (Standard, Additional, Win32, System, Data Access, Data Controls, dbExpress, DataSnap, BDE, ADO, InterBase, WebServices, InternetExpress, Internet …) с размещенными на них компонентами в виде пиктограмм (графических изображений).

Компонент – это функциональный элемент с определенными свойствами, размещаемый в окне формы (Form1) (например, кнопка, текстовое поле, текстовая метка, таблица и т.п.).

Интерфейс Delphi



- 2. Окно формы (Form1) – это проект Windows-окна будущей программы, в окне формы компонуются (размещаются) требуемые программисту компоненты, тем самым формируется интерфейс программы.
- 3. Окно инспектора объектов (Object Inspector) – это окно служит для изменения простых и сложных свойств компонентов, на вкладках Properties (Свойства) и Events(События).
- 3.1. Страница Properties (Свойства) инспектора объектов состоит из двух колонок. Слева в колонке приводится название свойства компонента, справа – его значение. Совокупность свойств отображают видимую сторону компонента: заголовок, положение, размер, цвет, шрифт и т.д. Свойства могут быть простыми и сложными. Простые свойства определяются конкретным значением – числом, символом, логическим значением (True/False) (например, свойство Caption является свойством заголовка компонента. Список всех возможных значений простого свойства раскрывается щелчком на стрелке в правой колонке таблицы «Свойства».

Интерфейс Delphi

- Сложные свойства определяются совокупностью значений. Слева от них стоит знак «+», а в правой колонке – «…», щелчок на котором открывает диалоговое окно по установке значений сложного свойства (например, свойство Font служит для изменения параметров шрифта компонента).
- 3.2. Страница Events(События) инспектора объектов – отображает совокупность событий, которые отражают поведенческую сторону компонента: реакция на щелчок мыши, нажатие на кнопку, вид при появлении на экране или закрытие окна. Эта страница также состоит из двух колонок. В левой колонке стоит название события, справа – имя подпрограммы, обрабатывающей это событие.
- 3.3. Список всех компонентов на форме.

Интерфейс Delphi

- 4. Окно кода программы, по умолчанию носящего имя Unit1.pas – это окно служит для создания и редактирования текста программы на алгоритмическом языке высокого уровня Object Pascal. В начале своей работы окно кода содержит минимальный исходный текст, который обеспечивает работу пустого окна формы.
- 5. Окно навигатора кода – это вспомогательный броузер (browse - просматривать), с помощью которого можно быстро найти элемент программы при большом объеме ее кода. Вызов броузера осуществляется при выборе в меню опции View – Code Explorer.
- 6. Окно структуры программы (ObjectTreeView) – это окно позволяет просматривать структуру и наименование компонентов, размещенных в окне формы (Form1).

- Программирование на Delphi состоит из конструирования Windows-окна с помощью палитры компонентов в окне формы (Form1) и написания кода программы в окне исходного кода (Unit1.pas) с помощью клавиатуры ПК. Переключение между окном формы и окном кода осуществляется щелчком левой клавиши мыши или клавишей F12.
- Совокупность окон и обслуживающих их программ в Delphi называется проектом (Project). Проект содержит следующие файлы:
- 1. *.pas – файл кода программы (модуль). * - имя файла
- 2. *.dfm – файл формы (Form1).
- 3. *.dcu – файл машинных кодов для модуля и формы.
- 4. *.exe –единый исполняемый файл.
- 5. *.dpr – файл проекта.
- Поскольку программы содержат несколько файлов, для записи каждой программы требуется создавать свой каталог в отдельной директории.

- Порядок программирования:
- 1. Конструируют форму (Form1), выбирая нужные компоненты в закладках панели компонентов и размещая их в пустом окне формы.
- 2. Изменяют простые и сложные свойства компонентов на страницы Properties(Свойства) инспектора объектов.
- 3. Назначают процедуры обработки (последовательность действий) поведенческой стороны компонента на странице Events (События) инспектора объектов.

- 4. Пишут необходимые фрагменты кода программы в окне кода.
- 5. Сохраняют файлы проекта (Project1) командой: File – Save All (сохранить по умолчанию два файла Project1.dpr и Unit1.pas в отдельную директорию work D: \Новая папка\ Л.р. №).
- 6. Производят компиляцию и отладку написанного текста программы: Project – Compile Project1 (или Compile All Projects).
- Компиляция – это процесс преобразования исходного текста программы (на алгоритмическом языке высокого уровня Object Pascal) в исполняемую (на языке машинных кодов). Процесс компиляции состоит из двух этапов. На первом этапе выполняется проверка текста на отсутствие ошибок. На втором – генерируется исполняемый файл проекта Project1.exe. Результаты компиляции отражаются в диалоговом окне.

- Схема процесса компиляции:

- 7. Производят запуск готовой программы командой: Run – Run, нажатием на клавишу или на клавишу F9.

- Структура файлов проекта и модуля:
- 1. Файл проекта. Этот файл представляет собой программу, написанную на языке ObjectPascal. Для просмотра файлов проекта следует выбрать в меню Project – ViewSource.
- Файл проекта имеет следующий вид:
- Справа от строк программы обозначено название команд.
- program Project1; *заголовок программы*
- uses *использовать*
- Forms, *модули форм и обрабатывающих их тексты*
- Unit1 in 'Unit1.pas' {Form1};
- {$R *.res} *директива компилятору*
- begin *начало тела программы*
- Application.Initialize; *объекты и их методы*
- Application.CreateForm(TForm1, Form1);
- Application.Run;
- end. *Терминатор закрытия программы (конец программы)*

- 2. Файл модуля. Модуль это программная единица, предназначенная для размещения фрагментов кода. Текст модуля редактировать, причем фрагменты кода программы вносятся между строкой директивы компилятору{$R *.dfm} и заголовком процедуры.
- Файл модуля:
- unit Unit1; *заголовок программы*
- interface *секция интерфейсных объявлений*
- uses *использовать*
- Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
- Dialogs;
- type *типы созданных компонентов и процедур*
- TForm1 = class(TForm) *в окне формы Form1*
- private *закрытая часть секции*
- { Private declarations }
- public *общедоступная часть секции*
- { Public declarations }
- end; *конец описания секций*
- var *строка описания глобальных переменных*
- Form1: TForm1;

- implementation *секция реализаций*
- {$R *.dfm}

тело программы

- end. *конец программы*

- Программа на ObjectPascal представляет собой последовательность операторов, разделенных символом «;».
- В текстах программ встречаются следующие основные элементы языка ObjectPascal:

1. Зарезервированные слова – это английские слова, определяющие действия (begin – начало, end – конец, IF – если, while – пока, repeat – повторять …)
2. Идентификаторы – имена, задаваемые пользователем. Могут состоять из латинских букв, цифр, знака подчеркивания, обычно это имена переменных.
3. Типы – специальные конструкции языка, образцы для создания других элементов. С помощью типов можно определить класс.
4. Класс – это образец, по которому создаются объекты. Пример:

```
Form1=class(Tform);
Button1:TBtn;
Label1:TLabel;
Edit1:TEdit;
private
public
end;
```

Здесь форма 1 это порожденный класс от родительского Tform – пустое окно.

- Новый класс обладает всеми свойствами родительского класса, но добавляет еще и свои, только ему присущие свойства – кнопка, метка, текстовое поле.

5. Объект – базовое понятие языка ObjectPascal, это специальный фрагмент программы, включающий данные и подпрограммы их обработки. Данные называются полями, а подпрограммы – методами. Объект обладает следующими свойствами:

   - функциональность;
   - неделимость;
   - свободное перемещение из одной программы в другую.

6. Константы – это неизменяемые области памяти, описываются в тексте программы командой:

   Пример: описание константы a=1,8 и b=1 в функции y=ax+b

   ```
   Const
   a=1.8;
   b=1;
   ```

7. Переменные – это изменяемые области памяти. Все переменные должны быть описаны с указанием их типа данных:

   Пример: Var x, y: single; команда var (variable – переменная), single – короткий вещественный тип данных.

8. Метки – имена операторов программы для перехода счета на некоторую строку. Имена задаются либо буквами или цифрами:

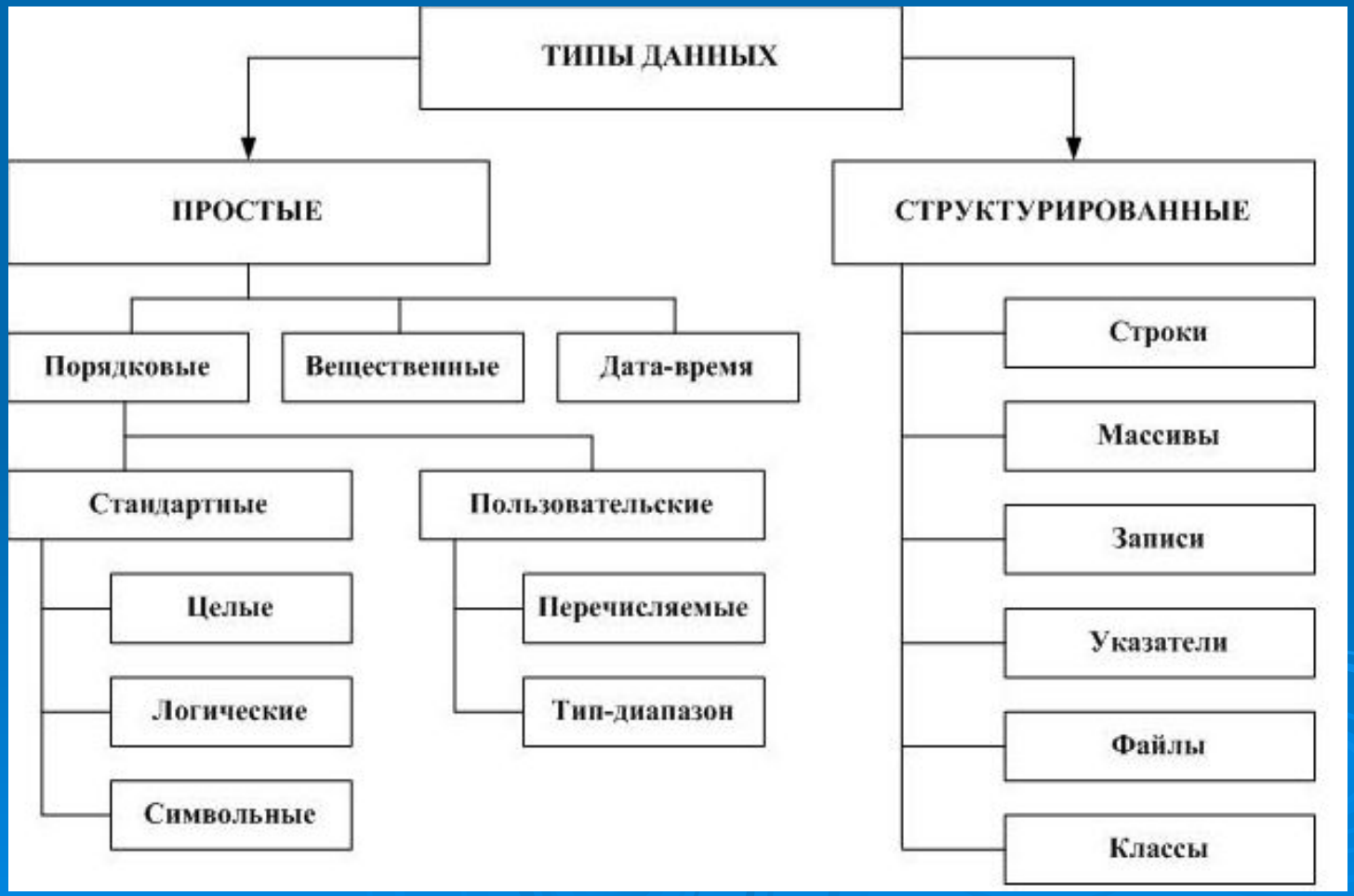
Label

Loop; 12;

9. Комментарии – это текстовые примечания, игнорируемые компилятором. В программы обозначаются одним из следующих способов: (* *), // , { }.

10. Подпрограммы (англ. *subroutine*) — поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограммы в ObjectPascal бывают в виде процедур и функций. Функция по окончанию своей работы возвращает один результат, а процедура – несколько результатов. Процедуры бывают с параметром и без параметра.

Любые данные, т.е. константы, переменные, свойства, значения функции или выражения, в Object Pascal характеризуются своими типами данных.



Наименование типа	**Обозначение в ObjectPascal**	**Диапазон применения**
Целый:		
целый	Integer	$\pm$ 2147483648
длинный целый	Longint	$\pm$ 2147483648
байт	byte	0……+65535
Логический:		
логический	boolean	1байт
длинный логический	longbool	4байт
Символьный:	char	Все символы
Вещественный:		
вещественный		
длинный	real	$5 \cdot 10^{-324}$……$1,7 \cdot 10^{308}$
короткий	single	$5 \cdot 10^{-45}$……$3,4 \cdot 10^{38}$
двойной	double	$5 \cdot 10^{-324}$……$3,4 \cdot 10^{308}$
Дата-время	TDateTime	8байт
Массивы	Array	не больше 2Гбайт
Строки	String[количество символов]	до 2Гб, 255 символов
Записи	Record	-

Для использования математических функций в строке файла модуля программы после **uses** требуется вписать слово **Math (подключение математического модуля)**.

Функция	Обращение в Object Pascal
$\|x\|$	ABS(X)
x^2	SQR(X)
$\sqrt{x}$	SQRT(X)
sinx	SIN(X)
cosx	COS(X)
e^x	EXP(X)
Lnx	LN(X)
tgx	TAN(X)
π	Pi
arctg x	ARCTAN(X)
Целая часть	TRUNC(X)
Округление	ROUND(X)
Нечетность	ODD(X)

Для вычисления остальных функций в Object Pascal используется пересчет:

$x^n := \exp(n \cdot Ln(x));$ *или* $x^n := power(x, n);$

$\lg x := \ln(x) / \ln(10);$

$\log_a x := \ln(x) / \ln(a);$

$arctgx := \arctan(1/x);$

$\arcsin x := \arctan((x) / sqrt(1 - sqr(x)));$

$\arccos x := \arctan(sqrt(1 - sqr(x)) / x);$

Алгоритм – точное и понятное предписание исполнителю совершать последовательность действий, направленных на решение поставленной задачи. Алгоритм – одно из основных понятий информатики и математики.

- **Формы представления алгоритмов.**

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная (записи на естественном языке);
- графическая (изображения из графических символов);
- псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- программная (текст на языке программирования Object Pascal).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

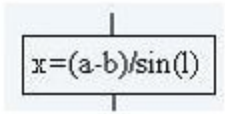
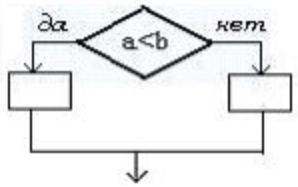
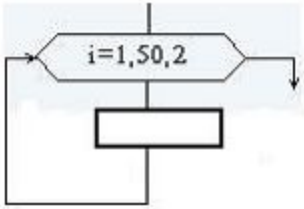
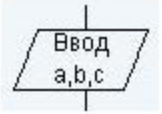
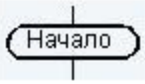
Словесный способ не имеет широкого распространения.

- **Графический способ** представления алгоритмов является более компактным и наглядным по сравнению со словесным.
- Такое графическое представление называется **схемой алгоритма** или **блок-схемой.**

При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.

Таблица основных блочных символов языка блок-схем.

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Модификация		Начало цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод-вывод		Ввод-вывод в общем виде
Пуск-останов		Начало, конец алгоритма, вход и выход в подпрограмму

- **Блок "процесс"** применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.
- **Блок "решение"** используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.
- **Блок "модификация"** используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.
- **Блок "предопределенный процесс"** используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных базовых (основных) элементов (блочных символов).

Для описания **алгоритмов** используется язык схем алгоритмов и алгоритмический язык Object Pascal.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур:

- **следование**;
- **ветвление**;
- **цикл**.

Характерной особенностью базовых структур является наличие в них **одного входа и одного выхода.**

Операторы языка Object Pascal, реализующие основные базовые структуры алгоритмов:

1. **Оператор присваивания**:

Имя := Выражение;

где **Имя** – переменная, значение которой изменяется в результате оператора присваивания.

:= – символ оператора присваивания.

Выражение состоит из операторов и операндов. В качестве операндов можно использовать: переменную, констант, функцию или другое выражение.

2. **Основные алгебраические операторы** следующие:

+ (алгебраическая сумма), **-** (разность) , * (произведение), / (частное) , **div** (деление нацело), **mod** (остаток от деления).

3. **Составной оператор** – это последовательность произвольных операторов, заключенных в операторные скобки:

- **Алгоритмический язык:**

```
begin
Оператор1(действие1);
Оператор2(действие2);
Оператор 3(действие3);
end;
```

- **Язык блок-схем:**
- **Базовая структура следование.** Образуется из последовательности действий, следующих одно за другим:

действие 1;
действие 2;
. . . . . .
действие n;

Свойства составного оператора:

- В него могут входить любые другие операторы;
- В него могут входить другие составные операторы;
- Число вложений не ограничено (рекурсия).

Рекурсия - процесс повторения элементов самоподобным образом.

Пример:

```
begin
begin
…….
end;
end;
```

4. **Условный оператор** (логический оператор) – позволяет проверять некоторое условие и в зависимости от результатов выполнять то или иное действие (последовательность действий(операторов)). Действие может принимать два значения: True(истина)/False(ложь).

- **Алгоритмический язык:**

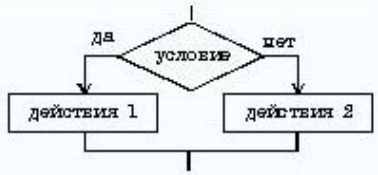
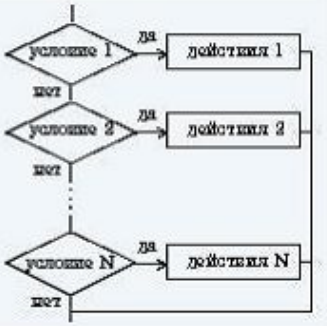
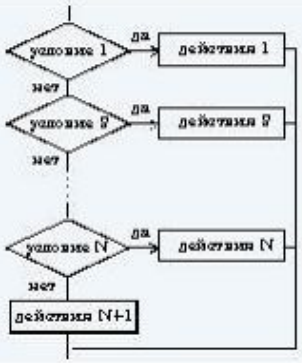
Структура записи логического оператора:

IF условие **THEN** действие1 (оператор1)

ELSE действие2 (оператор2);

- **Язык блок-схем:**
- **Базовая структура ветвление.** Обеспечивает в зависимости от результата проверки условия (да или нет) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к общему выходу, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

  Структура ветвление существует в четырех основных вариантах:
- **если-то;**
- **если-то-иначе;**
- **выбор;**
- **выбор-иначе.**

Алгоритмический язык	Язык блок-схем
если условие то действия все	*если-то* 
если условие то действия 1 иначе действия 2 все	*если-то-иначе* 
выбор при условие 1: действия 1 при условие 2: действия 2 . . . . . . при условие N: действия N все	*выбор* 
выбор при условие 1: действия 1 при условие 2: действия 2 . . . . . . при условие N: действия N иначе действия N+1 все	*выбор-иначе* 

Свойства условного оператора:

- Часть ELSE может быть опущена, тогда действие перейдет к следующему за IF оператору при значении false условного выражения.
- Сравниваться могут и два значения, для этого они объединяются в диапазон:

Пример: $1 \le x \le 2$

```
IF (1<=x) and (x<=2) THEN действие(оператор1)
ELSE действие(оператор2);
```

- Один условный оператор может вкладываться в другой и так далее:

Пример:

```
IF a<b THEN
IF c<d THEN
……
ELSE действие2(оператор2);
ELSE действие1(оператор1);
```

- Здесь любая встретившаяся часть ELSE соответствует ближайшей к ней сверху части THEN.

- Если на каждое условие требуется выполнить несколько действий, их заключают в операторные скобки:

Пример: Вычислить при x>0 – y=x, z=x+1, k=z+y

при x<=0 – y=x-1, z=x+2, k=z-y

Блок логического оператора запишется:

```
IF x>0 THEN
begin
y:=x;
z:=x+1;
k:=z+y;
end;
ELSE
begin
y:=x-1;
z:=x+2;
k:=z+y;
end;
```

- **Логический оператор** помимо проверки математических условий может работать и с классами. Для этого используется свойство класса, имеющее логический тип – boolean. Часто это бывает при обработке зависимых и независимых переключателей, свойство логического типа в данном случае – checked (проверка на истинность логического выражения).

- При обработке зависимого переключателя RadioButton логический оператор будет иметь вид:

  IF RADIOBUTTON1.CHECKED=TRUE THEN действие по первому зависимому переключателю RADIOBUTTON1;

- При обработке независимого переключателя CheckBox логический оператор будет иметь вид:

  IF CHECKBOX1.CHECKED THEN действие по первому независимому переключателю CHECKBOX1;

5. Операторы цикла (циклические операторы).

- Операторы цикла получили свое название потому, что позволяют многократно (циклически) выполнять один и тот же оператор или группу операторов.
- **В языке Object Pascal существует три оператора цикла:**

- 5.1. Оператор цикла с предусловием:
- Цикл с предусловием — цикл, который выполняется пока истинно некоторое условие, указанное перед его началом. Это условие проверяется **до** выполнения тела цикла, поэтому тело может быть не выполнено ни разу (если условие с самого начала ложно). В большинстве процедурных языков программирования реализуется оператором **while**, отсюда его второе название — while-цикл.
- **Алгоритмический язык:**

  WHILE условие DO действие (оператор);
- Оператор можно прочесть «пока выполняется условие – выполнять действие(делать)».
- Пример: Вычислить значение функции $y=\sqrt{x}$, пока x=>0.
- WHILE x=>0 DO y:=SQRT(x);

- Если действий (операторов) несколько, то эти операторы заключают в операторные скобки:
- Пример: Вычислить значения трех функции y=√x , z=cosx , n=tgx, пока x=>0.

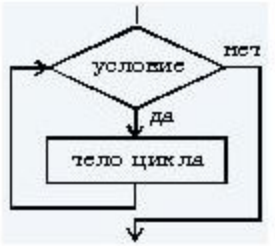
```
WHILE x=>0 DO
begin
y:= sqrt(x);
z:=cos(x);
n:= tan(x);
end;
```

Язык блок-схем:

- **Базовая структура цикл.** Обеспечивает многократное выполнение некоторой совокупности действий (операторов), которая называется телом цикла.
- Оператору цикла с предусловием соответствует следующая базовая структура цикла типа пока:

```
WHILE условие DO
begin
действие1 (оператор1);
действие2 (оператор2);
действие3 (оператор3);
……….
действие n (оператор n);
end;
```

тело цикла

Алгоритмический язык	Язык блок-схем
Цикл типа пока Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока.	
нц пока условие тело цикла (последовательность действий) кц	

- 5.2. Оператор цикла с постусловием:
- Цикл с постусловием — цикл, в котором условие проверяется **после** выполнения тела цикла. Отсюда следует, что тело **всегда выполняется** хотя бы один раз
- **Алгоритмический язык:**

  REPEAT тело цикла UNTIL условие;
- Дословно оператор можно прочесть «повторять до тех пор, пока не будет выполнено условие». Условие выхода из этого цикла обратно алгоритмическому.
- Пример: Вычислить значение функции $y=\sqrt{x}$, при $x>=0$.

  REPEAT y:=SQRT(x) UNTIL x<=0;

- Если действий (операторов) несколько, то тело цикла заключать в операторные скобки не требуется.
- Пример: Вычислить значения трех функции y=√x , z=cosx , n=tgx, при x>=0.
- REPEAT

  y:= sqrt(x);

  z:=cos(x);

  n:= tan(x);

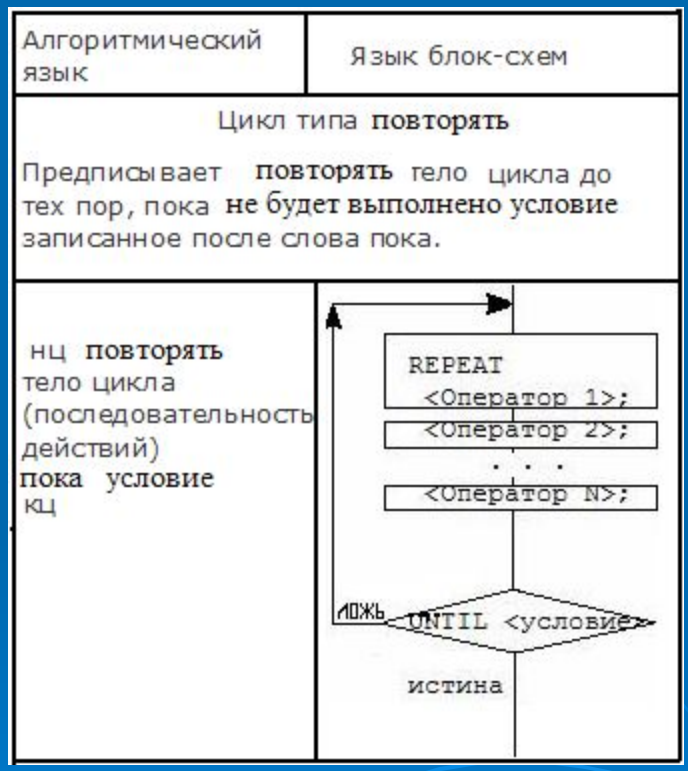
  UNTIL x<=0;

Язык блок-схем:

- Оператору цикла с постусловием соответствует следующая базовая структура цикла типа повторять:

REPEAT
действие1 (оператор1);
действие2 (оператор2);
действие3 (оператор3);
……
действие n (оператор n);
UNTIL условие;

тело цикла



- 5.3. Оператор цикла со счетчиком (с параметром):
- Цикл со счётчиком — цикл, в котором некоторая переменная изменяет своё значение от заданного начального значения до конечного значения с некоторым шагом, и для каждого значения этой переменной тело цикла выполняется один раз.
- **Алгоритмический язык:**

FOR Параметр цикла:=начальное значение TO конечное значение

DO действие (оператор);

FOR i:=1 TO N DO действие (оператор);

где i – параметр цикла со счетчиком, N – конечное значение параметра.

- Дословно можно прочесть «для… равного…до … выполнять(действие) …».
- Например: Сделать 10 раз нечто.

FOR i:=1 TO 10 Нечто;

- Если действий несколько, то они заключаются в операторные скобки.

Например: Вычислить сумма чисел заданной последовательности x1, x2, x3, ……xn (из 10 чисел), начиная с первого заданного значения.

```
x:=xn;
 FOR i:=1 TO N DO
begin
sum:=sum+x;
Memo1.Lines.Add('x='+floattostr(x)+'y='+floattostr(y));
x:=x+1;
end;
```

тело цикла

- Шаг оператора цикла со счетчиком постоянен и равен +1. Для отрицательного шага (-1) используется DOWNTO вместо TO.

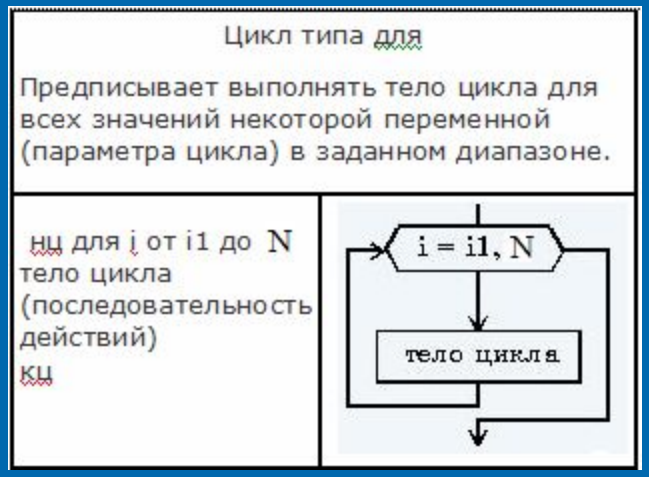
FOR i:=1 DOWNTO N DO действие (оператор);

Язык блок-схем:

- Оператору цикла со счетчиком (параметром) соответствует следующая базовая структура цикла типа для:

```
FOR i1:=1 TO N DO
begin
действие1 (оператор1);
действие2 (оператор2);
действие3 (оператор3);
……….
действие n (оператор n);
end;
```

тело цикла



- где i1 – начальный параметр цикла (равный 1), N – конечный параметр цикла.
- Построение таблицы значений функции y=f(x) на отрезке xе[xn;xk] с шагом hx – является типичной задачей с циклом.
- где xn – начальное значение, xk – конечное значение.

- Организация **ввода/вывода** используется в каждой программе.
- В Delphi **ввод/вывод** можно производить в следующих компонентах:

1. **Текстовые поля (поля редактирования) - TEdit**  ;
2. **Метки – TLable**  ;
3. **Многострочные поля ввода/вывода – Memo**  ;
4. **Таблицы – TStringGrid**  .

- Для **ввода/вывода** чисел в текстовых полях используется сочетание двух элементов – метки и поля текста. У метки в окне инспектора объектов изменяют свойство заголовка Caption, где вводят сообщение о вводимой величине, у текстового поля из свойства Text обычно удаляют содержимое (свойство заголовка текстового поля TEdit). Этому способу **ввода/вывода** данных присуща следующая особенность: при вводе в текстовом поле данные всегда будут иметь строковый тип – **string**, поэтому требуется преобразовать его к требуемому типу данных.

- Для преобразования из строкового типа данных в требуемый тип данных при **вводе/выводе** в **ObjectPascal** используются следующие функции преобразования:

1. При **вводе/выводе** вещественного типа данных (single, real, double, currency, extended):

STRTOFLOAT / FLOATTOSTR

2. При **вводе/выводе** целого типа данных (integer, longint, smallint, byte):

STRTOINT / INTTOSTR

- **1. Ввод/вывод в текстовое поле TEdit.**
- Преобразование переменной из строкового типа данных к вещественному при вводе в Edit1:
- Вещественный тип данных:

  Переменная := strtofloat(edit1.text);
- Целый тип данных:

  Переменная := strtoint(edit1.text);
- Пример: Ввод значения переменной x в текстовое поле Edit1: x:=strtofloat(edit1.text); или x:=strtoint(edit1.text);
- Преобразование переменной из вещественного типа данных к строковому при выводе:

  Edit1.text := floattostr(переменная);
- Преобразование переменной из целого типа данных к строковому при выводе в Edit1:

  Edit1.text := inttostr(переменная);
- Пример: Вывод значения переменной y в текстовое поле Edit2:

  Edit2.text := floattostr(y); или Edit2.text := inttostr(y);

- Преобразование переменной из вещественного типа данных к строковому при выводе:

  Edit1.text := floattostr(переменная);
- Преобразование переменной из целого типа данных к строковому при выводе в Edit1:

  Edit1.text := inttostr(переменная);
- Пример: Вывод значения переменной y в текстовое поле Edit2:

  Edit2.text := floattostr(y); или Edit2.text := inttostr(y);
- Для **вывода** в текстовом поле Edit1 сообщения используется конструкция:

  Edit1.text :=‘Текст сообщения’;

- **2. Ввод/вывод в текстовую метку TLabel.**
- Преобразование переменной из вещественного к строковому типу данных при вводе в Label1:
- Вещественный тип данных:

  Переменная := strtofloat(label1.caption);
- Целый тип данных:

  Переменная := strtoint(label1.caption);
- Пример: Ввод значения переменной x в текстовое поле Label1: x:=strtofloat(label1.caption); или x:=strtoint(label1.caption);
- Преобразование переменной из вещественного типа данных к строковому при выводе:

  label1.caption := floattostr(переменная);
- Преобразование переменной из целого типа данных к строковому при выводе в Label2:

  label2.caption := inttostr(переменная);
- Пример: Вывод значения переменной y в текстовое поле Label2:

  label2.caption := floattostr(y); или label2.caption:= inttostr(y);

- Пример: Вывод значения переменной y в текстовое поле Label2:

  label2.caption := floattostr(y); или label2.caption:= inttostr(y);
- Если требуется вывести сообщение в метке Label1 используется конструкция:

  Label1.Caption :=‘Текст сообщения’;

- **3.** Для **ввод/вывод** в многострочном редакторе **TMemo** обычно изменяют сложное свойство Lines. Многострочный редактор, так же как и поле ввода, может работать только со строковыми переменными, поэтому при вводе/выводе используются те же функции преобразования к типу данных.
- Для ввода в многострочном редакторе у строки Lines используется свойство Count(счет) – это число введенных строк начиная с 0 строки.
- Например, чтобы ввести число из последней строки редактора команда записывается:

```
Переменная := Memo1.Lines[Memo1.Lines.Count-1];
```

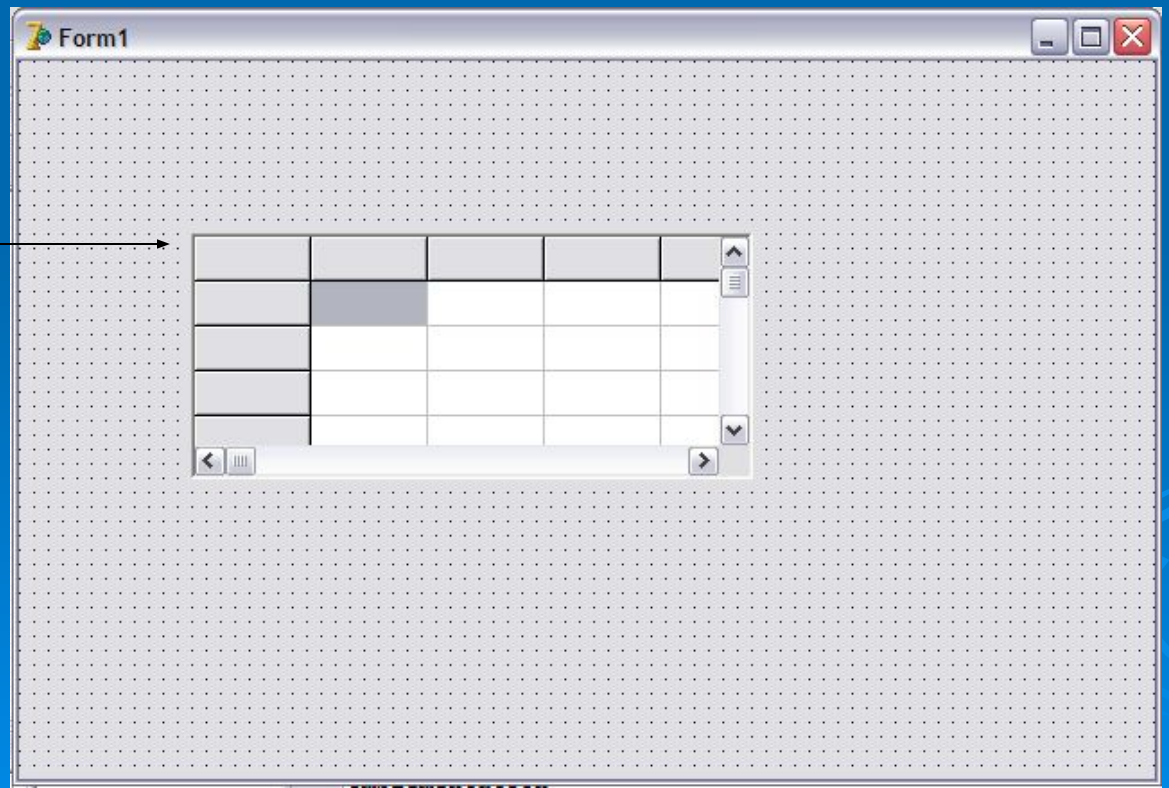
- Для **вывода** в многострочном редакторе значений переменных x и y используется конструкция:

```
Memo1.Lines.Add('x='+floattostr(x)+'y='+floattostr(y));
```

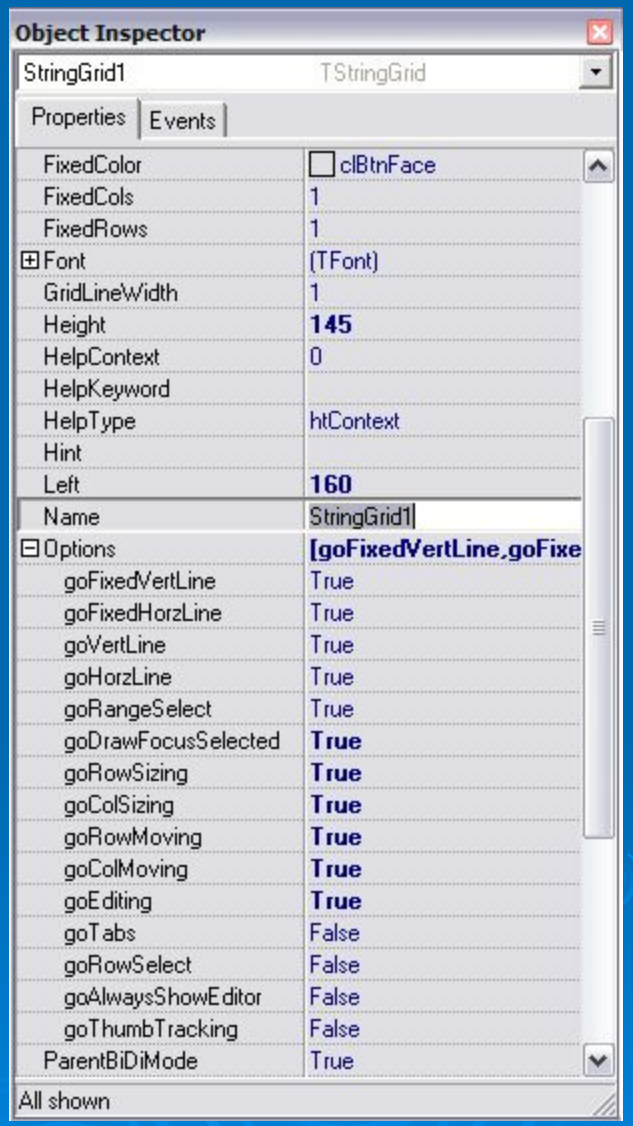
- Для **вывода** в многострочном редакторе значений переменных x и y используется конструкция:
- Для вещественного типа данных:
  Memo1.Lines.Add(‘x=‘+floattostr(x)+‘y=‘+floattostr(y));
- Для целого типа данных:
  Memo1.Lines.Add(‘x=‘+inttostr(x)+‘y=‘+inttostr(y));

- Для вывода в диалоговое окно сообщения используется следующая конструкция:
  ShowMessage(‘Текст сообщения’);

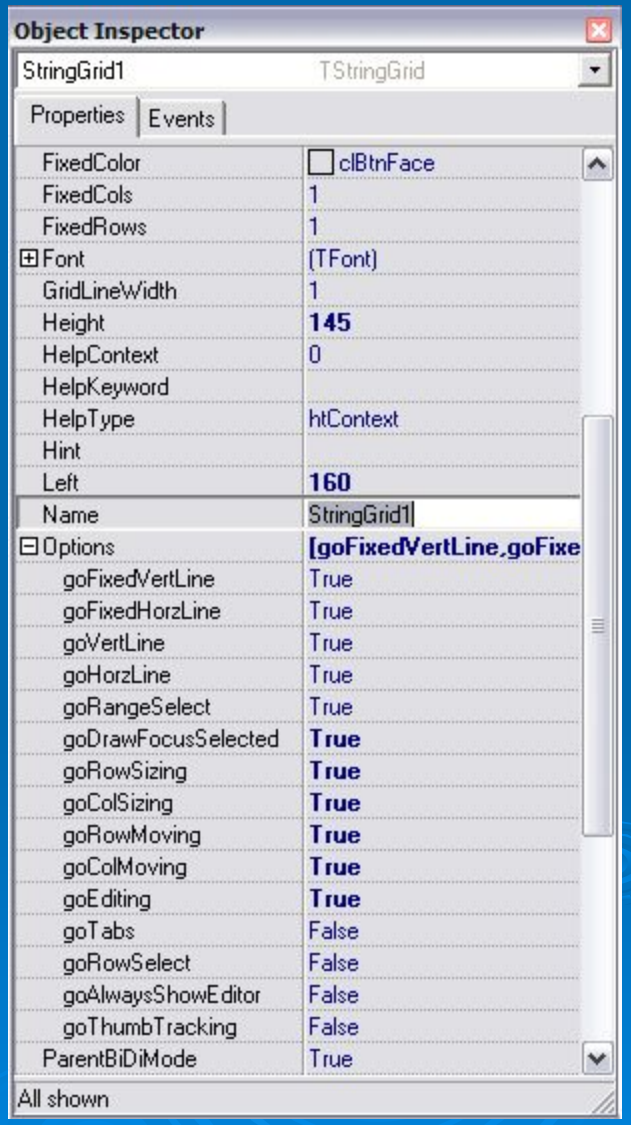
- **4. Свойства TStringGrid. Ввод/вывод** данных в таблице **TStringGrid.**
- **Компонент TStringGrid**  **находится на палитре компонентов главного меню (MainMenu) в панели Additional и является электронной таблицей, которая работает со строковыми типами данных string.**
- 



- Основные свойства компонента TStringGrid, назначаемые в окне ObjectInspector:
- I. Основные простые свойства компонента TStringGrid :
- 1. ColCount – количество столбцов таблицы, начиная с нулевого, в цикле номер столбца задается переменной j, количество столбцов - переменной n.
- 2. RowCount – количество строк таблицы, начиная с нулевого, в цикле номер строки задается переменной i, количество строк - переменной m.
- 3. Зафиксированное под шапку таблицы:
- FixedCols – число столбцов, FixedRows – число строк.



- II. Основные сложные свойства компонента (логического типа **True/False**) TStringGrid в строке [+] Options:
- 1. goColSizing – изменение размера столбцов;
- 2. goRowSizing – изменение размера строк;
- 3. goRowMoving – перестановка строк местами;
- 4. goColMoving – перестановка столбцов местами;
- 5. goEditing – редактирование данных (строкового содержимого) в ячейках таблицы.



- **4. Ввод/вывод** данных в таблице **TStringGrid**  .
- Для организации доступа к полям таблицы компонента TStringGrid в ObjectPascal используется оператор:

WITH переменная DO действие(оператор);

- Дословно оператор читается:
- “С переменной делать(выполнять) …действие(оператор)”.
- Если действие описывается несколькими операторами, то они объединяются операторными скобками begin …end:

```
WITH переменная DO
 begin
 действие 1 (оператор 1);
 действие 2 (оператор 2);
 …
действие n (оператор n);
end;
```

- Для записи или считывания данных из ячеек таблицы компонента TStringGrid  используется свойство:
- Cells[n,m], где n – номер столбца, m – номер строки.
- Ввод в таблицу осуществляется командой:
  StringGrid1.Cells[n,m]:=‘переменная(содержимое)’;
- Данные, размещаемые в таблице, могут быть только строкового типа, поэтому для расчетов следует использовать известные функции преобразования к типу данных.

- Для внесения заголовков, приведенных на примере рис.1, в шапку и нумерации строк и столбцов компонента StringGrid1 используется следующий фрагмент кода:

```
procedure TForm1.FormCreate(Sender: TObject);
var i,j:integer;
begin
 with Form1.StringGrid1 do
begin
Cells[0,0]:='N';
Cells[1,0]:='X';
Cells[2,0]:='Y';
for i:=1 to 12 do
Cells[0,i]:=Inttostr(i);
for j:=3 to 16 do
Cells[j,0]:=Inttostr(j);
end;
end;
end.
```

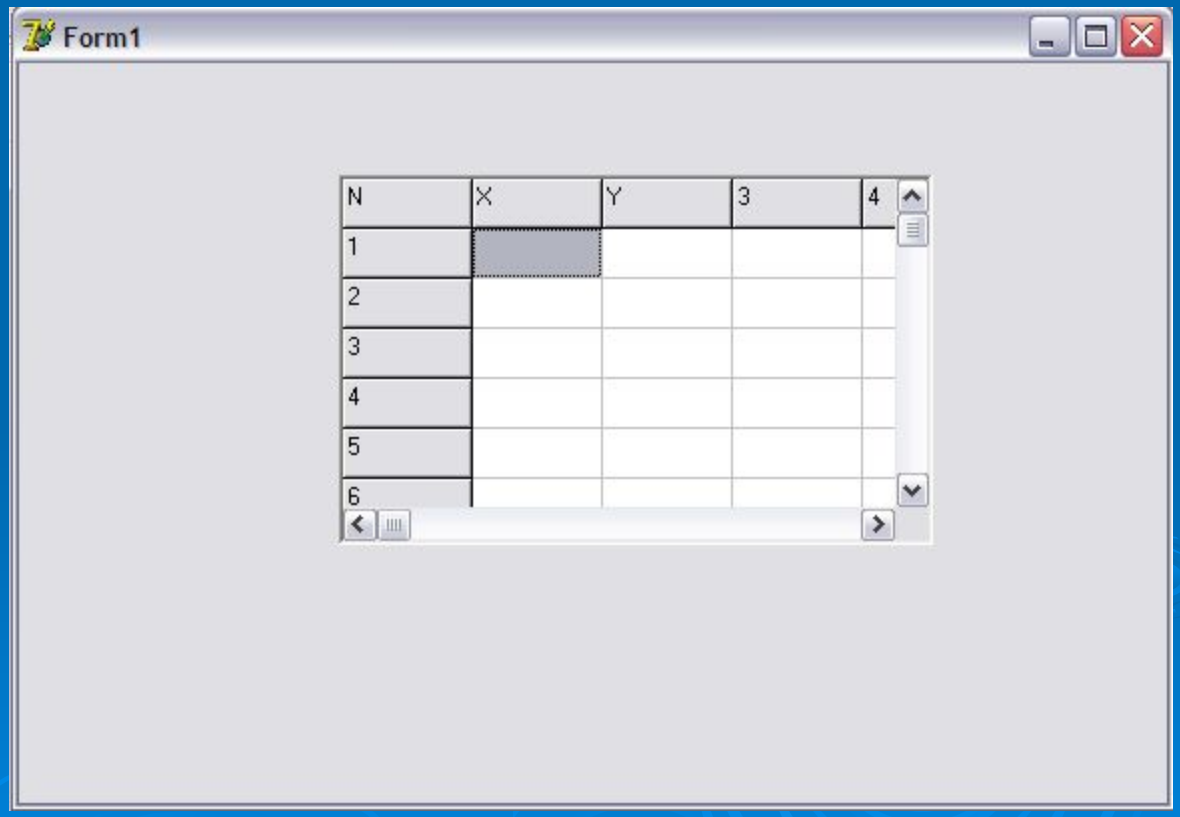


Рис. 1

- Строки и столбцы требуется пронумеровать подряд на примере рис. 2, причем количество строк – 16, столбцов – 16, тогда фрагмент кода следующий:

```
procedure TForm1.FormCreate(Sender: TObject);
var i,j:integer;
begin
 with Form1.StringGrid1 do
begin
RowCount:=16;
ColCount:=16;
Cells[0,0]:='N';
for i:=1 to 12 do
Cells[0,i]:=Inttostr(i);
for j:=1 to 16 do
Cells[j,0]:=Inttostr(j);
end;
end;
end.
```

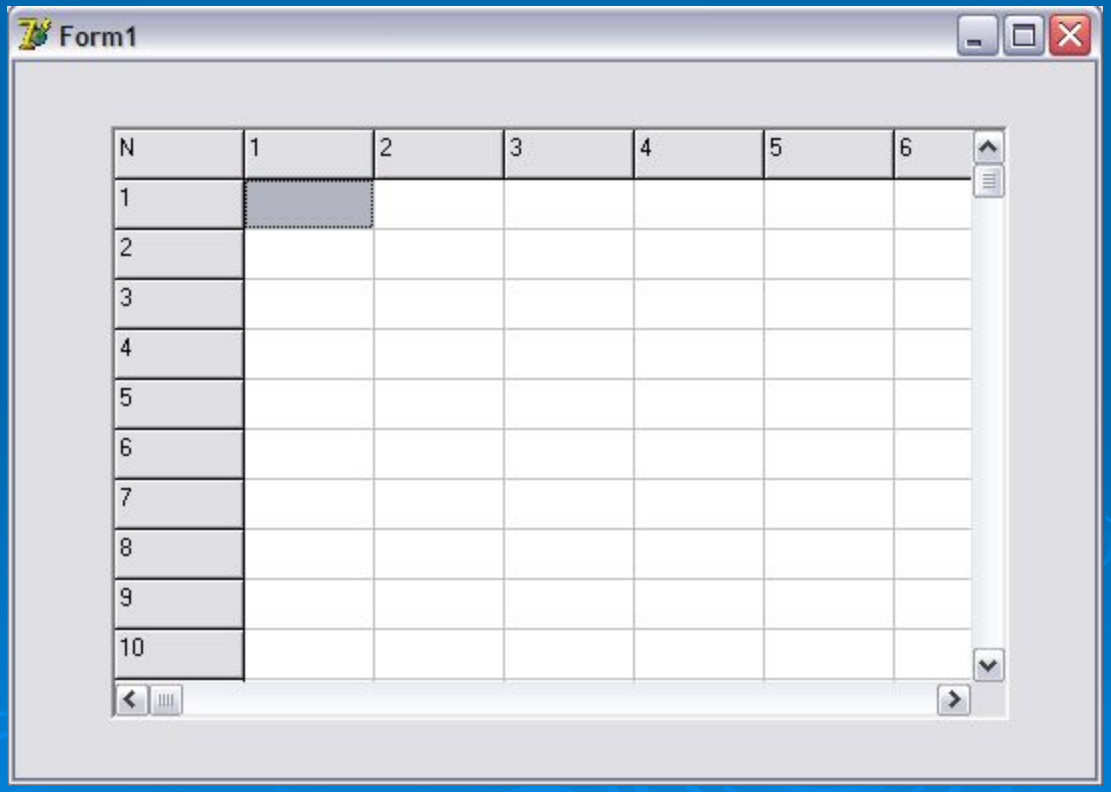


Рис. 2

- Очистка ячеек в таблице используется часто перед запуском программы или в ее начале. В следующем примере рис. 3 очищаются все ячейки таблицы:

```
procedure TForm1.FormCreate(Sender: TObject);
var i,j:integer;
begin
 with Form1.StringGrid1 do
begin
RowCount:=16;
ColCount:=16;
Cells[0,0]:='N';
for i:=1 to 12 do
for j:=1 to 16 do
Cells[j,i]:=' ';
end;
end;
end.
```

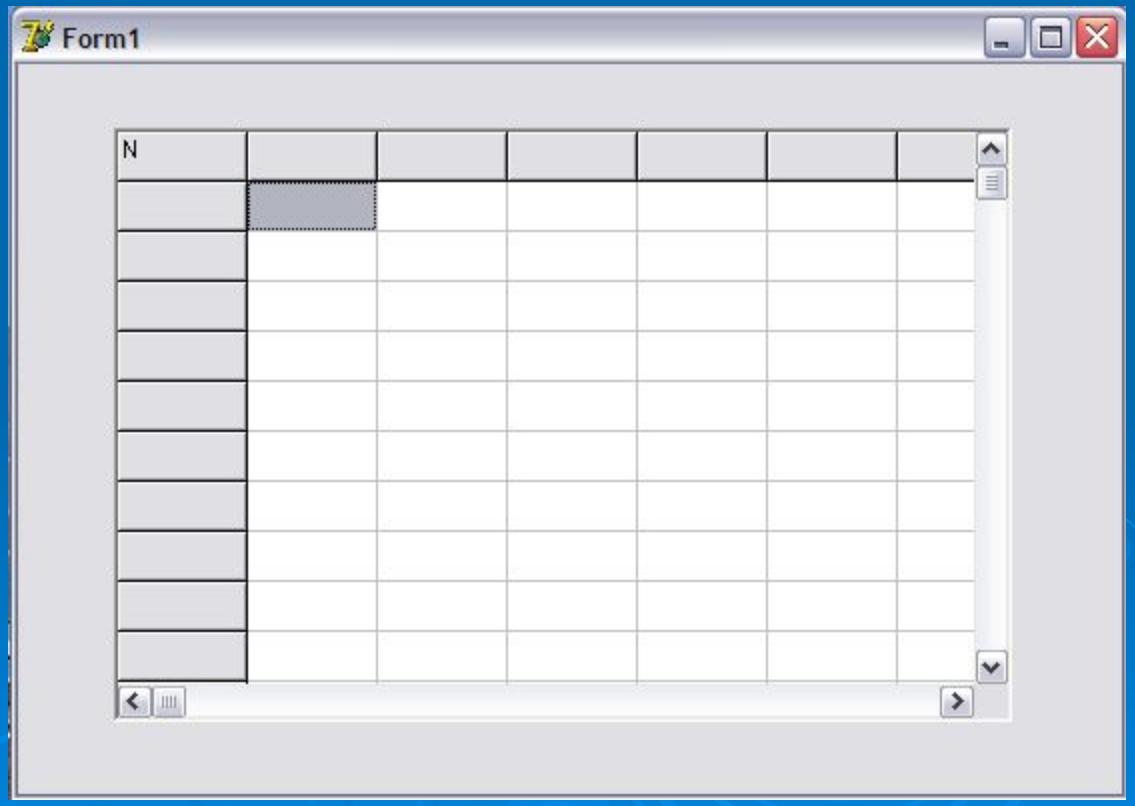


Рис. 3

- Вывод в таблицу StringGrid1 двух чисел x и y (x – в первую строку, y – во вторую строку) производится фрагментом кода:

```
procedure TForm1. BitBtn1Click (Sender: TObject);
var x,y:real;
    i,j:integer;
begin
 with Form1.StringGrid1 do
begin
RowCount:=16;
ColCount:=16;
StringGrid1.Cells[1,1]:=floattostr(x);
StringGrid1.Cells[1,2]:=floattostr(y);
end;
end;
end.
```

- Вывод в таблицу StringGrid1 последовательности двух векторов a[6] и b[6] (первый выводится в первом столбце, второй – во втором) :

```
procedure TForm1. BitBtn1Click (Sender: TObject);
var a[i], b[i]:array;
begin
 with Form1.StringGrid1 do
begin
For i:=1 to 6 do
StringGrid1.Cells[1,i]:=floattostr(a[i]);
StringGrid1.Cells[2,i]:=floattostr(b[i]);
end;
end;
end.
```

- Вывод числа с форматированием. Например, переменная x и y при выводе в таблицу должна иметь два знака после запятой:

```
procedure TForm1. BitBtn1Click (Sender: TObject);
var x,y:real;
begin
 with Form1.StringGrid1 do
begin
StringGrid1.Cells[1,1]:=floattostr(x,ffFixed,6,2);
StringGrid1.Cells[1,2]:= floattostr(y,ffFixed,6,2);
end;
end;
end.
```

- Ввод в таблицу StringGrid1 последовательности двух векторов a[i] и b[i] следующим фрагментом кода:

```
procedure TForm1. BitBtn1Click (Sender: TObject);
var a[i], b[i]:array;
begin
 with Form1.StringGrid1 do
begin
For i:=1 to 6 do
a[i]:=StrtoFloat (Cells[1,i];
b[i]:=StrtoFloat(Cells[2,i];
end;
end;
end.
```

- Для создания (построения) различных типов диаграмм используется компонент Chart. Размещается данных компонент Chart на вкладке Additional палитры компонентов главного окна Delphi:

- Размещенный на форму Form1 компонент Chart представляет собой панель (прямоугольную область рис. 5), на которой можно создавать диаграммы и графики различных типов, таких как:
- График (Line);
- Гистограмма (Bar);
- Горизонтальная гистограмма (Horiz. Bar);
- Диаграмма с областями (Area);
- Точечная диаграмма (Point);
- Круговая диаграмма (Pie);
- Линейный график (базовый) (Fast Line);
- Диаграмма форм (Shape);
- Биржевая (бизнес) диаграмма(Gantt);
- Диаграмма стрелок (Arrow);
- Пузырьковая (Bubble).

- Один компонент Chart можно использовать для создания сразу нескольких диаграмма(графиков). В этом смысле его можно рассматривать как координатную плоскость, на которой нанесены несколько кривых. Каждый из графиков является отдельным объектом Series (серия данных – Series1, Series2) и может иметь различные стили отображения.

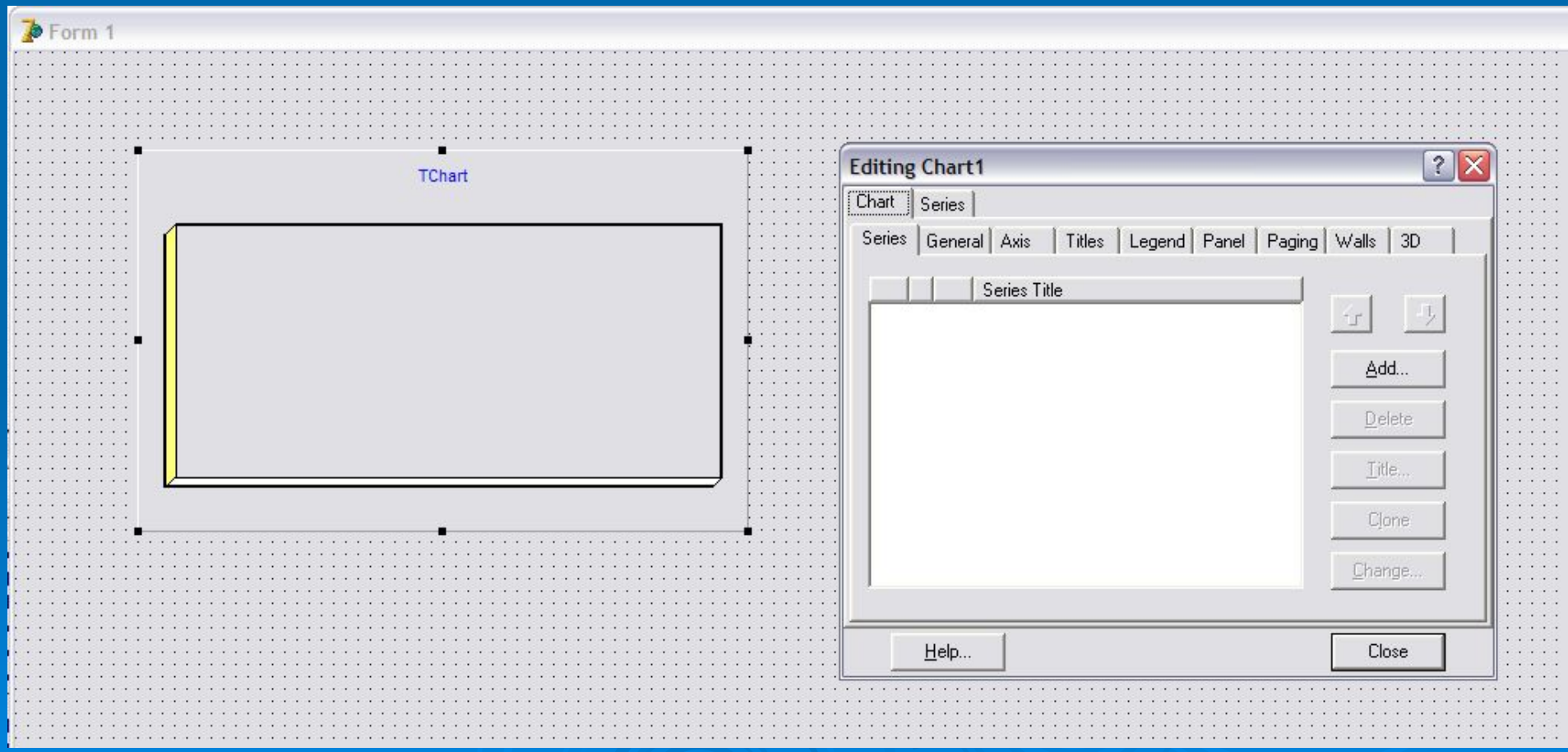
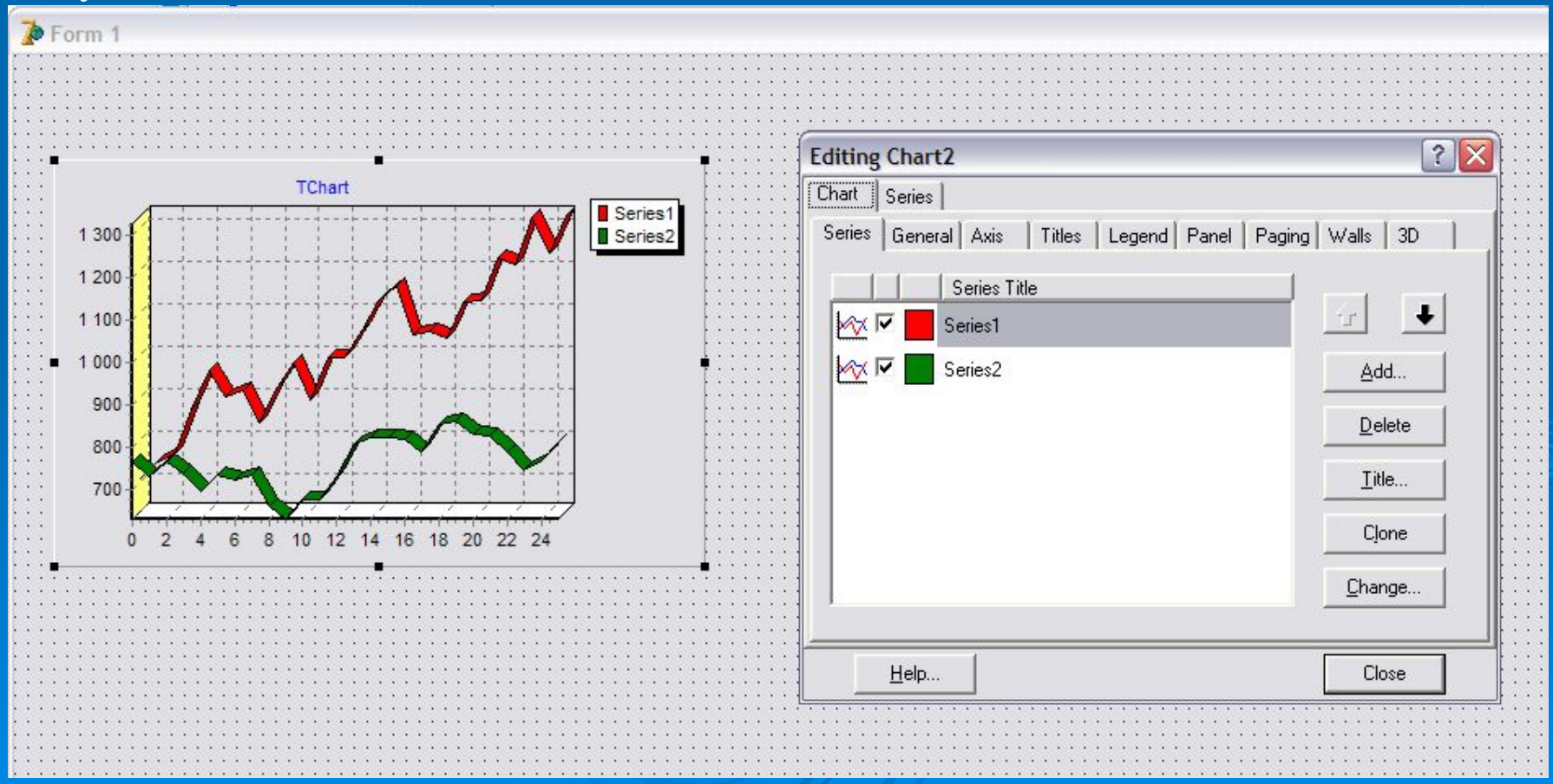
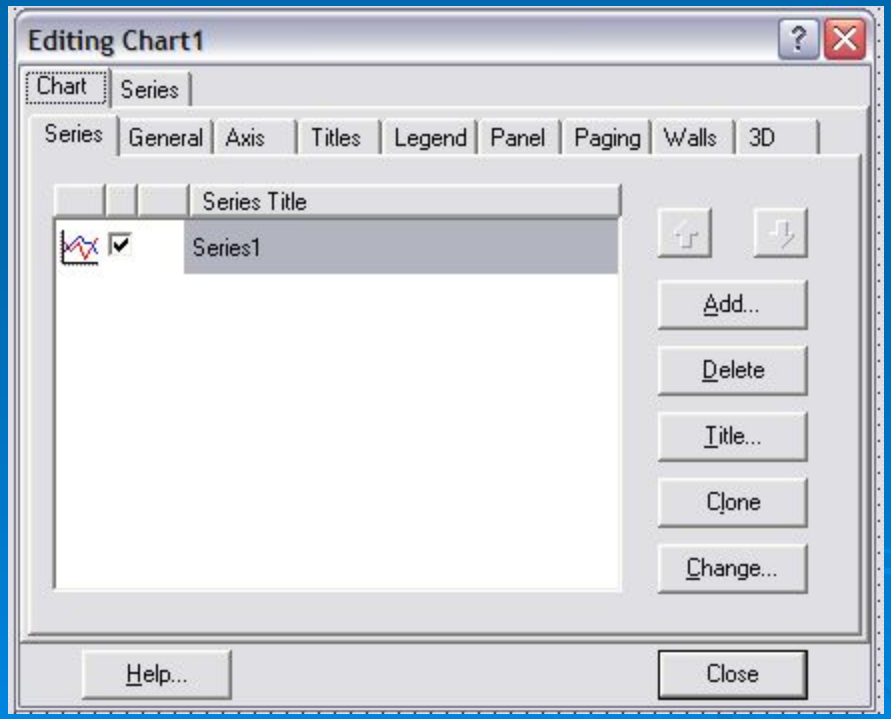


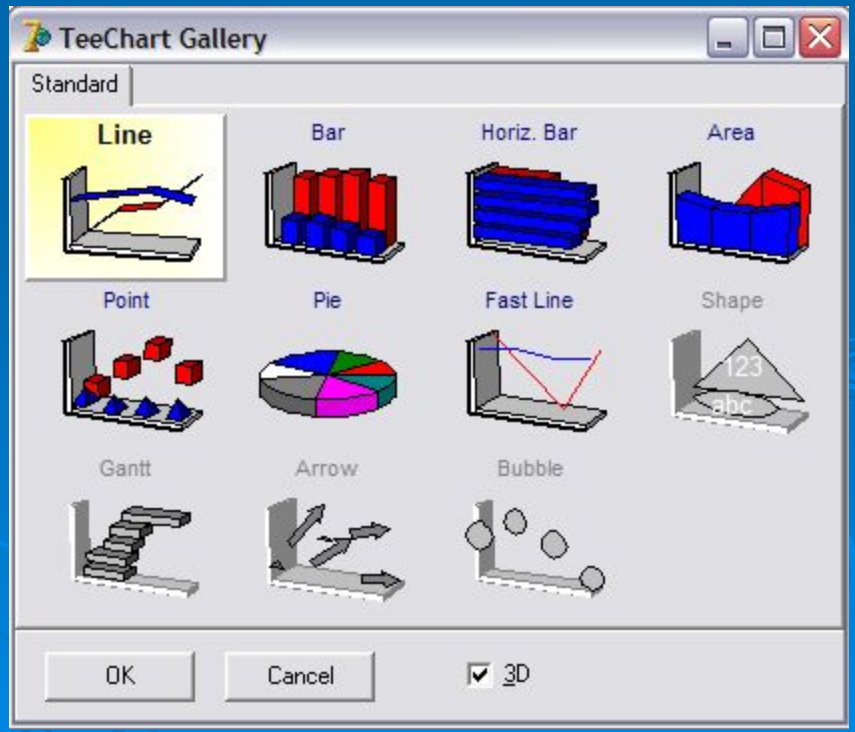
Рис. 5

- Порядок работы с компонентом Chart1 на этапе формы следующий:
- 1. Нанести на форму компонент Chart1 и откорректировать его линейные размеры.
- 2. Открыть Редактор диаграмм (свойство SeriesList или двойной щелчок на компоненте). Страница редактора имеет много вкладок, по умолчанию установлена на вкладке Series.

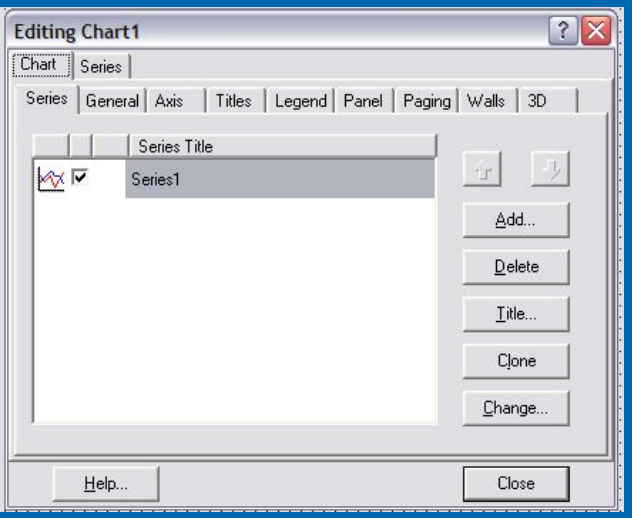


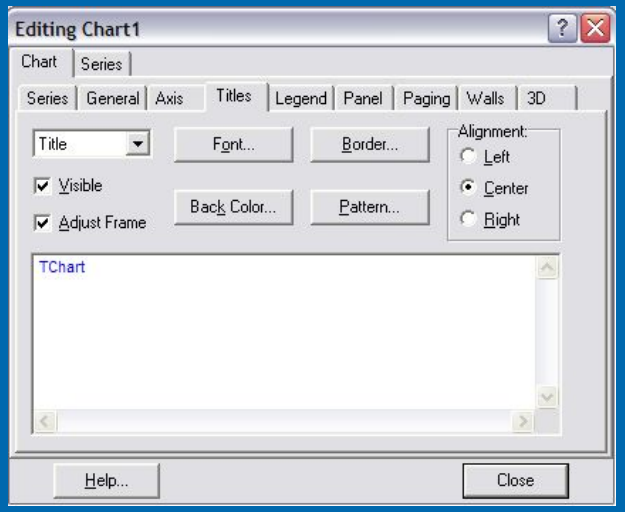
- 3. На вкладке Series нажать на кнопку Add(добавить), которая добавляет новую кривую (Series1 – серию данных) на график.
- 4. В раскрывшемся списке выбрать желаемый тип диаграммы или графика. Закрыть список. На форме появится график случайным образом заданной функции.
- 5. Перейти на вкладку Titles(заголовок), где задать требуемый заголовок (название) диаграммы.
- 6. Перейти на вкладку Legend, задать параметры отображения легенды диаграммы или совсем убрать ее с экрана.

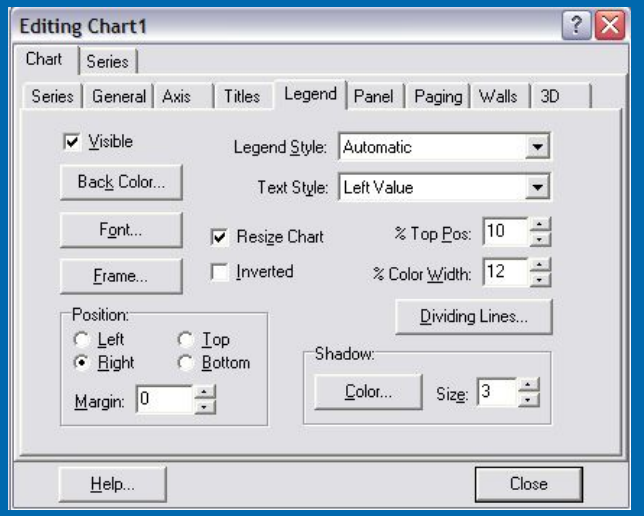


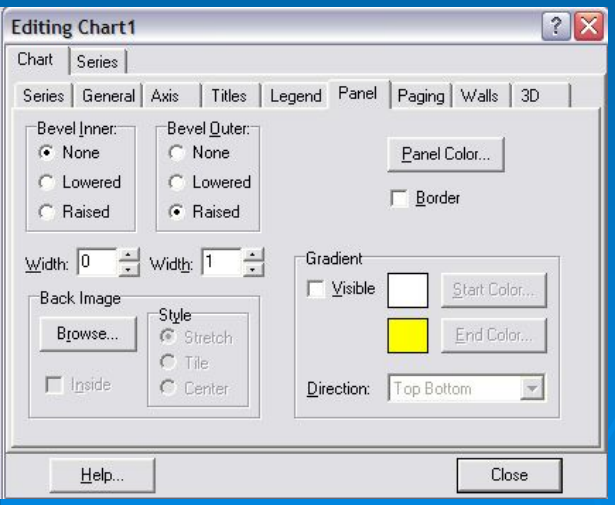


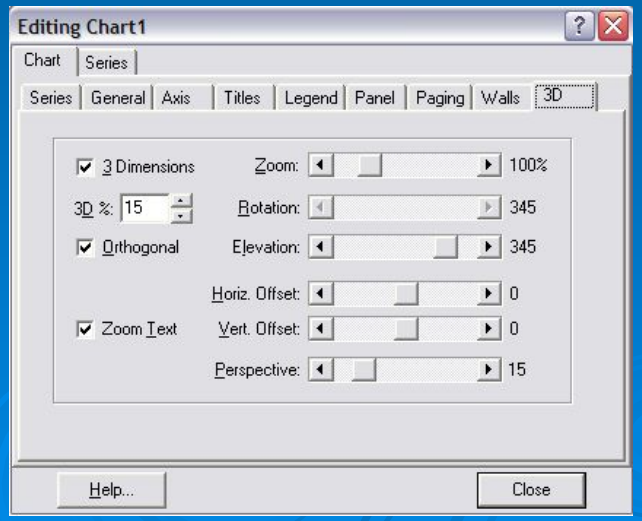
- 7. Перейти на вкладку Panel, откорректировать внешний вид панели Chart1.
- 8. Перейти на вкладку 3D, откорректировать вид графика: наклон, сдвиг, толщину.
- 9. Если необходимо, перейти на страницу Series и задать дополнительные характеристики отображения кривой графика.











- Для работы с компонентом из программы используются методы объекта Series.
- Исключение кривой с графика компонента Chart – метод Clear:

  Series1.Clear;
- Добавление на диаграмму новой точки – метод Add в общем виде:

  Series1.Add(A,S,Color);

  где A – выражение, задающее числовое значение точки на диаграмме,

  S – строка символов, содержащая пояснительную надпись к точке,

  Color – цвет точки, определенный при помощи стандартных цветовых констант. Например, следующие операторы очищают диаграмму и наносят на график три новых значения, задавая отображающие их цвета:

```
with Series1 do
begin
Clear;
Add(a1,'цех1',clRed);
Add(a2,'цех2',clYellow);
Add(a3,'цех3',clBlack);
end;
```

- Добавление новой точки на график функции – метод AddXY в общем виде:

Series1.Add(x,y,S,Color);

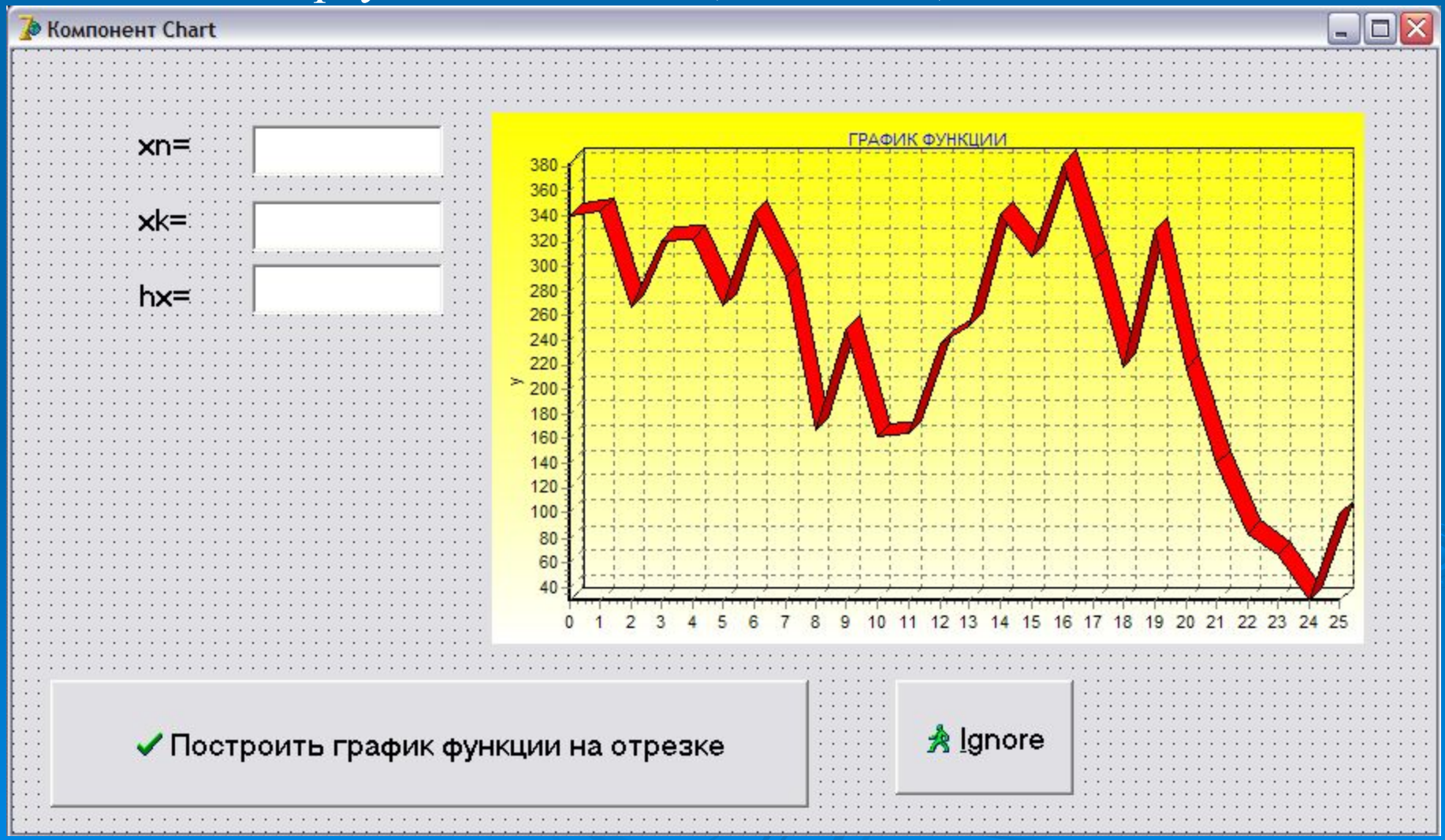
где x,y – выражения, задающие координаты точки на диаграмме (графике),

S – строка символов, содержащая пояснительную надпись к точке, как правило пустая строка, Color – цвет точки, определенный при помощи стандартных цветовых констант. Например, следующие операторы удаляют прежнюю кривую и наносят на график кривую синуса:

```
Series2.Clear;
For i:=0 to 100 do
Series2.Add(i,sin(i),’ ’,clRed);
```

Интерфейс программы:

- Пример: Построить график функции y=exp(x), используя компонент Chart, на длине отрезка – a=80, xe[xn;xk], с шагом изменения аргумента hx=a/100, xn=-a+hx, xk=a-hx.



Исходный код программы:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var x,a,y,xn,xk,hx:real;
begin
 a:=80;
hx:=a/100;
xn:=-a+hx;
xk:=a-hx;
x:=xn;
Series1.clear;
repeat
y:=exp(x);
Series1.AddXY(x,y,' ',clred);
x:=x+hx;

until x>xk;

end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
close;
end;

end.
```

- Окно проекта Project1.exe: