МИНЕСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА
РОССИЙСКОЙ ФЕДЕРАЦИИ

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ АРГАРНЫЙ УНИВЕРСИТЕТ –
МСХА имени К.А. ТИМИРЯЗЕВА

А.Л. Соколов

Информатика

Учебно-методическое пособие

Москва

2017

УДК 004(075.8)
ББК 32.81 я 73
С-59

Рецензент:
д.х.н., профессор **К.В. Кольцов**

Соколов А.Л. Информатика: учебно-методическое пособие /

С 59 А.Л. Соколов. – М.: ФГБНУ «Росинформагротех», 2017. – 101 с.

ISBN

Издание содержит учебно-методический материал для самостоятельной работы студентов. В нем представлены основные разделы информатики, необходимые будущим специалиста в практической деятельности. Теоретический материал дополнен примерами и заданиями для самостоятельного выполнения.

Предназначено для бакалавров факультета Техносферной безопасности, экологии и природопользования обучающихся по направлению Техносферная безопасность. Может быть также использовано студентами других направлений и профилей, проходящих курс обучения в аграрных и технических университетах.

Рекомендовано к изданию методической комиссией факультета Техносферной безопасности, экологии и природопользования РГАУ-МСХА имени К.А. Тимирязева, протокол №3 от 19 октября 2017 г.

УДК **004(075.8)**
ББК **32.81 я 73**

ISBN

ВВЕДЕНИЕ

Настоящее учебно-методическое пособие составлено в соответствии с рабочей программой учебной дисциплины Информатика для подготовки бакалавров по направлению: **Техносферная безопасность.** В ней нашли отражение основные темы рабочей программы: «Электронные таблицы Excel» и «Система объектно-ориентированного программирования Delphi».

Дисциплина «Информатика» включена в обязательный перечень федеральный государственный образовательный стандарт (ФГОС) дисциплин базовой части. Дисциплина «Информатика» реализуется в соответствии с требованиями ФГОС, основной профессиональной образовательной программы высшего образования (ОПОП ВО) и Учебного плана по направлению (профилю подготовки) Техносферная безопасность.

Дисциплина «Информатика» является основополагающей для изучения следующих дисциплин: «Экономика», «Основы информационных технологий в техносферной безопасности», «Информационные технологии управления в чрезвычайных ситуациях», «Информационная безопасность», «Надёжность технических систем и техногенный риск», «Принятие решений в кризисных ситуациях», «Геоинформационные системы».

Особенностью дисциплины является необходимость проводить лабораторные работы в аудиториях, оснащенных компьютерами.

В основу преподавания данной учебной дисциплины положена компетентностная модель, в соответствии с которой изучение дисциплины направлено на формирование у обучающихся необходимых компетенций.

Изучение дисциплины Информатика предполагает использование в учебном процессе помимо данного учебно-методического пособия и другие источники. Для дополнительного изучения теоретических аспектов рекомендуются работы [2,3], для овладения практическими навыками расчетов в Excel и программирования на Delphi следует самостоятельно выполнить задания, разо-

бранные в разделе «Примеры решения задач», также полезно выполнить задания из работ [1,3,4], большое количество заданий для самостоятельного обучения программирования представлено в работе [5]. Для контроля усвоения учебного материала следует самостоятельно ответить на контрольные вопросы, которые приведены в заключение каждого раздела.

Актуальность данного учебно-методического пособия и его отличие от других подобных пособий заключается в использовании инновационной компетентностной модели в овладении учебным материалом, а также в том, что акцент в пособии сделан на наиболее востребованные в настоящее время темы: анализ данных при помощи алгоритмических инструментов, решение оптимизационных задач, программирование на объектно-ориентированном языке высокого уровня.

Практическая ценность пособия заключается в его прикладной направленности, а также в том, что материал пособия соответствует с рабочей программой учебной дисциплины Информатика. Вначале в сжатом виде дается теоретический материал, без знания которого никак невозможно выполнение практических заданий, затем следуют контрольные вопросы по теме и практические примеры, которые студенты должны воспроизвести самостоятельно. Дополнительные задания обучающиеся берут из уже имеющихся пособий.

Ограниченный объем учебно-методического пособия не позволил включить в него базовые понятия и основу электронных таблиц. Предполагается, что студент знаком с этими темами в результате обучения по школьной программе.

Учебно-методическое пособие состоит из двух основных разделов: Электронные таблицы Excel и Система объектно-ориентированного программирования Delphi.

В настоящее время электронные таблицы Excel является мощным средством решения задач в самых различных сферах науки и техники, широко применяется для инженерных расчетов. Области применения данной программы не-

обычайно разнообразны – от создания простых расчетных таблиц до графического представления данных в виде диаграмм, выявления закономерностей, заложенных в данных, решения сложных оптимизационных задач. В нашем кратком курсе мы остановимся на наиболее важных возможностях программы в плане ее использования в будущей профессиональной деятельности. В настоящем пособии рассматривается Excel версии 2010.

Система объектно-ориентированного программирования Delphi является одной из самых популярных в настоящее время. Она позволяет создавать приложения любой сложности, которые могут работать не только в операционной системе Windows. Интегрированная среда быстрой визуальной разработки Delphi помимо компилятора включает в себя редактор текста, средства отладки, большие наборы готовых программ, включенные в библиотеки. Окна приложения строятся из готовых компонентов Delphi просто, словно из игрушечных кирпичиков. Для облегчения процесса подготовки начинающего программиста служит данное пособие. В нем рассматривается версия Delphi -7, как наиболее удобная в практическом использовании.

Для практического овладения материалом рекомендуется обратиться к последнему разделу Решение практических задач.

РАЗДЕЛ 1. ЭЛЕКТРОННЫЕ ТАБЛИЦЫ EXCEL

Электронные таблицы Excel позволяют не только накапливать, преобразовывать, обрабатывать информацию, отображать ее в удобной для восприятия графической форме, но и проводить всевозможные виды анализа информации. Цели этого анализа могут быть различны: выявление закономерностей, заложенных в данных, приближенное решение всевозможных задач, в том числе оптимизационных, поддержка принятия решений. Мы познакомимся с некоторыми наиболее важными в плане решения профессиональных задач инструментами анализа.

Анализ данных при помощи линии тренда

Это тенденция изменения чего- либо. Если исследуются некоторые числовые данные, то тренд – тенденция изменения исследуемых данных. Линия тренда – графическое изображение тенденции, она показывает выявленную закономерность. Excel позволяет автоматически выявить линию тренда, что бывает чрезвычайно полезно для решения целого ряда задач. Выделим некоторые из них.

1) **Выявление закономерностей, заложенных в экспериментальных данных.**

Предположим мы решали задачу выявить зависимость расхода бензина от средней скорости движения автомобиля при переезде из пункта А в пункт Б. Была проведена серия экспериментов и получены следующие данные:

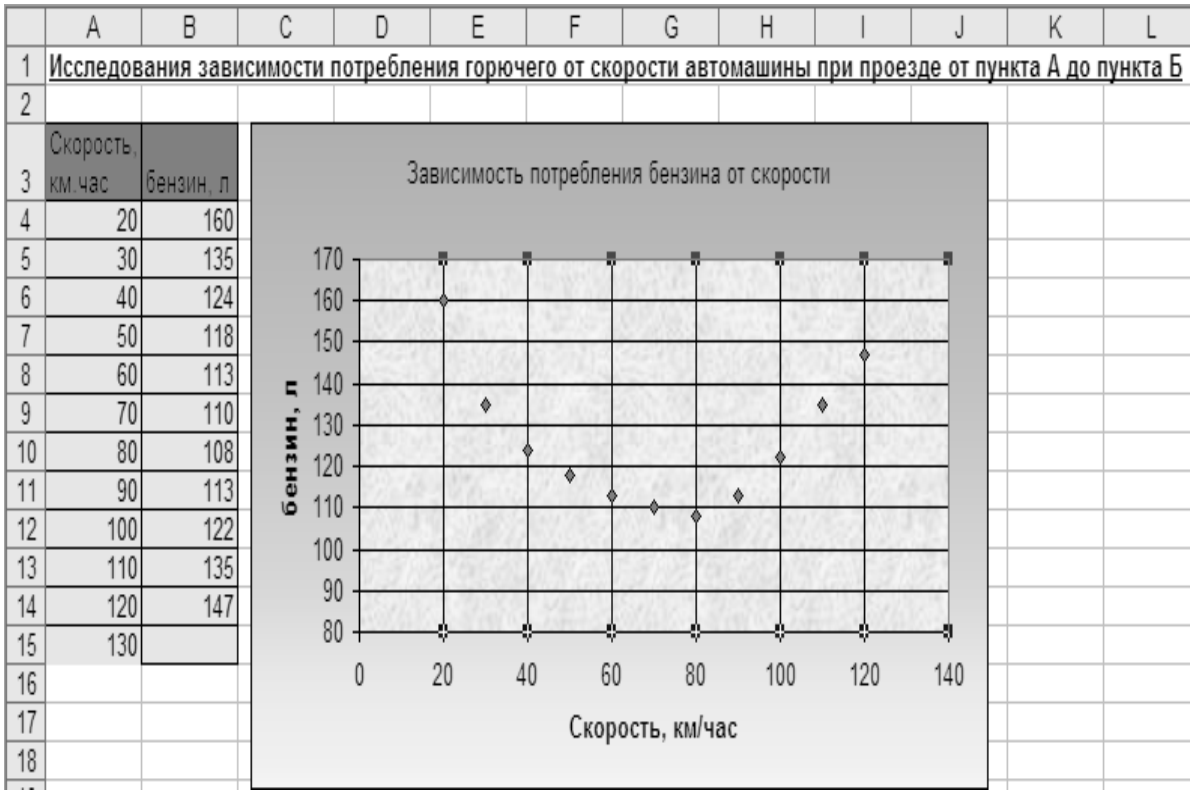


Рис. 1. **Табличное и графическое представление экспериментальных данных**

Excel позволяет не только построить наилучшую линию тренда, но и определить закономерность, которой подчиняются данные (см. рис. 2).

Рис. 2. **Выявление аналитического и графического представления зависимости, заложенной в экспериментальных данных**

2) Решение задачи прогнозирования

Excel позволяет сделать прогноз изменения данных, определить неизвестные будущие или прошлые значения. Для этих целей используется выявленное уравнение закономерности. Например, имея формулу, мы можем вычислить потребление бензина при любых скоростях движения автомобиля. Максимальная скорость движения в эксперименте составляла 120 км/час, но мы можем сделать прогноз потребления бензина при 130 км/час, для этого достаточно правильно записать приведенную на графике формулу в ячейку и подставить в качестве x=130 (см. рис. 3).

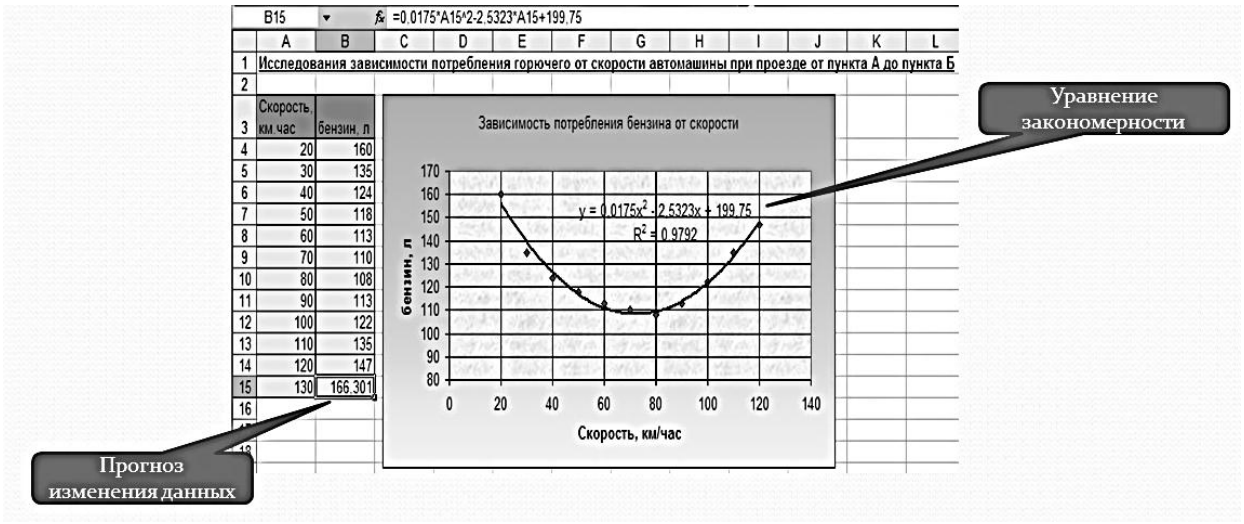


Рис. 3. **Решение задачи прогнозирования**

3) Определения неизвестных коэффициентов в известных закономерностях

Часто встречаются задачи, когда известен общий вид закономерности, но не известны ее коэффициенты. Excel справляется с задачей определения неизвестных коэффициентов в известных уравнениях. В приведенном ниже примере по экспериментальным замерам упругих деформаций образца металла определяется его модуль упругости. При этом заранее известно, что деформации подчиняются закону Гука, который выражается простой линейной зависимостью:

$$H = E \ L,$$

где H – напряжение, E – модуль упругости, L – деформация.

Пусть в лабораторию по измерению упругих деформаций поступил брусок из неизвестного металла. Задача заключается в том, чтобы определить материал, из которого сделан брусок. В результате экспериментов на соответствующем приборе были получены данные, приведенные в таблице, по ним была построена точечная диаграмма, построена линия тренда, получено уравнение зависимости (см. рис. 4).

Деформация, мм	Напряжение, Гпа
0	0
0,0015	0,15
0,003	0,35
0,0045	0,49
0,006	0,68

Ответ: Материал - титан.

Рис. 4. **Определение коэффициента линейной закономерности**

Если сравнить полученное на диаграмме уравнение с законом Гука, то ясно, что y – это напряжение, x – деформация, а коэффициент 113,33 – модуль упругости. Таким образом, неизвестный коэффициент определен, по справочнику определяем, что материал, из которого сделан образец – титан.

Оценка качества линии тренда

Excel позволяет прочерчивать линии тренда различных типов, но задача заключается в том, чтобы выбрать наилучшую из многих, именно о ней можно сказать, что данная линия тренда и ее уравнение отражают закономерность. Как же определить, какая линия тренда лучше, а какая хуже? Для этих целей вводится **величина достоверности аппроксимации R-квадрат.**

$$\mathbf{0 < R^2 <= 1}$$

Величина R^2 представляет собой квадрат коэффициента корреляции Пирсона. Эта величина изменяется от 0 до 1, но не равняется нулю никогда. Чем ближе она к 1, тем выше качество аппроксимации. Если $R^2 = 1$, то достиг-

нуто полное совпадение экспериментальных и расчетных данных, если данных статистически достаточно, то можно считать, что закономерность выявлена, то же самое можно сказать, если R^2 близка к 1, например, 0,9. Если R^2 в районе 0,5 или 0,6, то, скорее всего, закономерность не наблюдается.

Процесс построения линии тренда

Перейдём теперь к процессу построения линии тренда. Его можно разбить на несколько этапов:

1) Вначале следует построить точечную диаграмму (вариант без прочерчивании линии) по таблице с опытными данными (см. рис. 1).

2) Выделим диаграмму, щелкнув по ней, а затем нажмем кнопку **Линия тренда** в группе **Анализ** на вкладке **Макет**, в открывшемся меню выберем пункт **Дополнительные параметры линии тренда** (см. рис. 5).

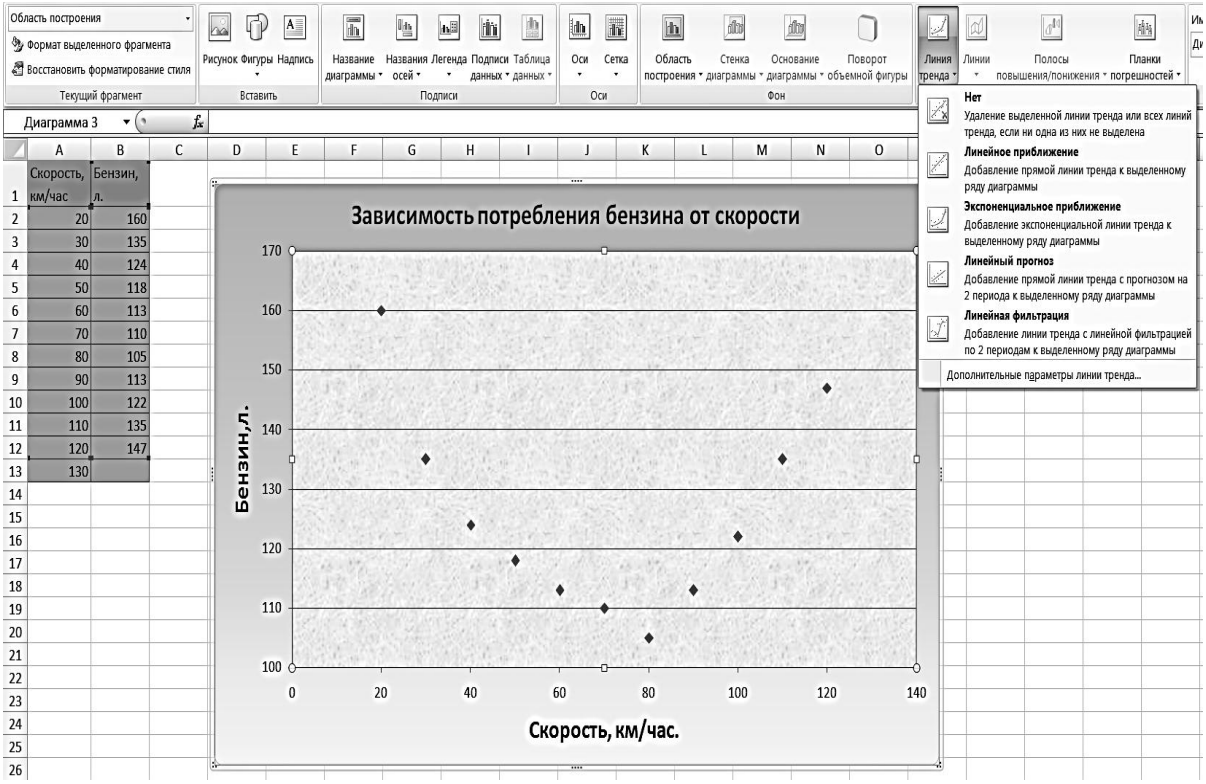


Рис. 5. **Начальный этап построения линии тренда**

3) В появившемся диалоговом окне следует выбрать один из 6 типов аппроксимации (см. рис. 6). На самом деле типов аппроксимации больше, т.к. при выборе полиномиальной линии тренда можно определить степень полинома от

2 до 6. Выбор типа линии облегчает небольшая картинка слева, на которой примерно показан вид данной линии.

4) Для того, чтобы увидеть на диаграмме аналитическое выражение линии тренда и ее оценку, следует поставить галочки напротив «показывать уравнение на диаграмме» и «поместить величину достоверности аппроксимации».

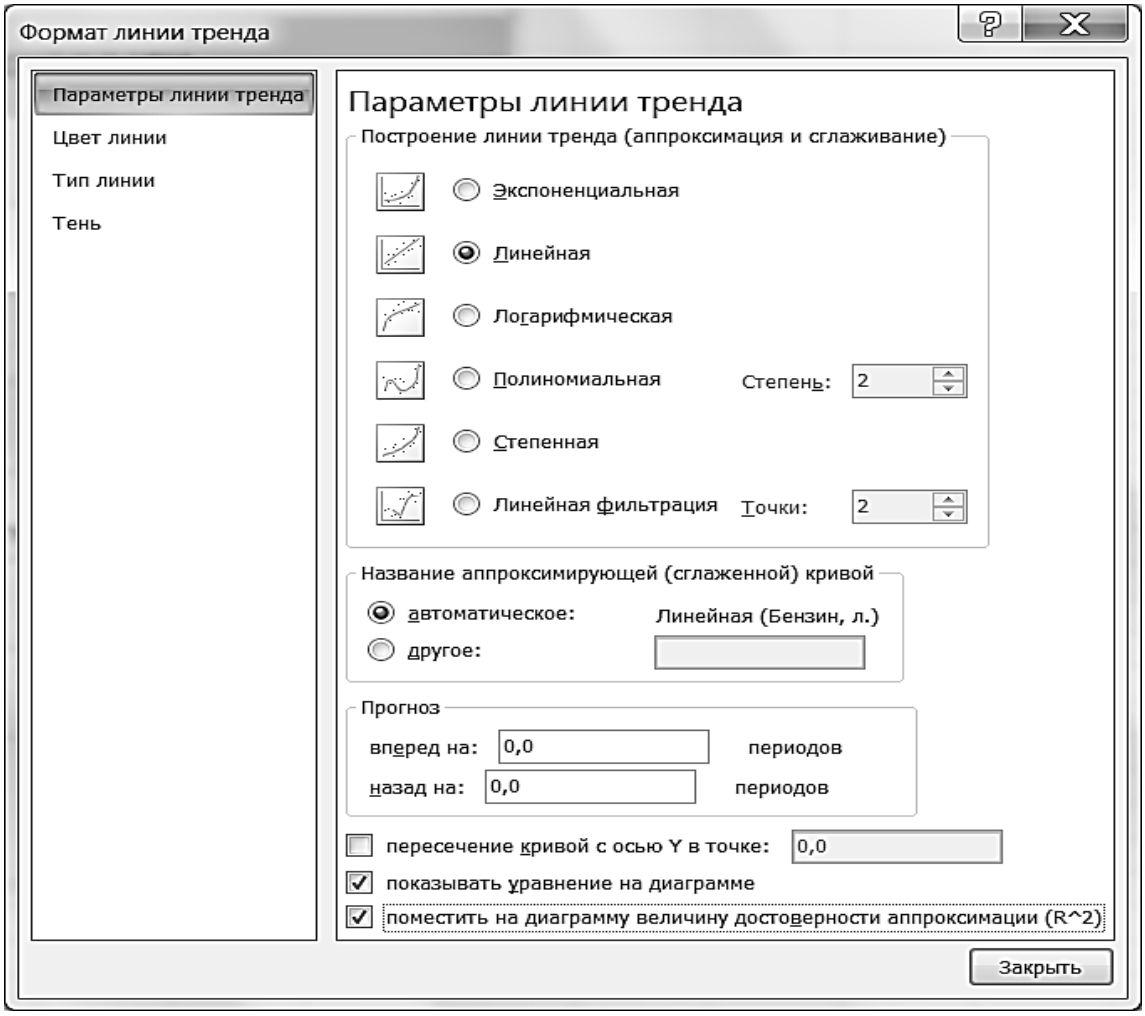


Рис. 6. **Выбор типа линии тренда**

5) После того, как линия тренда построена, следует записать величину достоверности аппроксимации $\mathbf{R^2}$. Поскольку в общем случае перебираются несколько типов зависимостей, результаты удобно записать в таблицу (см. табл. 1).

6) Удалите линию тренда: нажмите снова кнопку **Линия тренда** и укажите **НЕТ**. Постройте новую линию тренда, повторив шаги 2…6 и изменив тип зависимости. Результаты запишите в таблицу 1.

Таблица 1

Определение лучшей линии тренда

Вид зависимости	R^2
линейная	0,9330
логарифмическая	0,7900
экспоненциальная	0,8500
полиномиальная	0,9792
степенная	0,7110

Искомая линия тренда должна иметь максимальную величину $\mathbf{R^2}$. Построим ее снова, результаты показаны на рис. 7.

Рис.7**. Оптимальная линия тренда и ее аналитическое представление**

Решение задачи прогноза

После того, как определена зависимость, которой подчиняются экспериментальные данные, легко получить приблизительные значения функции, в точках, в которых отсутствуют наблюдения. Например, получим расход топлива при скорости 130 км/час. Запишем полученную формулу и подставим нужное значение аргумента (см. рис. 8).

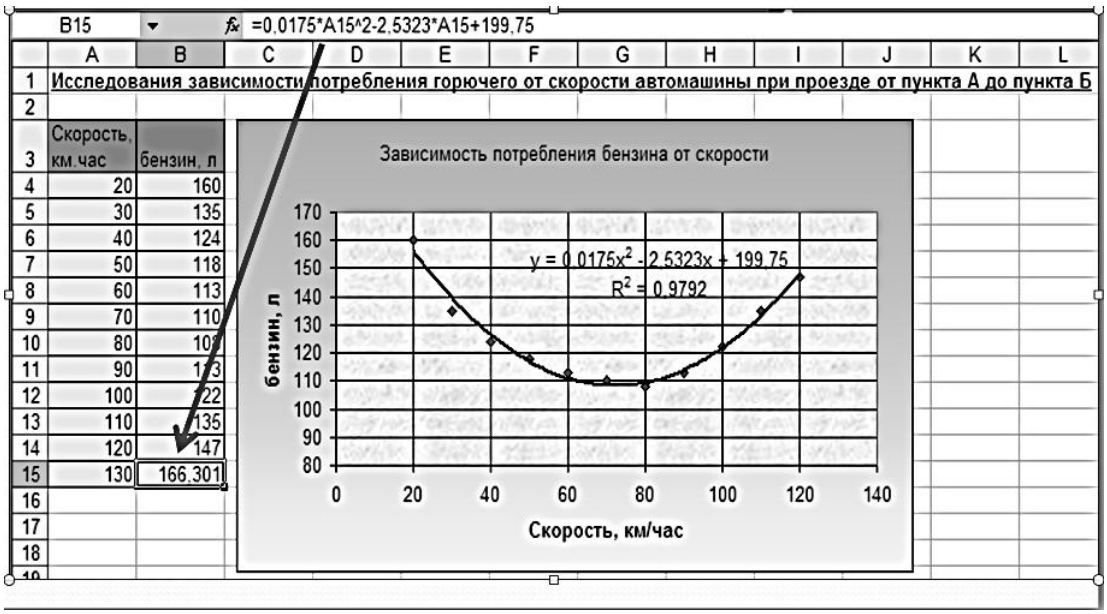


Рис. 8. **Решение задачи прогноза**

Поддерживаемые линии тренда

Для того, чтобы грамотно подходить к определению линии тренда в различных задачах, следует знать их уравнения. Excel поддерживает следующие типы аппроксимации:

- Линейная,
- Логарифмическая,
- Экспоненциальная,
- Полиномиальная,
- Степенная,
- Скользящее среднее.

Неизвестные коэффициенты зависимостей определяются методом наименьших квадратов.

Средство Поиск решения

Excel обладает универсальным средством решения уравнений и оптимизационных задач, которые включают в себя нелинейные уравнения практически

любой сложности. Оно называется Поиск решения и является надстройкой Excel. Ниже перечисляются лишь некоторые характерные задачи, которые можно решить с помощью этого средства:

- Оптимизация прибыли предприятия при ограничениях на ресурсы.
- Максимизация производства сельскохозяйственной продукции при ограничениях на различные ресурсы, в том числе на оросительную воду.
- Составление штатного расписания для достижения наилучших результатов при наименьших расходах.
- Планирование перевозок. Минимизация затрат на транспортировку товаров.

Охарактеризуем кратко те задачи, которые можно решать с помощью данного инструмента:

1. **Имеется единственная цель**, например, максимизация прибыли или минимизация расходов.
2. **Имеется набор входных значений-переменных**, непосредственно или косвенно **влияющих на оптимизируемую величину.** Эти переменные связанные уравнениями, которые составляют математическую модель некоторого объекта.
3. **Имеются ограничения**, выражающиеся, как правило, в виде неравенств. Например, объем используемого сырья не может превышать объем сырья, имеющегося на складе

Поясним, как можно записывать ограничения в Excel, при использовании **Поиска решения.** Под ограничениями понимаются соотношения типа:

A1<=B1, A1=A2, A1>=0.

По меньшей мере, одна из ячеек в соотношении, задающем ограничение, должна зависеть от переменных задачи, в противном случае это ограничение не влияет на процесс решения.

Часто ограничения записываются сразу для групп ячеек, например:

A1:A10<=B1:B10 или A1:E1>=0.

Типы математических моделей

При решении задач с помощью надстройки Поиск решения полезно различать линейные и нелинейные модели. Под линейными понимаются модели, в которых целевая функция представлена линейным выражением, а также используются линейные уравнения и линейные ограничения.

Общий вид линейной функции:

$X=A*Y_1+B*Y_2+C*Y_3+ ...$

В этом выражении: A, B и C — константы; Y_1, Y_2, Y_3 — переменные; X — результирующее значение. В этом случае можно применять более быстрые и надежные линейные методы поиска решения.

Порядок работы с надстройкой **Поиск решения** разберем на примере решения простенькой оптимизационной задачи:

Пусть задана математическая модель объекта, включающая в себя два нелинейных уравнения:

1) $y = 2a+\sin(b)$,

$x = a + b^{2.}$

2) Целевая функция задана выражением:

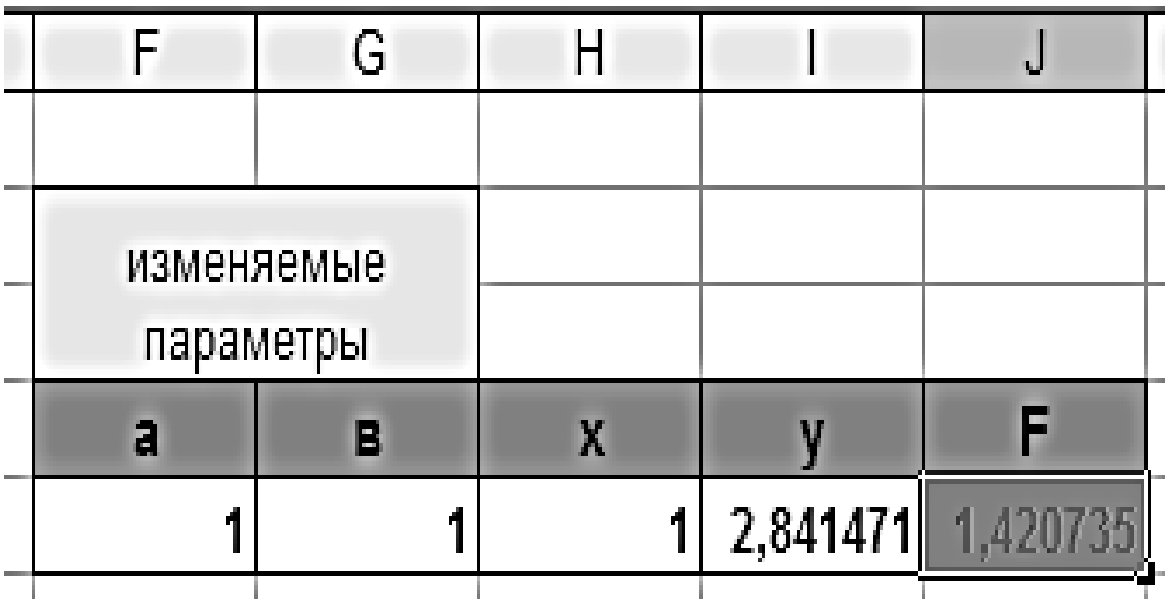
$F = y/(1+x^2)$.

3) На параметры **a** и **b** наложены ограничения:

$0 \le a \le 2, \quad -1 \le b \le 2$

Требуется найти максимум целевой функции.

A) Создадим на рабочем листе расчетную таблицу, в которой целевая функция рассчитывается исходя из приведенных выше уравнений (см. рис. 9):



	F	G	H	I	J
	изменяемые параметры				
	a	в	x	y	F
	1	1	1	2,841471	1,420735

Рис. 9. **Расчетная таблица для решения оптимизационной задачи**

Б) Выделим целевую ячейку и дадим команду: **Данные, Анализ, Поиск Решения.**

В) В открывшемся окне **Поиска решения** (см. рис. 10) укажем, что ищем максимальное значение целевой функции, заполним поле **«Изменяя ячейки»** ссылками на ячейки, в которых находятся значения параметров **a** и **b**, для чего проведем по ним мышкой. Если бы задача была линейная, то следовало бы указать в раскрывающемся списке **Выберите метод решения: Поиск решения линейных задач симплекс-методом**

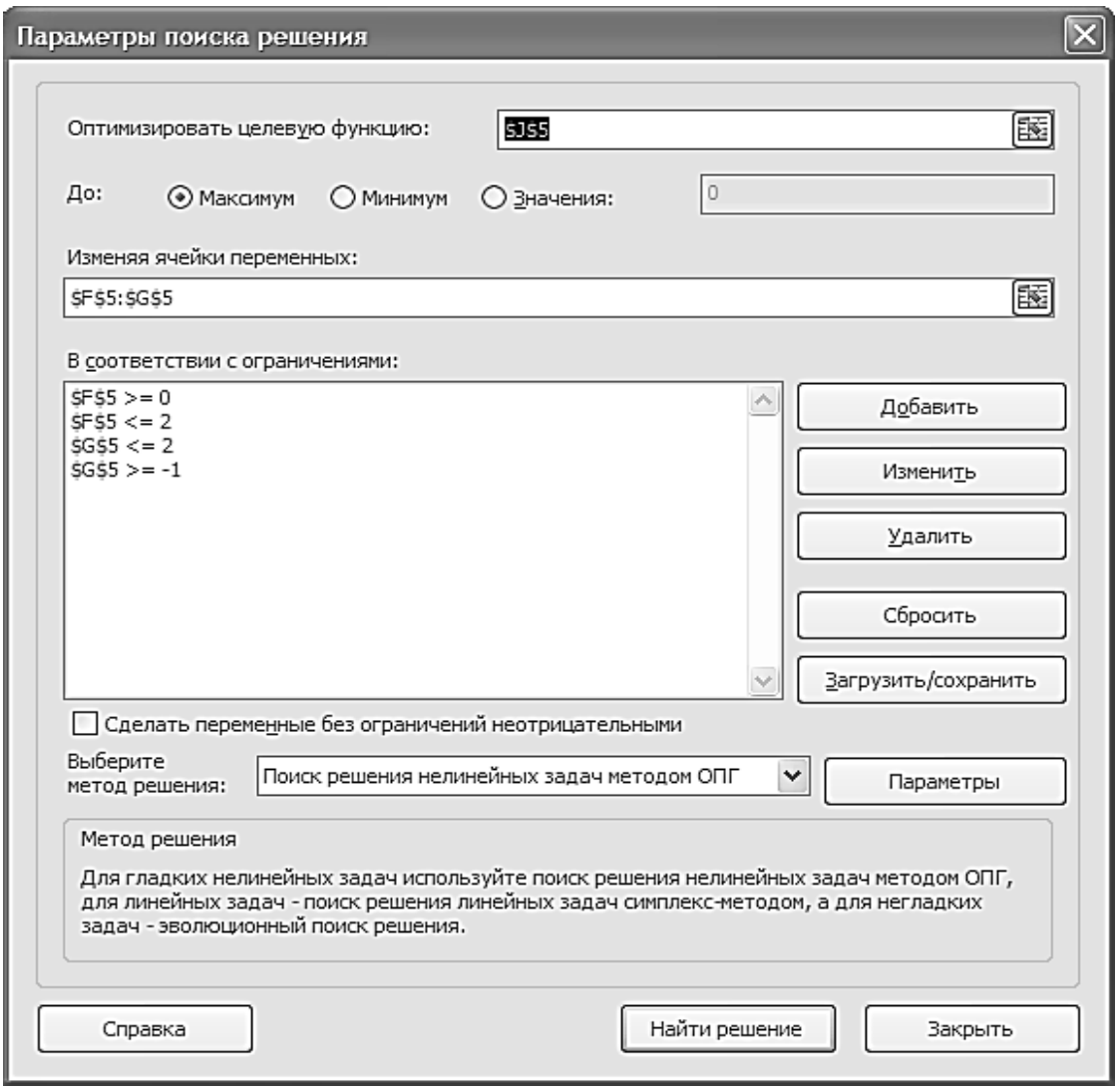


Рис. 10. **Диалоговое окно надстройки Поиск решения**

Г) Введем ограничения, которые наложены на параметры **a** и **b**, нажав кнопку **Добавить**. В открывшемся окне (см. рис. 11) одно за другим введем все ограничения. Если необходимо изменить ограничение, жмем кнопку **Изменить**, для удаления ограничения используется кнопка **Удалить.**

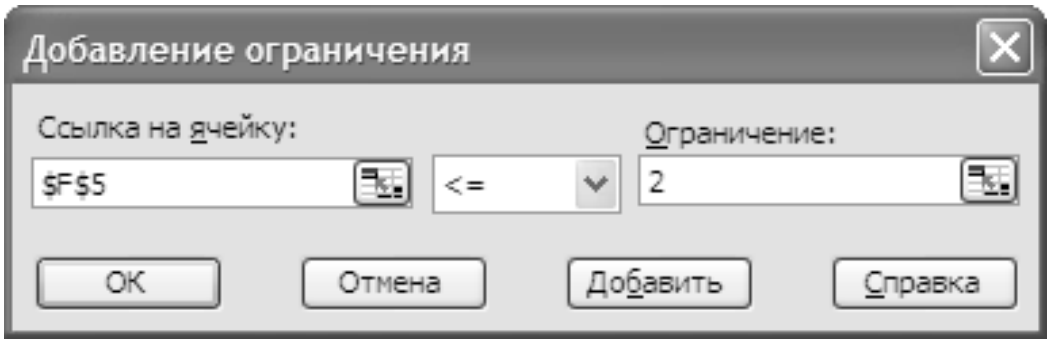


Рис. 11. **Окно ввода ограничений**

В некоторых задачах требуется, чтобы в расчетах участвовали целочисленные значения. В этом случае можно использовать условие целочисленности (см. рис. 12). Еще одно возможное ограничение — это двоичная переменная. При использовании данного ограничения число в ячейке может принимать только значение ноль или единица.

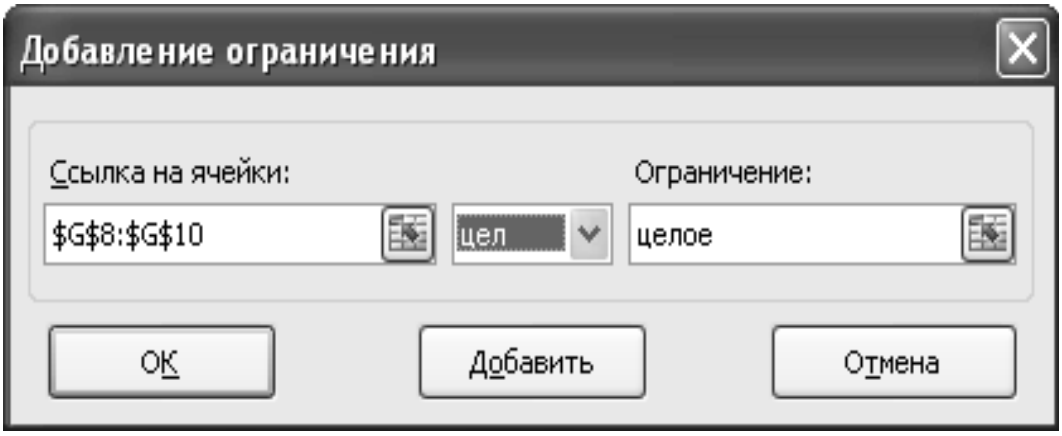


Рис. 12. **Добавление условия целочисленности**

Д) После того, как заполнены все необходимые поля, нажмите кнопку **Найти решение.** По окончании поиска решения откроется диалоговое окно Результаты поиска решения (см. рис. 13)**:**

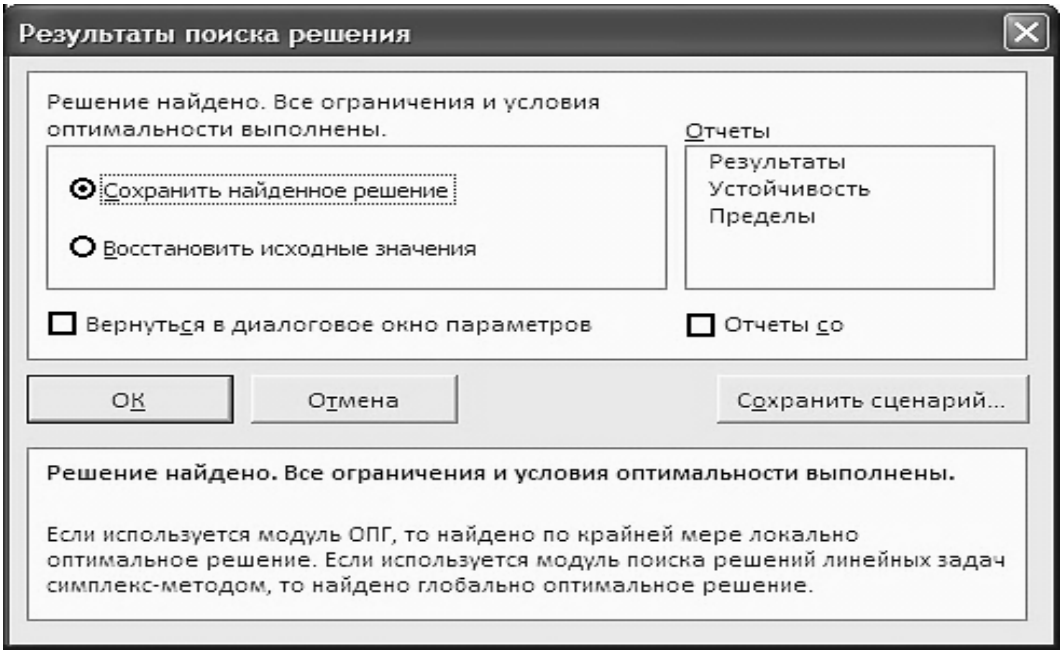
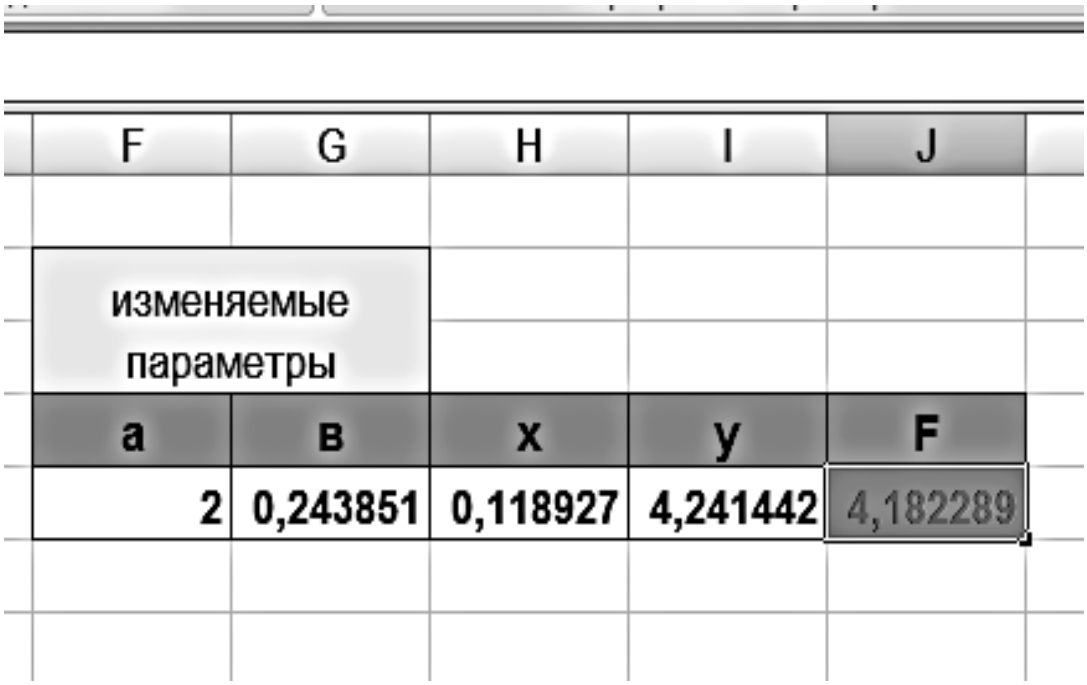


Рис. 13. **Отчет о результатах поиска решения**

Результаты решения задачи можно увидеть на рабочем листе (см. рис. 14). Выберите переключатель **Сохранить найденное решение**, чтобы сохра-

нить найденные значения, или переключатель **Восстановить исходные значения**, чтобы оставить начальные значения, которые были на рабочем листе. Нажмите кнопку **OK**.



	F	G	H	I	J
	изменяемые параметры				
	a	в	x	y	F
	2	0,243851	0,118927	4,241442	4,182289

Рис. 14. **Результаты решения оптимизационной задачи**

Изменение параметров работы

Можно изменить параметры работы при поиске решения, например, поменять метод поиска ответа, ограничить время поиска, задать другую точность вычислений. При нажатии в диалоговом окне **Параметры поиска решения (**см. рис. 10**)** кнопки **Параметры** открывается диалоговое окно **Параметры.**

Что делать, если оптимальное решение не найдено?

Это может быть связано со следующими причинами:

- начальные значения переменных выходят за пределы ограничений;
- количество итераций или время поиска решения превысило максимально допустимое;
- при решении нелинейной задачи в диалоговом окне Параметры выбран неподходящий метод решения;
- значение целевой ячейки неограниченно возрастало или убывало;

- при использовании условия целочисленности задано слишком маленькое допустимое отклонение (параметр Целочисленная оптимальность в диалоговом окне Параметры);
- модель включает переменные, значения которых отличаются друг от друга на несколько порядков, и при этом флажок **Использовать автоматическое масштабирование** в диалоговом окне **Параметры** не установлен.

Следует изменить параметры и снова запустить **Поиск решения**.

Вопросы для самоконтроля по разделу 1

1. Что такое тренд? Что такое линия тренда?
2. Какие основные задачи можно решать при помощи инструмента «Линия тренда»?
3. Как оценить качество линии тренда?
4. Назовите основные этапы построения линии тренда.
5. Как определить наилучшую из возможных линию тренда?
6. Как при помощи инструмента «Линия тренда» решить задачу прогноза?
7. Какие типы линии тренда поддерживает Excel?
8. Для решения каких задач следует использовать средство «Поиск решения»?
9. Как установить надстройку «Поиск решения»?
10. Охарактеризуйте специфику задач, которые можно решать с помощью средства «Поиск решения».
11. Чем отличаются линейные и нелинейные математические модели?
12. Приведите пример простой оптимизационной задачи, для решения которой может быть использовано «Поиск решения».
13. Опишите порядок действий при использовании средства «Поиск решения».

14. Как вводятся и корректируются ограничения в диалоговом окне «Параметры поиска решения»?
15. Как убедиться, что оптимизационная задача решена при помощи средства «Поиск решения»?
16. С чем связана невозможность решения задачи при использовании средства «Поиск решения» и что необходимо сделать, чтобы попытаться решить задачу?

РАЗДЕЛ 2. СИСТЕМА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ DELPHI

Основы программирования в Delphi

Интерфейс Delphi

Вид экрана после запуска Delphi несколько необычен. Вместо одного окна на экране появятся пять (см. рис. 15) . Окна можно перемещать независимо друг от друга, так как это удобно пользователю.

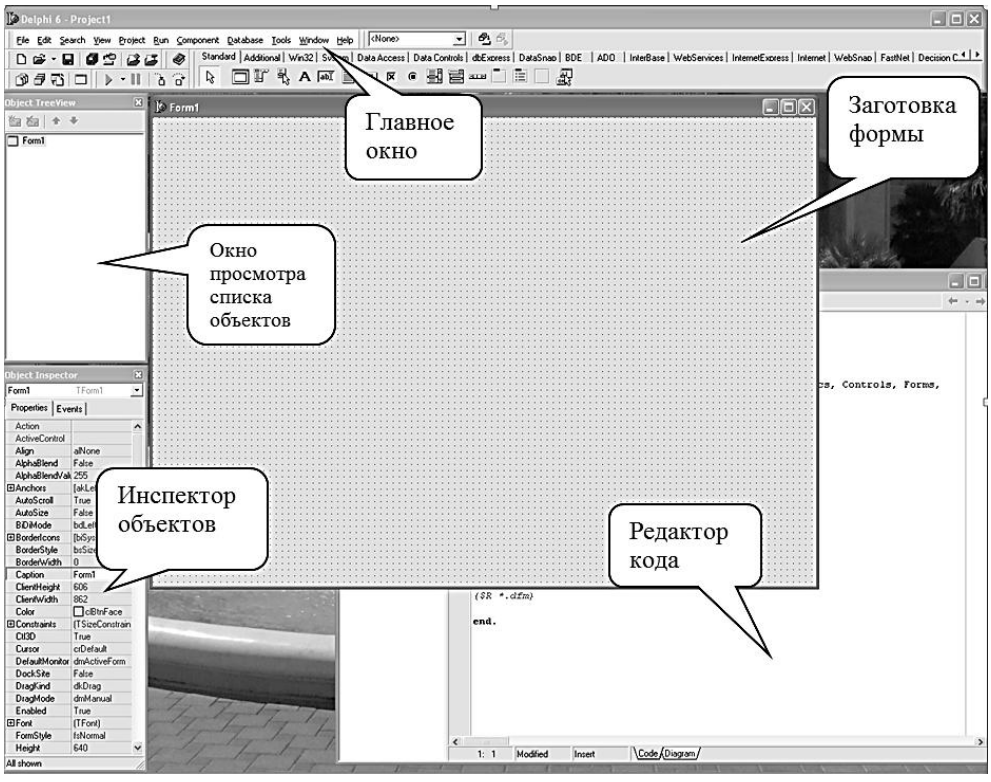


Рис. 15. **Интерфейс Delphi**

Главное окно

В главном окне находится главное меню, панели инструментов и палитра компонентов. Основные панели инструментов – **стандартная** и **вид**, на первой дублируются важнейшие пункты меню, на второй присутствуют кнопки, предназначенные для удобного перехода по формам, модулям, окнам проекта. На панели (палитре) компонентов присутствует несколько вкладок. Наиболее часто используемые компоненты находятся на вкладке **Standard** (см. рис. 16) .

Рис. 16. **Главное окно**

Инспектор объектов

Инспектор объектов – это настройщик объектов, предназначенный для редактирования значений свойств объектов, создания процедур обработки событий. Имеет две вкладки: **Properties** и **Events** (свойства и события). В верхней его части имеется раскрывающийся список всех имеющихся на форме компонентов. После выбора конкретного компонента можно перейти к редактированию его свойств (см. рис. 17).

Рис. 17**. Окно инспектора объектов**

Заготовка формы

Представляет собой заготовку главного окна разрабатываемого приложения. Это основной компонент Delphi (см. рис. 15).

Редактора кода

Предназначен для набора текста программы. В начале работы над новым проектом содержит модуль Delphi с описанием пустого окна.

Окно просмотра списка объектов

В данном окне отображаются в виде иерархического списка все компоненты и объекты проекта Delphi. Данное окно обычно используется при создании больших программ, имеющих несколько окон диалога с многочисленными элементами управления и многостраничный программный код.

Основополагающие понятия

Характеристики объектов Delphi

Объекты – это абстракции, с которыми оперируют в объектно-ориентированных языках программирования. Объекты обладают собственными признаками, отличающими их от других объектов. Объекты - это диалоговые окна, различные меню, управляющие кнопки, поля ввода-вывода, графики, программные модули. Объект – всегда представитель некоторого класса объектов. Стандартный объект Delphi, созданный при помощи палитры компонентов, также называют компонентом.

Объекты тремя главными характеристиками: свойствами, методами, событиями (см. рис. 18).

Рис. 18. **Характеристики объекта Delphi**

Свойства

Свойства - это характеристики, определяющие состояние, вид, положение, поведение объекта. Свойство объекта должно принимать определенное значение.

Значения свойств могут быть представлять собой:

1. Текстовое поле, которое можно заменять и редактировать.
2. Логическое значение True, False.
3. Константу. В случаях 2 и 3 значение свойства задается из раскрывающегося меню.

4. Числовое значение (обычно выражено в пикселях, если оно связано с размерами объектов).

В некоторых случаях для задания значения свойства используется специальный редактор, который появляется в поле значения свойства в виде пиктограммы с многоточием. Вызов редактора осуществляется щелчком мыши по пиктограмме.

Некоторые свойства бывают сложными, т.е. представляют собой список свойств более низкого иерархического уровня. В этом случае напротив такого свойства стоит значок «+». Доступ к подсвойствам осуществляется щелчком по значку. Примером такого свойства является Font (шрифт).

Методы

Методы – это программы действий над объектом и его свойствами и полями. Они представляют собой подпрограммы, связанные с объектом. Совокупность методов объекта определяет его поведение. Примеры методов объекта Form – Close, Show, Print (закрыть, показать, распечатать диалоговое окно).

События

Понятия событий играет важную роль в визуальном программировании. Программа в Delphi не делает ничего иного, как реагирует на происходящие во внешней среде события. Событие – это изменение состояния компьютера и его периферических устройств. Щелчок на изображении командной кнопки, нажатие клавиш клавиатуры, перемещение мыши и др. – это примеры того, что в Delphi называют событием. В Delphi каждому событию присвоено имя. Например, щелчок кнопкой мыши – это событие OnClick, двойной щелчок кнопкой мыши – это событие OnDbClick.

Работа по созданию приложения

Проектирование форм

При создании новой формы (окна) в первую очередь следует задать заголовок (изменить значение свойства Caption), скорректировать размеры и положение формы. Размеры и положение можно задать, перемещая форму при по-

мощи мыши, как обычное окно Windows. Затем на форму наносятся необходимые компоненты. Их выбирают в палитре компонентов, щелкнув мышью по соответствующей пиктограмме. Далее следует установить курсов в ту точку формы, в которой должен быть левый верхний угол компонента, и еще раз щелкнуть мышью. В результате на форме появится компонент стандартного размера (см. рис. 19).

Размер компонента можно задать в процессе его добавления к форме. Для этого надо при нанесении компонента удерживать нажатой левую кнопку мыши и переместить курсор в ту точку, где должен находиться правый нижний угол компонента. На форме появится компонент нужного размера. Каждому компоненту Delphi присваивает стандартное имя, которое состоит из названия компонента и его порядкового номера. Например, если к форме добавить два компонента Edit, то их имена будут Edit1, Edit2.

Для изменения положения компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку и, удерживая ее нажатой, переместить контур компонента в нужную точку формы, затем отпустить кнопку мыши.

Для того чтобы изменить размер компонента, необходимо его выделить, установить указатель мыши на один из маркеров и, прижав его, переместить мышью в нужное место.

Для удаления компонента следует выделить его и нажать клавишу Delete.

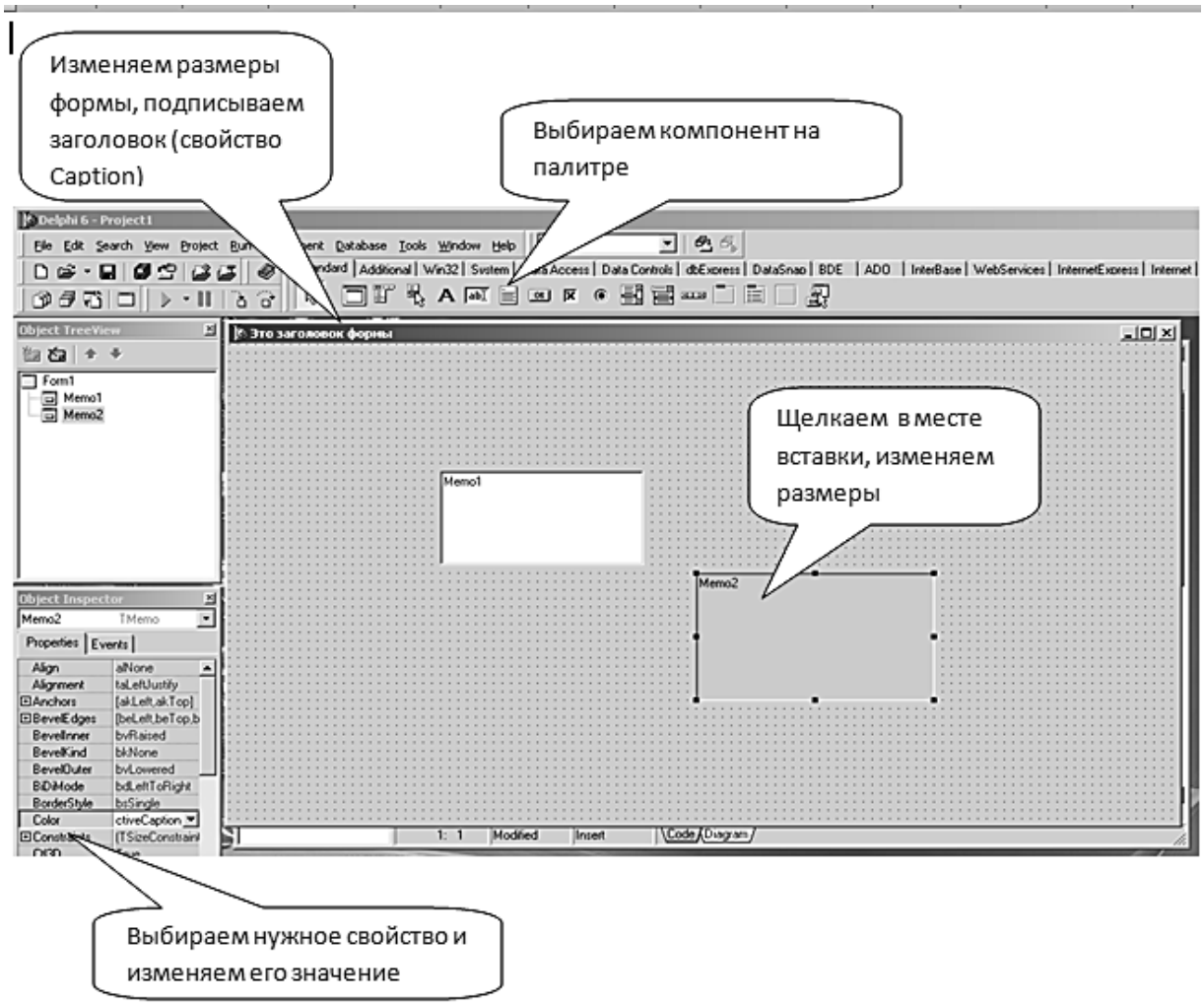


Рис. 19. **Работа над окном приложения**

Свойства компонента так же, как и свойства формы, можно изменять при помощи Инспектора Объектов. Для этого нужно выделить нужный компонент (щелкнуть мышью на его изображении), затем перейти на вкладку **Properties Инспектора объектов**, выбрать в списке нужное свойство и произвести необходимое изменение в поле значения свойства (правое поле). Ниже приводится таблица 2, в которой перечислены важнейшие свойства главного компонента Delphi – Form(окна) и два метода: закрыть и показать.

Таблица 2

Основные свойства и методы компонента Form

Свойства/методы	**Описание**
Name	Имя формы. **Изменять не рекомендуется!**
Autosize	Автоподстройка формы под компоненты (значение True). Форма автоматически сжимается, пустые места убираются.
Caption	Текст, размещенный в заголовке окна.
Width	Ширина (в пикселях).
Height	Высота (в пикселях).
Top	Расстояние от верхней границы экрана (в пикселях).
Left	Расстояние от левой границы экрана (в пикселях).
Color	Цвет
Font	Шрифт, для компонента, расположенного на форме.
Close (метод)	Закрывает окно. Нужно написать, например,Form1.Close;
Show(метод)	Выводит на экран окно формы. Нужно написать Form1.Show;

Написание текста программы, пуск на счет

После того, как создано окно программы, необходимо выполнить следующие действия:

1) **написать программный код**, чтобы программа могла выполнить соответствующие действия. Обычно окно программы содержит командные кнопки, после нажатия на которые выполняются действия. Для автоматического создания процедуры отработки события – нажатия на командную кнопку – необходимо два раза щелкнуть мышью по этой кнопке. Тут же появится окно ре-

дактора кода с готовым шаблоном процедуры: ее заголовок и ключевые слова Begin, End уже написаны:

```
procedure TForm1.Button1Click(Sender: TObject);
begin


end;
```

Курсор мигает между ними, пользователю лишь остается вписать необходимый программный текст.

2) **Сохранить проект.** Вся ваша работа: тексты программ, проекты окон приложения и установки системы программирования, называется проектом. Проект представляет собой набор файлов, хранящихся на диске.

После завершения определенного этапа работы над проектом его следует сохранять. Прежде необходимо создать отдельный каталог с именем разрабатываемой программы или именем близким по смыслу. Для первоначального сохранения следует дать команду **File, Save Project As**. Далее следует в открывшемся диалоговом окне выбрать папку для сохранения. Сохранение идет в два этапа.

- Вначале сохраняются тексты программ. Для простой программы сохраняется один программный модуль.
- Затем сохраняется файл проекта.

Имена файлов лучше не менять.

При последующих сохранениях проекта следует давать команду **File, Save All** или нажать на кнопку  на панели инструментов. Никакого диалога больше возникать не будет.

3) З**апустить программу на компиляцию и исполнение.** Запуск программы можно осуществить, задав команду **Run, Run**, или щелкнув на соответ-

ствующей кнопке панели инструментов (зеленый треугольник: ) или нажав клавишу **F9**.

Если ошибок в программе нет, то на экране появится окно работающей программы. Поля ввода готовы для ввода информации, но прежде чем вводить числа надо помнить, что при вводе дробная часть от целой отделяется запятой, в отличие от ввода дробной константы в тексте программы. Там дробная часть отделяется от целой точкой.

4) **Если программа зависла или выдала непонятную ошибку в момент выполнения**, следует закрыть работающую программу командой: **Run, Program Reset**. Затем можно перейти в редактор кода для выяснения ошибки. Возможные причины зависания: неверные данные или отсутствие данных, деление на нуль, слишком большое число – переполнение.

После исправления ошибок следует снова пустить программу на выполнение.

Компоненты

Компоненты – это те кирпичики, из которых складываются окна приложений Delphi. Без знания достаточного набора компонентов их свойств и методов эффективная работа невозможна. Приведенная ниже таблица (см. табл. 3) поможет получить общую информацию о компоненте, на какой вкладке Палитры компонентов он и для чего используется.

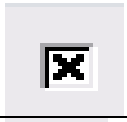
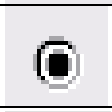
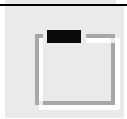
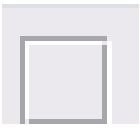
Таблица 3

Важнейшие компоненты Delphi

Вкладка	Компонент	Пиктограмма	Для чего используется
1	2	3	4
	Form		Заготовка формы
Standart	**MainManu**		Создает главное меню формы
	Label		Текстовое поле для вывода информации

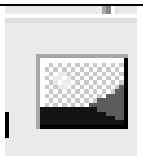
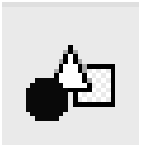
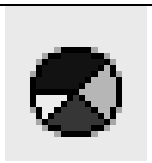
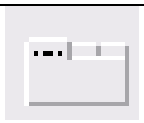
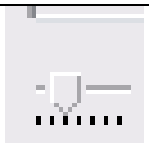
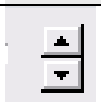
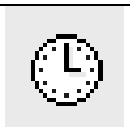
Продолжение табл. 3

Важнейшие компоненты Delphi

1	2	3	4
	Edit		Однострочное поле ввода – вывода
	Memo		Многострочное поле ввода-вывода
	Button		Командная кнопка
	Checkbox		Независимый переключатель - флажок
	RadioButton		Зависимый переключатель - флажок
	ListBox		Многострочное поле ввода - вывода с дополнительными свойствами
	ComboBox		Поле с раскрывающимся списком для выбора
	GroupBox		Группа элементов - компонент, используемый для группировки других компонентов под общим заголовком
	RadioGroup		Группа зависимых переключателей - флажков
	Panel		Панель - компонент, используемый для группировки других компонентов в общей рамке
Additional	**BinBtn**		Командная копка с дополнительными возможностями
	StringGrid		Таблица для отображения текстовой информации

Окончание табл. 3

Важнейшие компоненты Delphi

1	2	3	4
	Image		Изображение - компонент, используемый для вывода готовых изображений
	Shape		Фигура – компонент, используемый для рисования на форме простых геометрических фигур
	Chart		Диаграмма – компонент, используемый для создания графиков и диаграмм
Win32	**PageControl**		Набор вкладок – компонент, позволяющий располагать на вкладках элементы управления
	TrackBar		Ползунок – компонент, используемый для ввода целых чисел из заданного диапазона
	ProgressBar		Индикатор выполнения операций
	UpDown		Счетчик - – компонент, используемый для ввода целых чисел из заданного диапазона
System	**Timer**		Таймер – компонент, отсчитывающий фиксированные промежутки времени, и позволяющий создать реакцию на их завершение

Вначале рассмотрим наиболее часто используемые в прикладных программах компоненты.

Текстовое поле для вывода информации

Обычно окна программ содержат пояснительный текст, заголовки рядом с полями вывода и таблицами. Для этих целей эффективно используется компонент **Label.**

Пояснительный текст или выводимые результаты помещаются в свойство **Caption**. Шрифт может быть изменен по желанию пользователя, модифицируя значение свойства **Font**. Свойство **Color** определяет цвет фона, на котором будет выведен текст. Следует обратить внимание на свойства **Autosize** и **WordWrap**. Эти свойства нужно использовать, если поле вывода должно содержать несколько строк текста. После добавления к форме компонента **Label** значение свойства **Autosize** равно **true**, т.е. размер поля определяется автоматически в процессе изменения значения свойства **Caption**. Если вы хотите, чтобы находящийся в поле вывода текст занимал несколько строк, то надо сразу после добавления к форме компонента **Label** присвоить свойству **Autosize** значение **False**, свойству **WordWrap** – значение **True**. Затем при помощи мыши придать следует полю требуемый размер. Только после этого можно ввести в свойство **Caption** текст, который требуется вывести (см. рис. 20).

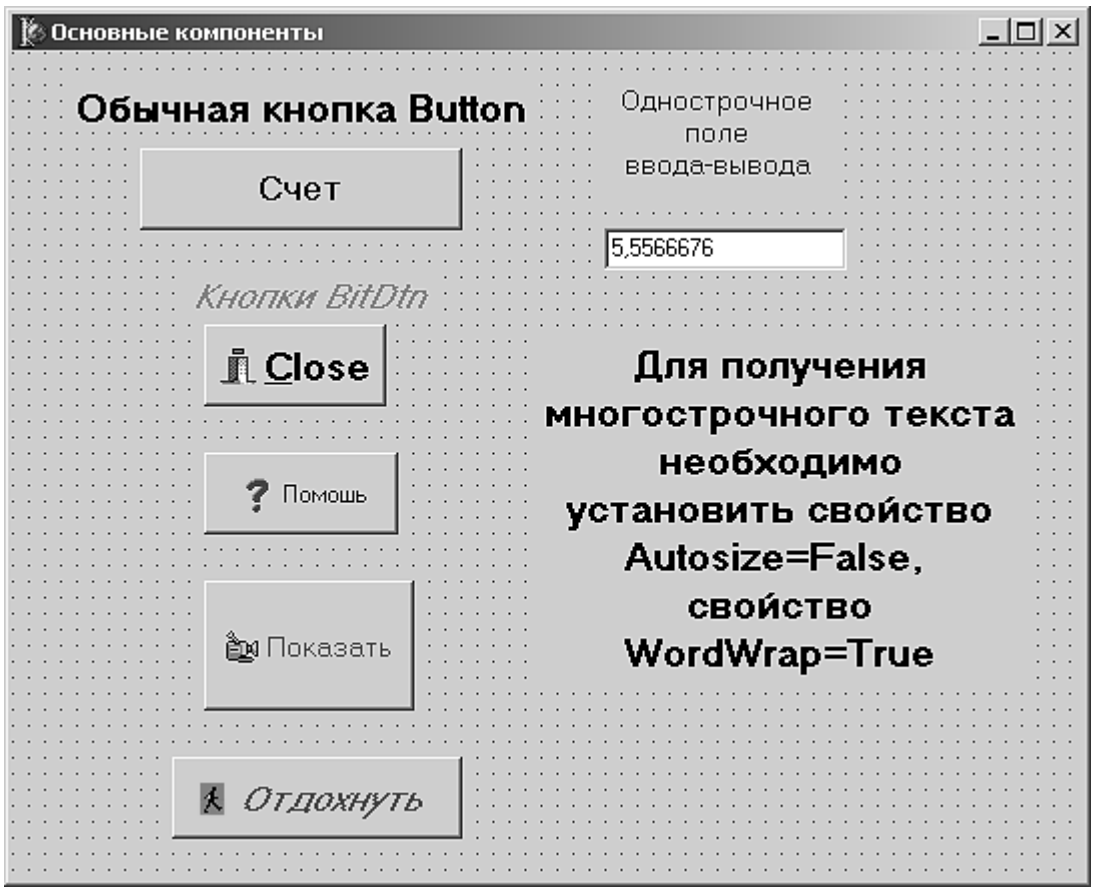


Рис. 20. **Простейшие компоненты на форме**

Однострочное поле ввода – вывода

Без ввода-вывода каких-либо данных не обходится почти не одна программа. Для целей ввода-вывода строк символов служит компонент **Edit** (см. рис. 20). Для ввода-вывода чисел необходимо использовать функции преобразования типа данных, поскольку данные отображаются в форме текста. Например, для ввода дробного числа из компонента Edit2 в переменную x следует использовать следующий оператор:

X:=StrToFloat(Edit2.text);

Обычно поле, размещаемое на заготовке формы, делают первоначально пустым. Для этого следует стереть значение свойства **Text**. Для изменения шрифта, появляющегося в поле текста, следует воспользоваться свойством **Font.** Свойство **Color** определяет цвет фона, на котором будет выведен текст.

Командная кнопка

Нажатие на командные кнопки активизирует программы, заставляет ее выполнять те или иные действия, например, расчет, создание графика, выход, и

т.д. Чаще всего для этого используется компонент **Button** (см. рис. 20). Изменяя его свойство **Caption**, можно сделать нужную надпись на кнопке, для настройки шрифта служит свойство Font. Придав свойству **Enabled** значение **False**, можно сделать кнопку недоступной (бледной на вид и не реагирующей на нажатие). Присвоив свойству **Visible** значение **False**, можно сделать кнопку невидимой.

Командная копка с дополнительными возможностями

Компонент **BitBtn**, расположенный на вкладке **Additional**, функционально мало, чем отличается от кнопки **Button** (см. рис. 20). Меняя его свойство **Kind**, можно вывести на кнопку стандартную пиктограмму. Придав данному свойству значение **bkClose**, можно добиться реализации закрытия формы при нажатии на кнопку. Остальные главные свойства у компонентов **BitBtn** и **Button** совпадают.

Независимый переключатель

Флажок – независимый переключатель реализуется компонентом **CheckBox**. Он устанавливается на форме, когда нужно, например, выбрать один из вариантов расчета. Элемент находится на вкладке «Стандартная» и обозначается пиктограммой  . Рядом с полем, куда ставится галочка, находится пояснительный текст. Для того, чтобы сбросить или установить флажок, надо щелкнуть мышью в квадратике. Нередко на форму помещают несколько элементов данного типа. В этом случае они оправдывают свое название – независимые переключатели. Включение – выключение одного из них ни как не сказывается на остальных (см. рис. 21) .

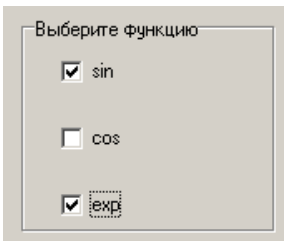


Рис. 21. **Группа независимых переключателей**

Основные свойства данного элемента представлены в таблице 4.

Таблица 4

Основные свойства компонентов CheckBox и RadioButton

Свойство	Назначение
Caption	Текст, поясняющий назначение флажка
Checked	Состояние, внешний вид флажка: если флажок установлен (в квадратике есть «галочка»), то **Checked** = True (истина), если флажок сброшен (нет «галочки»), то **Checked** = False (ложь).
Font	Шрифт, используемый для отображения поясняющего текста.
Alignment	Определяет положение надписи (слева или справа) относительно флажка.

Зависимые переключатели

Зависимые переключатели используются, объединившись в группу (см. рис. 22). Они реализуются при помощи компонентов **RadioButton** и используются для выбора взаимоисключающих альтернатив. Компонент представляет собой флажок, который включается или сбрасывается по щелчку мыши. Щелчок по одному из элементов вызывает сброс других элементов, в каждый момент времени в «выбранном» состоянии может быть лишь один из зависимых переключателей.

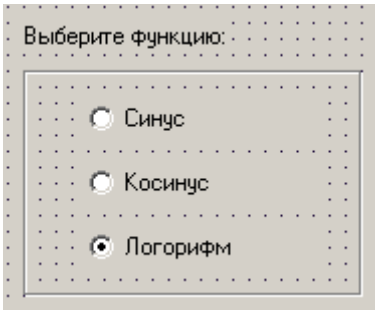


Рис. 22. **Группа зависимых переключателей**

Располагается **RadioButton** на вкладке Стандартная палитры компонентов, пиктограмма -  . Также как и в случае с независимым переключателем,

рядом с кнопкой находится поясняющий текст. Основные свойства данного элемента приводятся в табл. 6. Группа зависимых переключателей также может быть реализована при помощи компонента **RadioGroup**.

Групповая панель GroupBox

Данный элемент представляет собой контейнер с рамкой и надписью, который может объединять различные элементы на форме, логически связанные по своему назначению. Надпись поясняет общее назначение элементов или порядок работы с ними. В качестве элементов, помещенных в GroupBox, часто выступают несколько независимых переключателей (см. рис. 77). Порядок работы с компонентом следующий: вначале на форму помещают элемент **GroupBox**, выравнивают его, а затем помешают внутрь его другие элементы. Компонент находится на вкладке **Стандартная** и имеет пиктограмму  . В таблице 5 представлены основные свойства данного компонента:

Таблица 5

Основные свойства компонента GroupBox

Свойство	Назначение
Caption	Пояснительная надпись в углу рамки панели.
Color	Цвет фона контейнера
Font	Определяет атрибуты шрифта надписи.

Многострочное поле ввода вывода

Для ввода и вывода многострочных данных удобно использовать компонент **Memo**. Компонент добавляется обычным образом и расположен на вкладке **Standard**, его пиктограмма -  .

Порядок работы с компонентом на этапе проектирования формы следующий:

1. Перейти в свойство **Lines**.

2. В открывшемся редакторе набрать строки текста.
3. Выйти из редактора.

Основные свойства данного компонента перечислены в таблице 6:

Таблица 6

Основные свойства компонента Memo

Свойство	Описание
Lines	Текст, находящийся в поле Memo. Рассматривается, как совокупность строк. Доступ к строке осуществляется по номеру. Например, a[i]:=Memo1.Lines[i];
Lines.Count	Количество строк текста в поле редактора. Доступно из программы.
Color	Цвет фона
Font	Шрифт
ReadOnly	Значение True не позволяет редактировать текст.
WordWrap	Значение True устанавливает режим переноса текста в другую строку при превышении ширины компонента.
ScrollBars	Установка полос прокрутки: ssNone – отсутствие полос прокрутки, ssBoth присутствуют обе полосы, ssHorizontal – присутствует горизонтальная полоса, ssVertical – присутствует вертикальная полоса.

Данный компонент имеет несколько полезных методов, с двумя из которых мы познакомимся:

1. **Clear** – удаляет текст из окна. Например, Memo1.clear.
2. **Lines.Add**(строка) – добавляет строку в конец таблицы. Например, Memo2.Lines.Add(‘Добавлена последняя строка’).

Получить доступ к находящейся в редакторе строке текста можно при помощи свойства Lines, указав в квадратных скобках номер нужной строки (строки нумеруются с нуля).

Считать введенные строки текста в массив можно при помощи следующих операторов:

```
For i:=0 To Size do
A[i]:= Memo1.Lines[i];
```

Универсальная таблица StringGrid

StringGrid – мощный компонент, который позволяет решать самые разнообразные задачи. С его помощью, например, можно реализовать электронную таблицу. Размещенный на форме **StringGrid** похож на обыкновенную таблицу, ячейки которой предназначены для записи текстовой информации (см. рис. 23) . Компонент находится на вкладке Additional и имеет пиктограмму abc.

Рис. 23. **Компонент StringGrid, нумерация строк, столбцов подписана компонентом Label**

Основные свойства компонента представлены в таблице 7:

Таблица 7

Основные свойства компонента StringGrid

Свойство	Значение
Cells	Представляет собой двумерный массив строк, который и составляет таблицу. Используется для записи или считывания данных из ячеек. Ссылки на ячейки осуществляются при помощи двух индексов, изменяющихся от нуля до заданных предельных значений. Первый индекс определяет номер столбца, второй – номер строки. Свойство схоже с переменной типа массив. Для того, чтобы записать какое-либо число, например, 2,5 в ячейку с индексами 2, 3 следует добавить в программу такую строку StringGrid1.Cells[2,3]:=’2,5’;(см. рис. 83)
Col	Номер текущего столбца, в котором находится фокус (ячейка выделена).
Row	Номер текущей строки, в которой находится фокус. Данные свойства обычно используются, чтобы узнать, в какую ячейку введена информация.
ColCount	Число столбцов таблицы
RowCount	Число строк таблицы.
FixedCols	Число столбцов, используемых для пояснительных надписей.
FixedRows	Число строк, используемых для пояснительных надписей.
Options	Сложное логическое свойство. Его подсвойства определяют разрешенные операции по модификации таблицы. В частности подчиненное свойство GoEditing определяет можно ли редактировать таблицу (значение True).
BoderStyle	Определяет тип рамки таблицы.

Наиболее распространенный порядок работы с данным компонентом.

1. Нанести компонент на форму.
2. Задать число строк и столбцов (свойства ColCount и RowCount)
3. Задать число строк и столбцов, используемых для пояснительных надписей.
4. При необходимости задать дополнительные установки (Options и BoderStyle).
5. Отрегулировать размеры компонента.

Покажем, каким образом можно создать заголовки строк и столбцов таблицы. Предположим, что надо создать следующую таблицу:

N	X	Y
1		
2		
3		

Для этого можно использовать следующий фрагмент программы:

```
with form1.StringGrid1 do
    begin
    cells[0,0]:='N';
    cells[1,0]:='A';
    cells[2,0]:='B';
    for i:=1 to 3 do
    cells[0,i]:=IntToStr(i);
      end;
```

Создание диаграмм

Для этой цели используется компонент **Chart**. Данный компонент имеет очень широкие возможности. Располагается **Chart** на вкладке Additional и имеет пиктограмму  . Размещенный на форме компонент представляет собой панель (прямоугольную область), на которой можно создавать диаграммы и графики различных типов. Один компонент **Chart** можно использовать для создания сразу нескольких графиков или диаграмм других типов. В этом смысле его можно рассматривать как координатную плоскость, на которой нанесены несколько кривых. Каждый из графиков (рядов данных) является от-

дельным объектом **Series** (серия данных) и может иметь различные стили отображения.

Порядок работы с компонентом на этапе создания формы следующий:

I. Нанести на форму компонент **Chart** и откорректировать его линейные размеры.

II. Открыть Редактор Диаграмм (свойство **SeriesList** или двойной щелчок на компоненте). Страница редактора имеет много вкладок, по умолчанию установлена на вкладке **Series** (см. рис. 24).

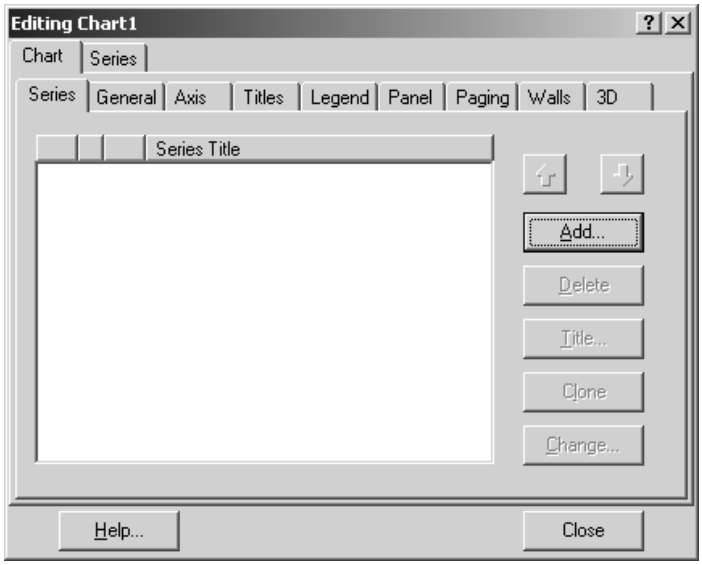


Рис. 24. **Окно настройки диаграммы**

III. На вкладке Series нажать на кнопку ADD, которая добавляет новую кривую на график.

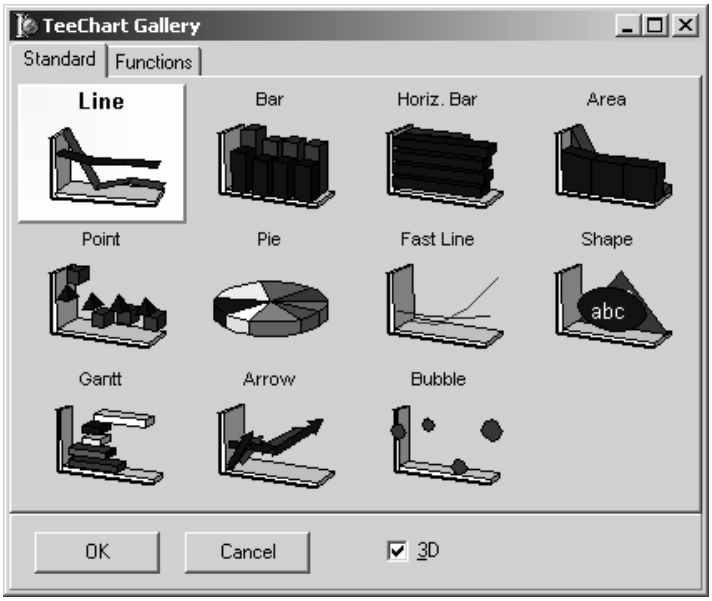


Рис. 25. **Выбор типа диаграммы**

IV. В раскрывшемся списке выбрать желаемый тип диаграммы или графика (см. рис. 25). Закрыть список. На форме появится график случайным образом заданной функции (см. рис. 26).

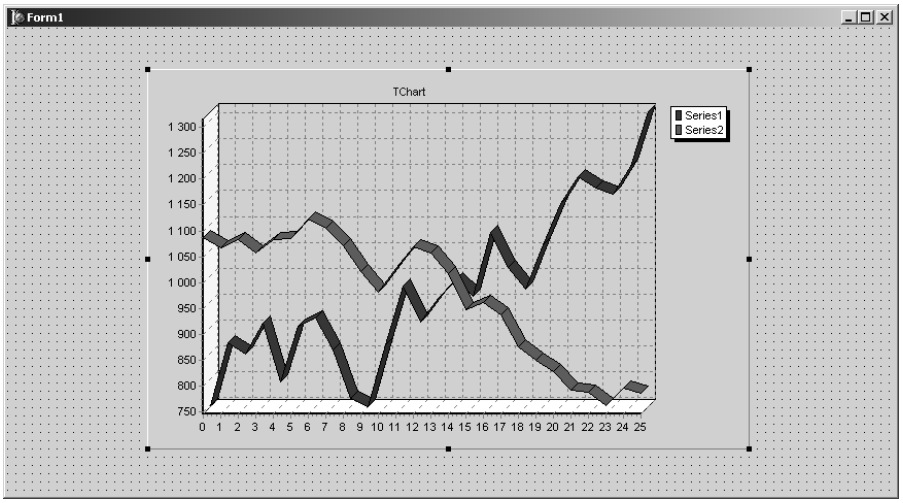


Рис. 26. **Вид компонента Chart после добавления двух объектов Series**

V. Перейти на вкладку **Titles**, задать заголовок графика.

VI. Перейти на вкладку **Legend**, задать параметры отображения легенды диаграммы или вообще убрать ее с экрана.

VII. Перейти на вкладку **Panel**, откорректировать внешний вид панели **Chart.**

VIII. Перейти на вкладку **3D**, откорректировать вид кривой графика: наклон, сдвиг, толщину.

IX. Если необходимо, перейти на страницу **Series** и задать дополнительные характеристики отображения кривой графика.

Для работы с компонентом из программы используются методы объекта **Series.**

1) **Исключение ненужной кривой с графика** – метод **Clear**: Например, **Series1.Clear**;
2) **Добавление на диаграмму новой точки** – метод **ADD** в общем виде: **Series1.Add(A,S,Color);**

где A – выражение, задающая 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 нее четыре новых значения, задавая отображающие их цвета (результат выполнения фрагмента программы показан на рис. 27):

```
With Series1 do
Begin
Clear;
Add(13,'цех 1',clYellow);
Add(18,'цех 2',clBlue);
Add(21,'цех 3',clRed);
Add(9,'цех 4',clPurple);
End;
```

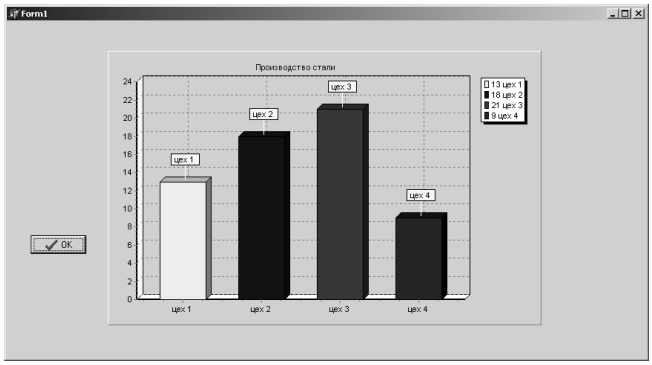


Рис. 27**. Пример построения гистограммы.**

3) **Добавление новой точки на график функции** – метод **AddXY** в общем виде:

Series1.AddXY(X,Y,S,Color);

где X,Y – выражения, задающие координаты точки на 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 стирают прежнюю кривую и наносят на график кривую синуса (см. рис. 28):

```
Series2.Clear;
For i:=0 to 20 do
Series.AddXY(i,sin(i),'',clRed);
```

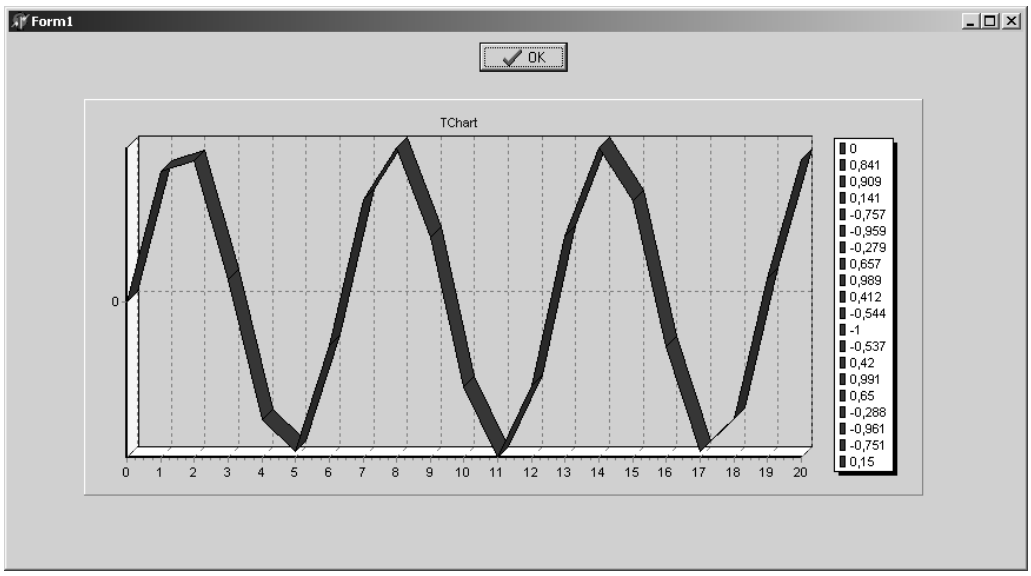


Рис. 28. **Пример построения графика**

Главное меню

Компонент **MainMenu** отображает на форме главное меню. Значок компонента находится на вкладке стандартная и похож на раскрывшееся меню:  . Этот значок можно поместить в любое место формы, т.к. во время выполнения программы он не виден. Пункты меню появляются в верхней части формы в результате настройки меню (см. рис. 29).

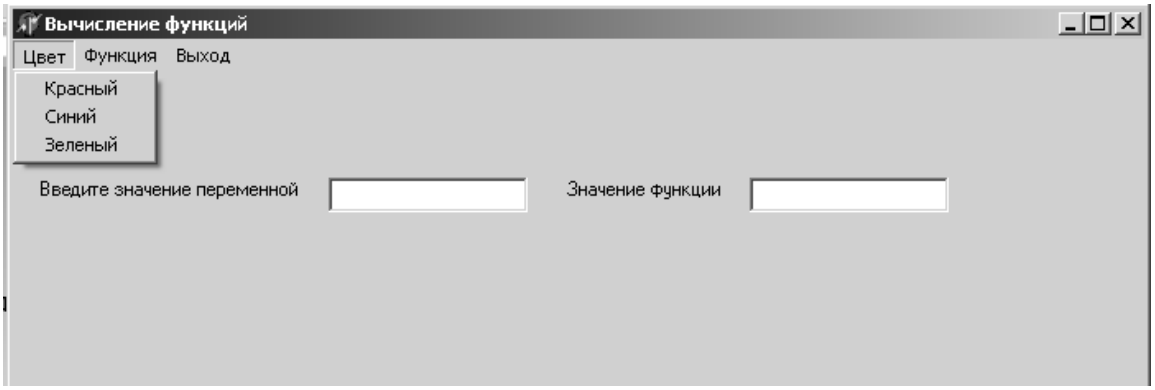


Рис. 29**. Использование компонента MainMenu**

С объектом **MainMenu** следует работать следующим образом:

1. Перейти в свойство **Items**, вызвать соответствующий редактор или два раза щелкнуть на компоненте. В начале работы над новым меню в окне редактора находится один-единственный прямоугольник – заготовка

пункта меню. Чтобы добавить в главное меню элемент, необходимо ввести текст в пустой элемент (см. рис. 30). Если поставить перед названием знак &, то во время работы программы можно будет обращаться к этому пункту, нажимая комбинацию клавиш Alt и первой буквы пункта меню. Эта буква, кроме того, станет подчеркнутой.

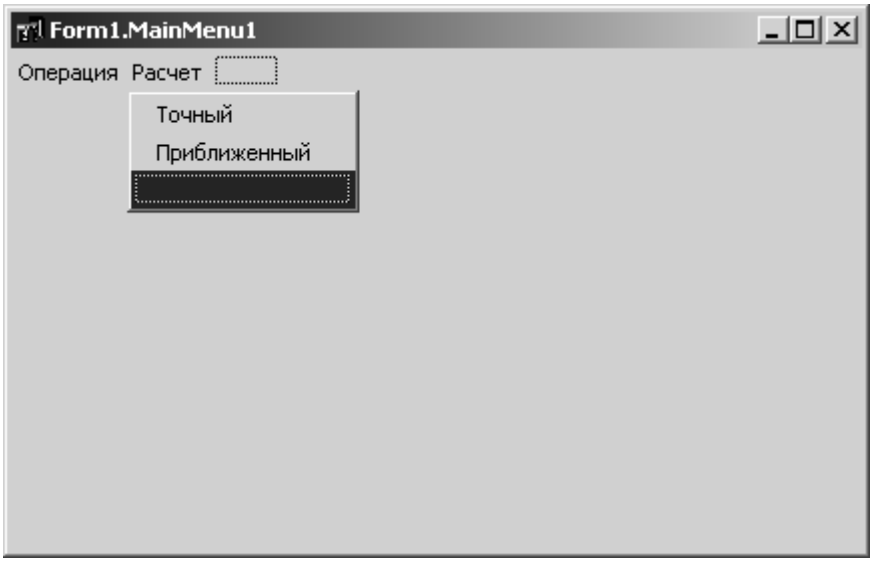


Рис. 30. **Пример настройки главного меню**

2. Создать названия пунктов, перемещаясь по заготовке меню при помощи стрелок. При завершении ввода пункта меню нажимать клавишу Enter. Каждый новый пункт меню представляет собой новый объект типа TmenuItem, называемый N1, N2 … Свойства этих объектов определяют вид меню во время работы программы. Название пункта меню содержится в свойстве **Caption** каждого объекта. К списку свойств данного типа объектов также относятся известные нам **Enabled, Visible,** которые могут сделать пункт меню недоступным или невидимым. При желании можно создать разделитель между пунктами меню. Для этого вместо названия пункта меню вводится символ “-“.
3. Для удаления или вставки нового пункта меню использовать соответственно клавиши **Delete, Ins** или вызов контекстного меню.
4. Когда меню создано, закрыть редактор.
5. Создать процедуры, которые необходимо выполнить после нажатия на каждый из пунктов меню. Для этого достаточно один раз щелкнуть мы-

шью на заготовке формы по нужному пункту меню. Система тотчас же создаст необходимый шаблон процедуры обработки этого события. Вписать необходимый программный код. При выборе во время работы элемента меню происходит событие **OnClick.**

Счетчик времени Timer

Счетчик времени представляет собой невизуальный компонент, позволяющий задавать в приложении интервалы времени. В результате работы компонента генерируется событие **OnTimer**, в обработчике этого события можно прописать любые действия. **Timer** находит многочисленные применения: синхронизация мультипликации, закрытие окон, с которыми пользователь долгое время не работал, регулярный опрос источников информации, задание отсчета времени в обучающих программах. Находится **Timer** на вкладке **System** и имеет характерную пиктограмму  . Значимых свойств у данного компонента всего два:

Interval - задает расчетный интервал времени в миллисекундах, после которого генерируется событие **OnTimer**. Значение 0 соответствует выключенному таймеру. **Interval**=5000 означает, что каждые 5 секунд будет генерироваться событие **OnTimer**.

Enabled – включение/выключение часов. Значение **True** включает таймер, **False** - выключает таймер

Работа с компонентом очень проста:

- его следует расположить в любом месте формы,
- придать необходимые значения его свойствам,
- зайти в **Инспектор объектов** на вкладку **Events**, два раза щелкнуть в пустом поле напротив события **OnTimer.**
- записать текст программы в раскрывшемся редакторе кода.

Выключить таймер из программы можно двумя способами. Для этого следует записать следующие строки:

Timer1.Interval:=0;

или

Timer1.Enabled:=False;

Включить таймер можно, также двумя способами:

Timer1.Interval:=10000; (в том случае, если интервал был равен нулю)

или

Timer1.Enabled:=True;

Ниже приводим пример процедуры, закрывающей некоторую форму через 15 сек.:

procedure TForm1.Timer1Timer(Sender: TObject);

begin

Close;

End;

Предварительно следует задать значение свойства **Interval**:=15000.

Язык программирования Object Pascal

Для того, чтобы оживить разработанные окна приложения: заставить программы считать и выполнять всевозможные действия служит язык программирования. В основу системы объектно-ориентированного программирования **Delphi** положен язык **Object Pascal**, который можно также называть **Delphi.** Программа на этом языке представляет собой последовательность операторов, разделенных символом «;».В программу включается несколько зарезервированных слов языка, описывающих ее структуру (см. рис. 31).

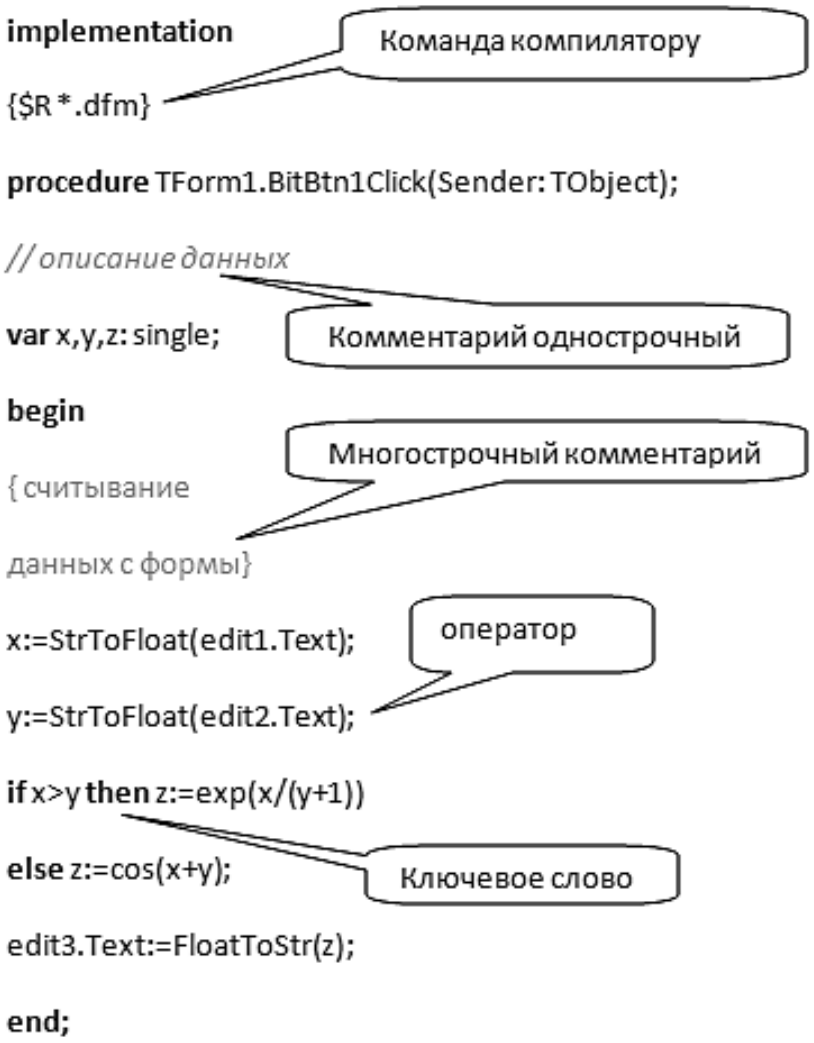


Рис. 31. **Фрагмент программы на языке Object Pascal**

Оператор состоит из идентификаторов. При записи идентификаторов могут использоваться латинские буквы, цифры, символ подчеркивания «_». Идентификатор не может начинаться с цифры и не может содержать пробелов. Длина идентификатора не ограничена, но воспринимается не более 255 первых символов. Впрочем, лучше использовать короткие, но осмысленные идентификаторы. Идентификаторы представляют собой:

- зарезервированное (ключевое) слово языка (например, **:=, if, while, for** и др.);

- переменную (например, X, Temp, West);

- константу (например, целое или дробное число, строку символов);

- арифметическую или логическую операцию(например, +,-,*);

- подпрограмму (процедуру или функцию, например, Sin, Ln);

Программный текст должен начинаться с ключевого слова **Program** или **Unit**, а заканчиваться словом **End.**

Хотя в одной строке можно записать несколько операторов, как правило, каждый оператор записывают в отдельной строке. Нельзя разрывать имена переменных и операторы.

Комментарии служат для включения пояснительного текста в программу, и являются непременным атрибутом даже простой программы. Комментарии заключаются в фигурные скобки и могут занимать любое количество строк. Комментарий в фигурных скобках не может начинаться со знака $, т. к. это означает команду компилятору. Для записи однострочных комментариев используются в начале строки две наклонные черты.

Типы данных

Программа может оперировать со следующими основными типами данных:

- целыми числами (**Integer**)(например, 1,2,3,10,-12),
- дробными числами (**Real**)(например, 1.123),
- символами (**Char**)(например, ‘Ц’),
- строками символов (**String**)(например, ‘qwewer’),
- логическими величинами (**Boolean**)(True, False).

Каждый из основных типов, перечисленных выше, дополняется родственными ему типами, которые отличаются размером занимаемой памяти. В таблице 8 приведены различные типы данных, используемые в Object Pascal.

Таблица 8

Типы данных, используемые в языке Object Pascal

Группа типов	Тип	Занимаемая память, байты
Целые числа	Shortint	1
	Smallint	2
	Longint	4
	Int64	8
	Byte	1 (>0)
	Word	2 (>0)
	Longword	32 (>0)
	Integer	4
Дробные числа	Real48	6
	Single	4
	Double	8
	Extended	10
	Comp	8
	Currency	8
	Real	8
Символы	AnsiChar	1
	WideChar	2
	Char	1
Строки	ShortString	256
	LongString	Ограничение – объем свободной памяти
	WideString	То же, но каждый символ строки занимает 2 байта.
	String	256
Логический тип	Boolean	1

Кроме перечисленных выше типов в программе на языке Object Pascal могут использоваться данные типа **Variant**. В переменных этого типа могут храниться данные любого типа.

Язык **Object Pascal** позволяет оперировать с огромным количеством типов данных, в частности с типами данных, созданных пользователем.

Переменные

Переменная – это поименованная область памяти, куда могут записываться и откуда могут считываться данные. Значение переменной может изменяться и задаваться программой. Чтобы к переменной можно было обратиться, она должна иметь имя, например, X, У. Перед использованием любая переменная должна быть объявлена и ей должен быть присвоен определенный тип, для чего используется ключевое слово Var.

Var B : real;

После ключевого слова **Var** может следовать не одно, а множество объявлений. Каждое объявление может содержать список идентификаторов переменных, разделяемых запятыми.

Var I: integer;
X,Y,Z: Double;
Ok: Boolean;

Описания переменных должны предшествовать их первому использованию.

Константы

Константы могут быть обычными и поименованными. Обычная константа – это целое или дробное число, строка символов или отдельный символ, логическое значение.

Числовые константы записываются, как это принято при решении математических задач. Разделителем дробной части выступает «.». Дробные константы могут изображаться в виде числа с плавающей точкой. Строковые и символьные константы заключаются в кавычки «’». Логические константы принимают значения True или False (истина или ложь) (см. рис. 32).

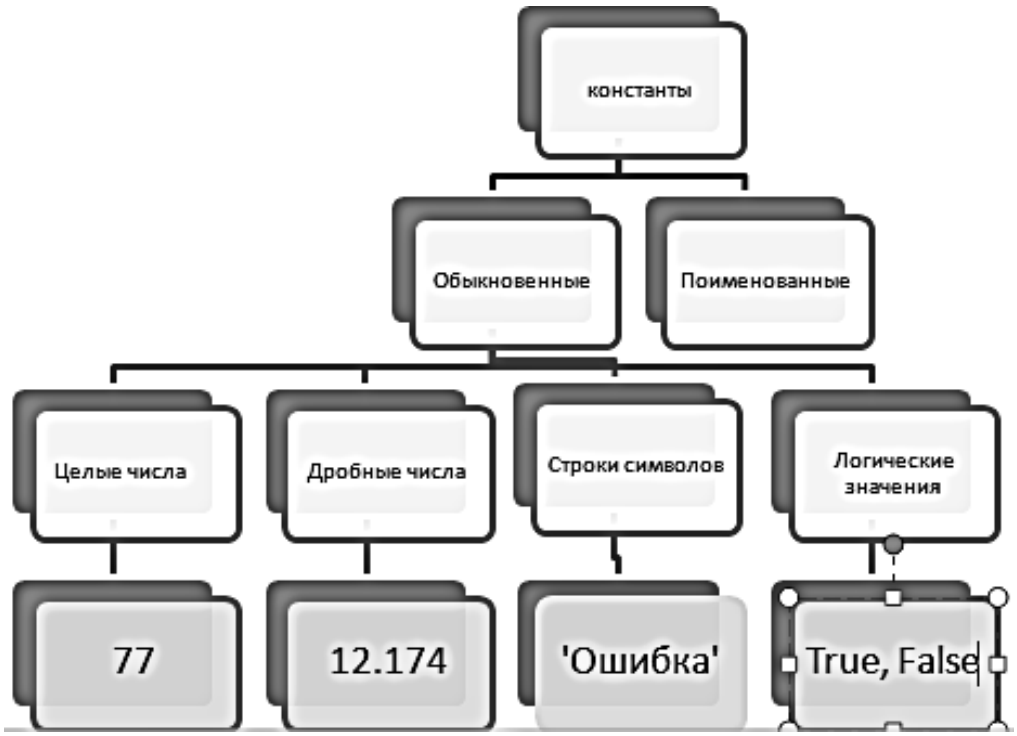


Рис. 32. **Различные типы констант**

Именованная константа – это имя, которое используется в программе вместо самой константы. Она должна быть объявлена в программе в разделе объявления констант:

Const

Alfa = ‘Значение’;

Pi = 3.14159265358979323846264338327 95;

Nmax = 55;

Name = ‘Алексей Павлович Проскуряков’;

Оператор присваивания

Оператор присваивания используют в двух случаях:

- ✓ Если надо выполнить вычисление по какой-нибудь формуле, например F:=2*sqr(x)-18*y;
- ✓ Если надо переменной присвоить какое-либо значение, например A:=1;

В общем виде оператор выглядит так:

Имя := Выражение;

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

:= - символ оператора присваивания.

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

Основные алгебраические операторы следующие: +, -, *, /, div (деление нацело), mod (остаток от деления). Для задания нужного порядка вычислений используются скобки (). Примеры выражения:

(2*sin(a)+cos(3*b))/(a+b).

Form1.Color:=clYellow; Здесь в качестве переменной используется свойство формы – цвет. Ему присваивается значение – желтый.

Программный блок или составной оператор

Любые операторы могут быть объединены в программный блок посредством использования зарезервированных слов языка **Begin** и **End.**

Begin

Операторы;

End;

Программный блок воспринимается как один оператор, может включать внутри себя другие программные блоки. В дальнейшем примем, что слово «оператор» может означать как простой оператор, так и программный блок.

Условный оператор

Этот оператор позволяет выбрать один из двух возможных вариантов продолжения алгоритма.

If условие **Then** оператор1

Else оператор2;

Приведем пример: Пусть требуется рассчитать функцию:

$$y = \begin{cases} \sin(x), & x < a \\ \cos(x), & x \geq a. \end{cases}$$

Для этого достаточно написать в программе следующее:

IF x<a **THEN** y:=sin(x)

ELSE y:=cos(x);

Часто используется краткая форма записи оператора IF:

If условие **Then** оператор;

Приведем пример:

IF x<0 **Then begin**

showMessage(‘Аргумент должен быть больше нуля!’);

exit;

end;

В приведенном примере, если переменная отрицательна, выдается сообщение и прерывается работа процедуры.

Простые условия

В программе условие – это выражение логического типа (Boolean), которое может принимать одно из двух значений: **True** (истина) или **False** (ложь). Простое условие состоит из двух операндов и оператора сравнения. Например, X > 0.

В общем виде простое условие записывается так:

Оп1 **Оператор** Оп2,

где операнды Оп1 и Оп2 могут представлять собой переменную, константу, функцию или выражение.

В качестве Оператора может выступать любой из 6 существующих в языке операторов сравнения **(<, >, <>, =, >= ,<=).** Операнды должны быть одного типа или приводиться к одному типу. Например, условие A<>B принимает значение **Истина**, если А не равно В, если А равно Б они принимает значение **Ложь**.

Сложные условия

Они строятся из простых при помощи логических операторов: **and** (логическое И), **or** (логическое Или), **not** (отрицание). В общем случае сложное условие записывается следующим образом:

Логическое выражение1 **оператор** Логическое выражение2,

или

NOT Логическое выражение,

где логические выражения могут представлять собой простые или сложные условия, оператор: – **and, xor** или **or**. Логические выражения следует заключать в скобки.

Как работают логические операторы?

Оператор **and** принимает значение **Истина** только в том случае, если оба его операнда принимают значение **Истина**, иначе он принимает значение **Ложь**. Оператор **or** принимает значение **Истина**, если хотя бы один из его операндов принимает значение **Истина**, иначе он принимает значение **Ложь.** Оператор **Not** меняет логическое значение аргумента на противоположное.

Приведем пример: пусть условие предоставления скидки на покупку сформулировано следующим образом: «Скидка 10% предоставляется, если сумма покупки превышает 500 руб. и день покупки – воскресенье». Если день покупки – целая переменная **Day**, сумма покупки переменная Sum то условие записывается так:

(**Sum** > 500) **and** (**Day**=7)

Расчет скидки может быть осуществлен при помощи оператора If:

IF (Sum > 500) **and** (Day=7) **THEN** Sum:=Sum*0.9;

Оператор множественного выбора

Иногда значение какой-либо переменной или выражения определяют разветвления алгоритма сразу на несколько ветвей. В этом случае удобнее использовать оператор **Case.** Он имеет следующий формат записи:

Case Селектор **of**

Список 1 : Оператор 1;

Список 2 : Оператор 2;

…

Список n : Оператор n;

Else Оператор;

где **Селектор** – выражение, значение которого определяет дальнейший ход выполнения программы; тип выражения – целое, символ или строка символов, **Список** – список констант (например, 2,4,6,8,10). Если константы представляют собой диапазон чисел, то вместо списка можно указать первую и последнюю константу диапазона, разделив их двумя точками (например, 1..1021).

Выполняется оператор **Case** следующим образом:

1. Сначала вычисляется значение выражения-селектора.
2. Значение селектора последовательно сравниваются с константами из списков констант.
3. Если значение выражения совпадает с константой из списка, то выполняется соответствующая этому списку группа операторов. На этом выполнение **Case** завершается.
4. Если значение выражения-селектора не совпадает ни с одной из констант, то выполняется последовательность операторов, следующая за **Else.** Впрочем, **Else** можно и пропустить, в этом случае будет выполняться следующий за **Case** оператор программы.
   Пример оператора **Case**:

   Case N **of**

   1..5: Day :=’Рабочий день’;

   6: Day :=’Суббота’;

   7: Day :=’Воскресенье’;

   End;

Операторы цикла

Алгоритмы решения многих задач являются циклическими, т.е. для достижения результата определенная последовательность действий должна быть выполнена несколько раз.

Операторы цикла в Delphi можно разделить на две группы:

- Циклы со счетчиком;
- Циклы с условием.

Оператор цикла со счетчиком

Цикл со счетчиком используется в том случае, когда заранее известно, сколько раз нужно повторить выполнение заданной группы операторов. В общем виде он записывается следующим образом:

For С **:=** НЗ to КЗ **do**

Begin

// здесь операторы, которые надо выполнять несколько раз.

End;

где: С – переменная-счетчик числа повторений операторов цикла; НЗ. – выражение, определяющее начальное значение счетчика циклов, КЗ - выражение, определяющее конечное значение счетчика циклов. Все перечисленные переменные и выражения должны быть целого типа. Количество повторений цикла можно вычислить по формуле:

N = КЗ - НЗ+1.

Если между **begin** и **end** находится только один оператор, то эти ключевые слова можно не писать.

Примеры: Суммирование n натуральных чисел:

```
s := 0;
for i := 1 to n do
s:=s+i;
```

Расчет и выдача на форму кубов 10 натуральных чисел

```
t:=0;
for i := 1 to 10 do
begin
t:=t+IntToStr(i) + '=' + IntToStr(i*i*i) + chr(13);
end;
Label1.Caption:=t;
```

Последний пример демонстрирует, что переменную-счетчик можно использовать внутри цикла. **Переменную-счетчик ни в коем случае нельзя изменять внутри цикла!**

Цикл с предусловием

Оператор цикла **While** используется в том случае, если некоторая последовательность действий (операторов программы) надо выполнить несколько раз, причем необходимое число повторений во время разработки программы не известно и может быть определено только во время работы программы. Типичными примерами использования данного оператора являются вычисления с заданной точностью, поиск в массиве или файле.

В общем случае оператор записывается следующим образом:

```
While условие do
  Begin
   // здесь операторы, которые надо выполнять несколько раз.
  End;
```

где условие – выражение логического типа, определяющее условие выполнения цикла.

Оператор **While** выполняется следующим образом:

1. Сначала вычисляется значение выражения условие.
2. Если его значение равно False, то на этом оператор завершается.
3. Если условие выполняется, то выполняются операторы, заключенные между **begin** и **end**. После этого снова проверяется выполнение условия. Так продолжается до тех пор, пока условие не станет ложным. Поскольку про-

верка условия осуществляется перед выполнением операторов тела цикла, то если условие сразу ложно, то оператор не будет выполняться ни одного раза. Для того чтобы цикл завершился, необходимо, чтобы в теле цикла находились операторы, влияющие на выражение-условие. Программист должен быть уверен, что рано или поздно условие станет ложным. Если этого не произойдет, то программа «Зациклится», цикл будет выполняться бесконечное число раз и программа зависнет. Иногда такие бесконечные циклы используются, но тогда внутри тела цикла должно быть предусмотрено его прерывание. О том, как это сделать, речь пойдет далее. Пример использования оператора:

Расчет функции y=sin(x) на отрезке [xn,xk] при изменении аргумента с шагом hx.

```
x:= xn;
While x<=xk do
Begin
z:=sin(x);
y:=FloatToStr(z);
Memo1.Lines.Add(y);
x:=x+hx;
End;
```

Цикл с постусловием

Он реализуется с помощью оператора **Repeat.** Цикл с постусловием также используется в программе в том случае, если необходимо выполнить повторные вычисления, но число повторений во время разработки программы неизвестно. Оно определяется ходом вычислений. Организация цикла при помощи этого оператора отличается от использования **While** тем, что условие завершения цикла проверяется после выполнения операторов тела цикла.

Оператор **Repeat** записывается следующим образом:

Repeat

// операторы тела цикла

until условие;

где условие – выражение логического типа, определяющее условие завершения цикла.

Заметим, что операторы тела цикла для **Repeat** выполняются как минимум один раз. Для того чтобы цикл завершился, необходимо, чтобы операторы цикла изменяли значение переменных, входящих в выражение условие. В противном случае произойдет зацикливание, о котором уже шла речь при рассмотрении оператора **While**.

Рассмотрим предыдущий пример «Расчет функции y=sin(x) на отрезке [xn,xk] при изменении аргумента с шагом hx», но организуем цикл при помощи оператора **Repeat**.

```
x:= xn;
repeat
z:=sin(x);
y:=FloatToStr(z);
Memo1.Lines.Add(y);
x:=x+hx;
until x>xk;
```

Операторы управления циклом

Для того, чтобы успешно программировать различные циклические структуры, особенно когда используются вложенные циклы, никак не обойтись без вспомогательных операторов, которые мы объединили под названием «Операторы управления циклом». Эти операторы расширяют возможности программирования и могут быть использованы и в других случаях.

Оператор Goto

Оператор **Goto** позволяет прервать обычную линейную последовательность операторов и передать управление на произвольный оператор, помеченный специальной меткой. Он может быть использован, например, для аварийного завершения процедуры. В общем виде оператор записывается следующим образом:

Goto Метка;

Где Метка – это идентификатор, находящийся перед оператором, который должен быть выполнен сразу после оператора **Goto**. После метки ставится двоеточие. Метка, используемая в операторе **goto**, должна быть объявлена в разделе меток, который начинается словом **Label** и располагается перед разделом объявлений переменных.

```
Label  Bye, 1, By;
Var n : integer;
…..
goto bye;
…..
bye: x:=18;
….
```

В свое время применение данного оператора считалось дурным тоном, т.к. частое его использование приводит к запутанности программы и усложняет ее читаемость. Поэтому часто использовать **Goto** не следует.

Цикл без оператора цикла

Цикл может быть реализован при помощи операторов **if** и **goto**.

Пример: суммирование 3000 натуральных чисел.

```
I:=1;
S:=0;
m1: If  i>3000  Then goto m2;
```

```
S:=S+I;
I:=I+1;
Goto m1;
m2: showmessage(FloatToStr(S));
```

В некоторых случаях желательно прервать повторение цикла нестандартным образом, проанализировав какие-то условия, или изменить линейную последовательность выполнения операторов внутри цикла, пропустить какие-либо операторы. Для этих целей могут быть использованы следующие операторы:

Универсальный прерыватель

Этот оператор прерывает выполнение тела любого цикла и передает управление следующему за циклом выполняемому оператору (см. рис. 33):

Break;

Завершитель процедур и функций

Для прерывания циклов, размещенных в процедурах и функциях, можно воспользоваться процедурой **Exit**. В отличие от предыдущего оператора данная процедура не только прервет выполнение цикла, но и выполнение той процедуры или функции в которой был расположен цикл(см. рис. 33):

Exit;

Безусловный переход Goto

Еще один способ прерывания цикла – использование оператора **goto**, передающего управление какому-то оператору вне тела цикла(см. рис. 33).

Переход на конец цикла

Описанные способы прерывали выполнение цикла. Данная процедура прерывает только выполнение текущей итерации, текущего выполнения тела цикла и передает управление на следующую итерацию (см. рис. 33):

Continue;

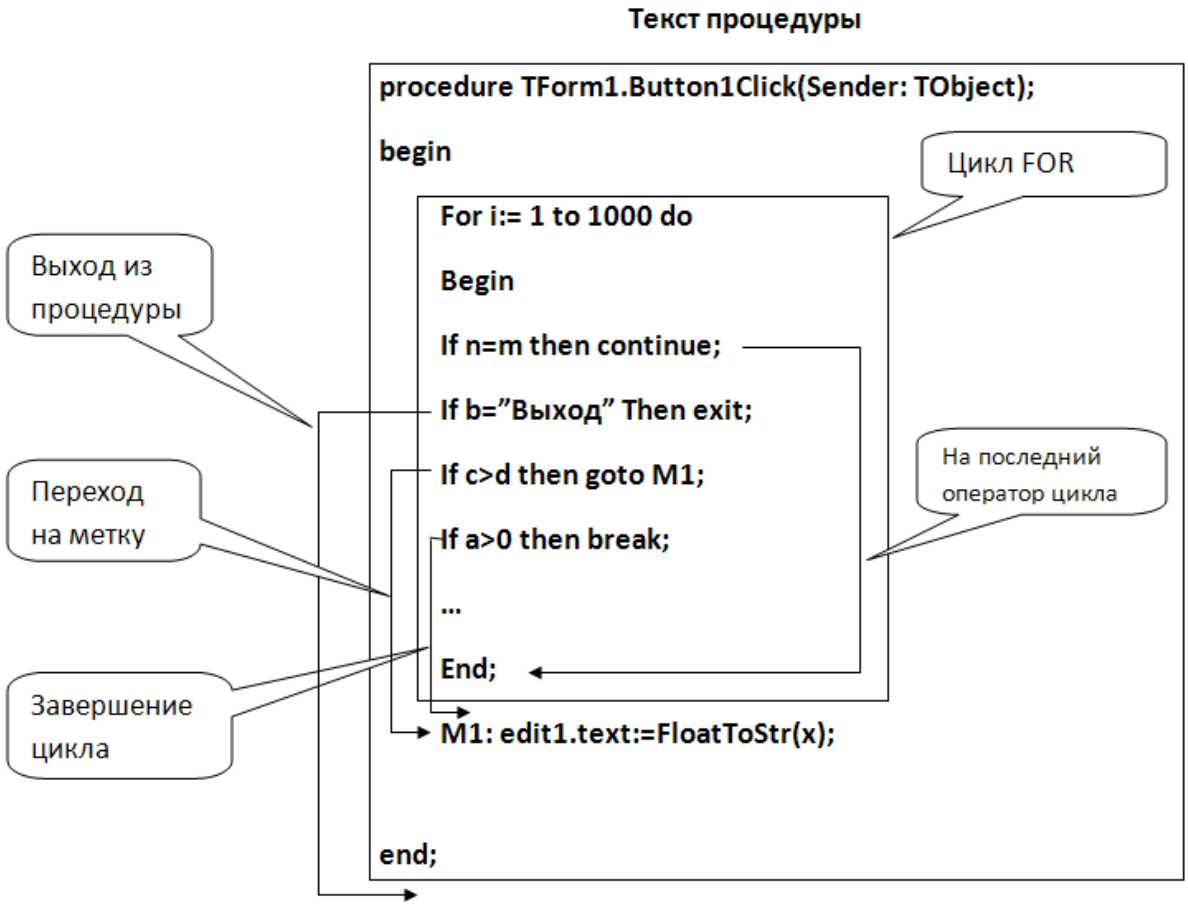


Рис. 33. **Схема работы операторов управления циклом**

Оператор сокращенной записи

Данный оператор упрощает написание текста программ, т.к. позволяет упростить обращение к полям и свойствам сложных типов данных, таких классы (компоненты) и записи. Он позволяет использовать в программе имена свойств (полей) без указания имени объекта, компонента, (записи). В общем виде данный оператор записывается так:

With Имя **do**

Begin

// операторы

End;

где Имя – имя переменной типа класс(компонент) или запись. Приведенная выше запись означает, что в операторах, заключенных между **begin** и **end**, имя объекта при обращении к полям (свойствам) можно не указывать. Пример:

```
With StrihgGrid1 do
  Begin
    Cells[0,0]:=’Фамилия’;
    Cells[1,0]:=’Имя’;
    Cells[2,0]:=’Отчество’;
  End;
```

Дополнительные возможности программирования в Delphi

Массивы

Массив – это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя. Массивы удобно использовать для хранения однородной по своей природе информации, например, таблиц или списков. С массивами мы уже знакомились, изучая Excel. В качестве примера можно привести таблицы оценок студентов по различным предметам.

Массивы характеризуются своим именем, типом хранимых данных, размером (числом хранимых элементов), нумерацией элементов и размерностью. Для ссылки на элементы массива используются индексы, число которых совпадает с его размерностью. Индексами чаще всего являются целые числа, изменяющиеся в заданном диапазоне. Вначале рассмотрим одномерные массивы.

Одномерные массивы

Одномерный массив представляет собой набор данных, упорядоченный в строку или столбец. Например, массив дней недели. Одномерный массив перед использованием должен быть объявлен в разделе объявления переменных:

Var Имя: **array** [нижний_индекс .. верхний_индекс] **of** тип;

где Имя – имя массива,

нижний_индекс, верхний_индекс, – целые константы (как правило), определяющие диапазон изменения индекса элементов массива, тип – тип элементов массива.

Опишем, например, одномерный массив целых чисел размером 6 элементов, имя которого KL

Var kl:**array**[1..6] **of integer**;

Для ссылки на элементы массива используется один индекс:

индекс	1	2	3	4	5	6
ячейки		7				

Для ссылки на элемент массива следует указать имя массива и заключить индекс в квадратные скобки. Например, запишем число 7 во второй элемент:

Kl[2]:=7;

Другой пример объявления массива целых чисел в 30 элементов:

Var Koef: **array**[0..30] **of real**;

При объявлении массивов можно использовать поименованные константы

Var Team : **array** [1..nt] **of string**[sn];

Обратимся к первому элементу этого массива:

Team[1]:=’xxx’;

Если массив – глобальная переменная, то одновременно с объявлением можно выполнить инициализацию (присвоение начальных значений):

Var A : **array**[0..10] **of integer =** (0,0…0);

Существует и другой способ объявления массива, предпочтительный в некоторых случаях. Вначале можно объявить новый тип данных – определенный массив, а затем определять переменные данного типа.

Например:

Type Arr **= array** [0..6] **of integer**;

Var A,B:Arr;

Многомерные массивы

Очень часто приходится иметь дело с информацией, которая представлена в табличной форме. Мы можем привести множество таких примеров из нашей практики работы в Excel. Если вся таблица содержит однородную информацию, например, только целые числа, то она может быть представлена в виде массива. В общем случае объявление двумерного массива выглядит так:

Var Имя: **array** [нижний_индекс 1 .. верхний_индекс 1,

нижний_индекс 2 .. верхний_индекс 2,] **of** тип;

где Имя – имя массива,

нижний_индекс 1, верхний_индекс 1, нижний_индекс 2, верхний_индекс 2 – целые константы, определяющие диапазон изменения индекса элементов массива, тип – тип элементов массива. Для ссылки на элементы массива нужно использовать два индекса.

Например, таблица учета успеваемости студентов

Индексы:i j		1	2	3	4
0	Фамилия	Информатика	Физика	Ин.Яз.	Химия
1	☺ Иванов	3	4	5	5
2	😐 Петров	3	3	3	5
3	☹Сидоров	4	5	4	5

может быть описана в виде двумерного массива:

Var Ozen: **array**[1..3, 1..4] **of integer**;

Количество элементов двумерного массива можно вычислить по формуле:

N = (ВИ1-НИ1+1)*(ВИ2-НИ2+1),

где ВИ – верхний индекс, НИ – нижний индекс. В нашем случае, число элементов – 12.

Для того, чтобы использовать элемент массива, нужно указать имя массива и индексы массива. Первый индекс соответствует номеру строки таблицы, второй – номеру колонки. Так, элемент Ozen[2,4] соответствует оценке студен-

та Петрова по физике. Чтобы изменить оценку необходимо, например написать: Ozen[2,4]:=4;

При работе с таблицами очень удобно использовать оператор **for**. Например, фрагмент программы, вычисляющий среднюю оценку группы по физике, выглядит так:

```
s:= 0;
for i :=1 to 3 do
s:= s + ozen[i,2];
s:= s/3;
```

Фрагмент программы, вычисляющий среднюю по группе оценку, выглядит так:

```
s:= 0;
for i :=1 to 3 do
for j :=1 to 4 do
s:= s + ozen[i,j];
s:= s/12;
```

В приведенном фрагменте программы каждый раз, когда внутренний цикл (по j) завершается, во внешнем цикле значение I увеличивается на 1 и внутренний цикл выполняется вновь. Таким образом, к текущему значению переменной s последовательно прибавляются все значения элементов массива.

Стандартные функции

Для выполнения часто встречающихся вычислений и преобразований язык Object Pascal предоставляет программисту ряд стандартных функций. Значение функции связано с ее именем. Поэтому функцию можно использовать в качестве операнда выражения, например, в операторе присваивания. Так, чтобы вычислить квадратный корень, достаточно записать k:=Sqrt(n), где Sqtr – функция вычисления квадратного корня, n – переменная, которая содержит число, квадратный корень которого следует вычислить. Тип переменной, которой присваивается значение функции, должен соответствовать типу функции.

Математические функции

Они позволяют выполнять различные математические вычисления (см. табл. 9):

Таблица 9

Математические функции

Функция	Пояснение
Abs(x)	Абсолютное значение x.
Sqrt(x)	Квадратный корень их x.
Sqr(x)	Квадрат x.
Sin(x)	Синус x.
Cos(x)	Косинус x.
Arctan(x)	Арктангенс x.
Exp(x)	Экспонента x.
Ln(x)	Натуральный логарифм x.
Random(n)	Случайное целое число в диапазоне от 0 до n-1.

Функции преобразования типа

Функции преобразования наиболее часто используются в операторах, обеспечивающих ввод и вывод информации. Например, для того чтобы вывести в поле вывода **Label** значение переменной целого типа, необходимо преобразовать число в строку символов, изображающую данное число. Это можно сделать при помощи функции IntToStr, которая возвращает строковое представление значения выражения, указанного в качестве параметра функции. Необходимо записать следующий оператор:

Label1.Caption := IntToStr(x).

Ниже приведены основные функции данного типа (см. табл. 10):

Таблица 10

Функции преобразования типа

Функция	Пояснение
IntToStr(k)	Преобразует целое **к** в строку.
FloatToStr(x)	Преобразует вещественное **x** в строку.
StrToInt(s)	Преобразует строку символов **s** в целое число.
StrToFloat(s)	Преобразует строку символов **s** в вещественное число.
Round(n)	Округление действительное число до целого.
Trunc(n)	Целое число, получающееся отбрасыванием дробной части **x.**
Frac(x)	Вещественное число, представляющее собой дробную часть вещественного **x.**
Int(x)	Действительное число, получающееся отбрасыванием дробной части **x.**

Структура программного модуля

Проект Delphi представляет собой набор программных единиц – модулей, файлов с установками среды разработки, а также некоторых вспомогательных файлов. Один из модулей – главный, содержит операторы, с которых начинается выполнение программы. Главный модуль приложения полностью формируется системой.

Помимо главного модуля, каждая программа включает в себя еще как минимум один модуль формы, который содержит описание стартовой формы приложения и поддерживающих ее работы процедур. В Delphi каждой форме соответствует свой модуль.

Модуль Delphi – это отдельный, логически независимый блок программных текстов, хранящийся на диске в виде единого файла и имеющий возможность входить в состав проектов Delphi . Он содержит описания данных, процедур, функций. Delphi позволяет программисту поместить свои

подпрограммы в отдельный модуль, а затем использовать их в своих программах, указав имя модуля в списке модулей, необходимых программе. Для этого используется оператор **Uses.** (См. состав модуля).

Для того, чтобы в программе могли применяться функции и процедуры другого 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). Для добавления модуля необходимо в меню **Project** выбрать команду **Add to Project** и в открывшемся диалоговом окне указать имя файла модуля. Увидеть структуру проекта можно в окне **Project Manager**, которое появляется в результате выбора соответствующей команды из меню **View**. После добавления модуля к проекту и включения его имени в список используемых модулей можно выполнить компиляцию программы. Ниже схематически приводится структура модуля:

unit Unit1; // Unit1 - *Имя Модуля*

interface

Uses // *Здесь перечислены другие модули, которые могут использоваться в данном модуле*

// *Здесь находятся описания процедур и функций модуля, которые могут использоваться другими модулями*.

Const // *Раздел объявления констант*

// *Здесь находятся объявления глобальных констант модуля, которые могут использоваться процедурами и функциями модуля*

type // *Раздел объявления типов*

// *Здесь находятся объявления глобальных типов модуля, которые могут использоваться процедурами и функциями модуля*.

var // Раздел объявления переменных

// *Здесь находятся объявления глобальных переменных модуля, которые могут использоваться процедурами и функциями модуля.*

implementation // *Раздел реализации*

// *Здесь находятся описания (текст) процедур и функций данного модуля.*

end.

Глобальные и локальные переменные

Локальными называются переменные, описанные в разделе описания переменных подпрограммы: процедуры или функции. Они доступны для использования только внутри данной подпрограммы.

Глобальными называются переменные, описанные в разделе описания переменных модуля. Они доступны внутри всех подпрограмм модуля, а также из других модулей. Не рекомендуется создавать лишние глобальные переменные, если они могут быть заменены локальными, т.к. это позволяет более эффективно использовать память компьютера. После выхода из подпрограммы память, занимаемая ее локальными переменными, освобождается. Если имя локальной переменной совпало с именем глобальной переменной, то глобальная переменная будет «затерта» локальной, т.е. ее значение будет заменено значением локальной переменной.

Подпрограммы

Процедуры и функции представляют собой подпрограммы, небольшие программы, которые решают часть общей задачи. Они используются для того, чтобы избежать дублирования кода в программе, упростить ее читаемость. С вызовом подпрограммы происходит процесс перехода к выполнению ее операторов. При вызове в подпрограммы передаются аргументы: некоторые переменные, константы, выражения. Внутри подпрограммы аргументы воспринимаются как параметры, с их использованием производится расчет значений некоторых переменных, значения которых возвращаются в вызывающую программу (см. рис. 34):

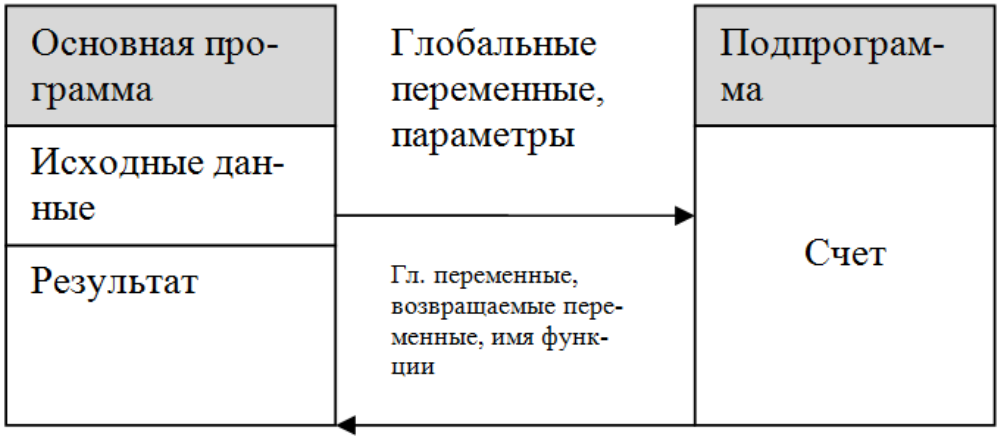


Рис. 34. **Обмен данными между основной программой и подпрограммой**

Для возврата рассчитанных данных в основную программу могут использоваться глобальные переменные, и специальным образом объявленные переменные-аргументы.

Таким образом, параметры подпрограммы используются:

- для передачи данных в подпрограмму;

- для получения результата из подпрограммы.

Если подпрограмма является функцией, то возврат рассчитанного значения может осуществляться через ее имя. В этом заключается отличие функции от процедуры. С именем функции связано значение, поэтому функцию можно использовать в качестве операнда выражения. Нередко она используется в операторе присваивания. Например, оператор **I:= 5*F(x);** вызывает функцию F с аргументом x, умножает возвращенное значение на 5 и присваивает результат переменной I.

Оператор **N:=sin(x)/cos(x); в**ызывает функции синус и косинус и рассчитывает значение тангенса x.

Если функция используется в вычислениях, она должна иметь тип дробного или целого числа. Тип функции определяется в момент ее описания. Текст подпрограммы записывается в разделе выполнения модуля (**implementation**).

Приведем теперь примеры обращения к процедурам:

Dupl(2,3*M/100,wer,'Undo');

Данная процедура имеет четыре аргумента, второй аргумент задан в виде выражения, четвертый в виде символьной константы. Возвращает ли данная процедура какие-либо данные в вызывающую программу неизвестно.

ShowMessage('введите данные');

Данная процедура является стандартной и выводит текст в отдельном окне.

Процедура

Memo1.Lines.Add('Иванов И.И. тел. 222 – 12- 44');

является методом компонента Memo, она добавляет новую строчку в соответствующее многострочное поле.

Создание подпрограммы

Для того, чтобы создать подпрограмму, необходимо ее описать, соблюдая определенные правила. Прежде всего, записывается первая строка подпрограммы или ее заголовок. Она начинается с ключевого слова **Function** для функции, и **Procedure –** для процедуры. После этой строки следуют разделы описания локальных переменных и констант, типов. Затем, после ключевого слова **Begin,** следуют расчетные операторы или операторы тела подпрограммы. Заканчивается подпрограмма ключевым словом **End;. Ниже приводится общая схема объявления функции:**

Function *ИмяФункции* (**Var** *Список Параметров1 : Тип1*;
Var *Список Параметров2: Тип2*;
. . .
Var *Список ПараметровI : ТипI*) : тип;

Const

// описания констант

Var

// описания переменных

Begin

// операторы функции

Result:=Значение;

// Присвоение зарезервированной переменной языка рассчитанного значения функции.

End;

Ниже приводится общая схема объявления процедуры:

Procedure *ИмяПроцедуры* (**Var** *Список Параметров1 : Тип1;*

Var *Список Параметров2: Тип2*;

. . .

Var *Список ПараметровI : ТипJ*);

Const

// описания констант

Var

// описания переменных

Begin

// операторы процедуры

End;

Где **Var** – необязательное ключевое слово, **Типi** – тип параметров i, **тип** – тип функции, **Result** – зарезервированная переменная языка, которая должна содержать результат вычисления функции. Вместо нее можно использовать имя функции. Переменные, помеченные ключевым словом **Var,** могут возвращаться из подпрограммы в вызывающую программу. Приведем пример описания простой процедуры, вычисляющей площадь круга заданного радиуса **r**:

```
procedure kol(r:real;var s:real);
var pi:real;
begin
pi:=3.14;
s:=pi*sqr(r);
end;
```

Рассчитанное значение площади круга возвращается через переменную s помеченную ключевым словом **var**. После того, как процедура описана, ее можно использовать в тексте программы, например оператор kol(5,x); рассчитает площадь круга диаметра 5 и поместит результат в переменную x.

Программирование событий

Понятия событий играет важную роль в визуальном программировании. Программа в Delphi не делает ничего иного, как реагирует на происходящие во внешней среде события. Событие – это изменение состояния компьютера и его периферических устройств. Щелчок на изображении командной кнопки, нажатие клавиш клавиатуры, перемещение мыши и др. – это примеры того, что в Delphi называют событием. В Delphi каждому событию присвоено имя. В Delphi-7 оно начинается с приставки **on**. Например, щелчок кнопкой мыши – это событие **OnClick**, двойной щелчок кнопкой мыши – это событие **OnDbClick**.

Обработка событий

Компонент может реагировать на определенные события, происходящие в процессе работы приложения. Список этих событий представлен в инспекторе объектов на вкладке **Events**. Возьмем для примера события для командной кнопки (см. рис. 35):

Рис. 35. **Список событий компонента Button**

Реакцией объекта на произошедшее событие может быть выполнение им специальной процедуры, называемой процедурой обработки события. Любому событию компонента может быть назначена процедура его обработки. В таблице 11 представлен список наиболее типичных событий, на которые способны реагировать различные объекты.

Таблица 11

Основные события

Событие	**Когда происходит**
1	**2**
OnClick	При щелчке кнопкой мыши
OnDbClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При отпускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyUp	При отпускании нажатой клавиши клавиатуры

Окончание табл. 11

Основные события

1	2
OnCreate	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
OnPaint	При появлении окна на экране вначале работы программы, после появления части окна, которая, например, была закрыта другим окном
OnEnter	При получении элементом управления фокуса
OnExit	При потере элементом управления фокуса
OnTimer	При отсчете интервала времени компонентом Timer
OnChange	При изменении состоянии компонента, например, при вводе данных в однострочное поле ввода-вывода

Система автоматически генерирует шаблон процедуры обработки событий, а программист должен написать основной текст данной процедуры. Delphi по умолчанию создает шаблон реакции на одно из событий при двойном щелке на объекте. Для командной кнопки такое событие по умолчанию будет OnClick. Если пользователь хочет создать шаблон процедуры реакции на другие события, то он должен выполнить следующие действия (см. рис. 36):

1. Выделить необходимый объект.
2. Перейти в инспектор объектов на вкладку Events.
3. Найти в списке нужное событие и щелкнуть два раза в пустом поле справа от названия события.

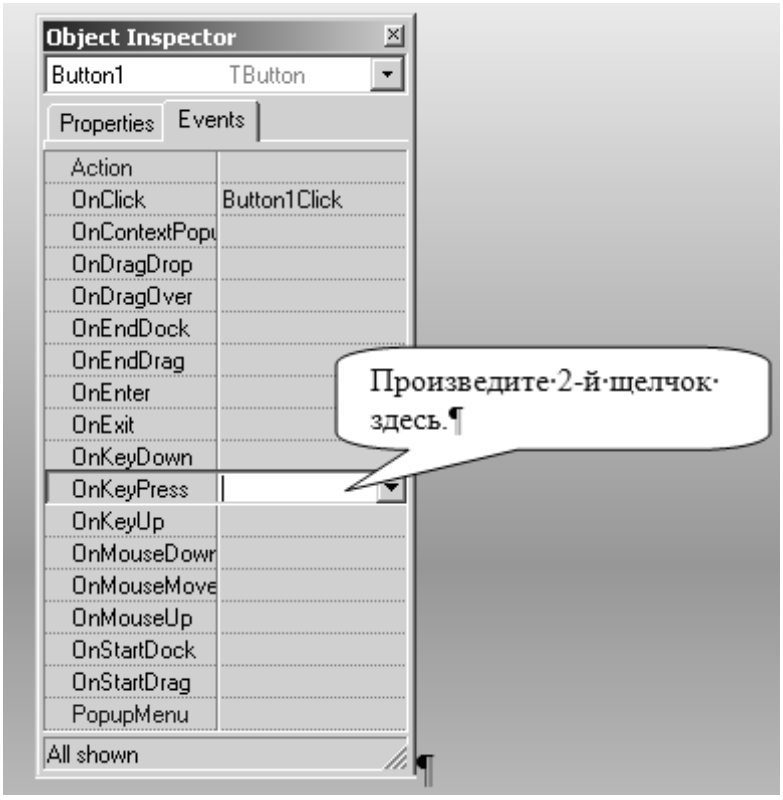


Рис. 36. **Создание заготовки процедуры реакции на произвольное событие**

В поле запишется название процедуры, а редактор кода раскроется и в нем появится готовый шаблон необходимой процедуры.

```
procedure TForm1.Button1KeyPress(Sender: TObject; var Key: Char);
begin
end;
```

Delphi присваивает процедуре обработки события имя, которое состоит из трех частей (см. рис. 37). Первая часть имени идентифицирует форму, содержащую объект. Вторая часть имени идентифицирует сам объект, третья – программируемое событие (при этом частица **on** отбрасывается).

Рис. 37. **Три части имени процедуры обработки события**

Перевести название приведенной выше процедуры можно так: на форме 1 при вводе данных в поле Edit1 нажали на клавиатуре очередную клавишу.

Другие примеры заголовков процедур:

TForm1.FormCreate,

TForm1.Edit1Change,

TForm2.Edit1KeyPress,

TForm5.Label1Click,

TForm3.Button1MouseUp.

Аргументы процедуры могут быть различны, иногда они нужны и могут использоваться.

Вопросы для самоконтроля по разделу 2

Система программирования Delphi

1. Какие окна появляются при запуске Delphi и для чего они используются?
2. Для чего применяют Инспектор объектов?
3. Как используют палитру компонентов?
4. Какая информация записывается в Редакторе кода?
5. Что такое форма?
6. Каким образом осуществляется создание заготовки окна будущего приложения?
7. Что такое компонент?
8. Каковы основные характеристика объектов Delphi?
9. Что такое методы объекта?
10. Что могут представлять собой значения свойств объекта?
11. Какую команду следует дать при зависании программы Delphi?
12. Как создать заготовку процедуры, обрабатывающую нажатие на пункт главного меню?
13. Что такое событие?
14. Как называется событие, возникающее при создании компонента?

15. Как создать заготовку процедуры, обрабатывающую наступление произвольного события?
16. Из каких частей состоят имена процедур, реагирующих на определенные события?
17. Как запустить программу на счет?

Язык Object Pascal

1. Чем заканчивается программный текст?
2. Как записать комментарий в программе?
3. Каким символом заканчивается любой оператор в программе?
4. Для чего используется оператор присваивания, какие символы используются при его записи?
5. Перечислите основные типы данных.
6. Как объявить переменную?
7. Как определить массив целого типа, состоящий из 100 элементов?
8. Какие функции преобразования типа используются при организации ввода-вывода?
9. Как записать стандартную функцию вычисляющую экспоненту?
10. Для чего используется логический оператор, с какого ключевого слова он начинается?
11. Что представляет собой условие в логическом операторе и для чего оно используется?
12. С какого ключевого слова начинается оператор множественного выбора?
13. Чем отличаются циклы с условиями от цикла со счетчиком?
14. Что такое зацикливание и как его избежать?
15. С какого ключевого слова начинается цикл с постусловием?
16. Какой оператор организует цикл со счетчиком?
17. Как обратиться к свойству компонента из программы?
18. Чем функция отличается от процедуры?
19. С какого ключевого слова начинается процедура?

20. Запишите оператор, выдающий на график точку с координатами **A, B.**

Тема 3: Компоненты Delphi

1. Какой компонент используется для создания пояснительных надписей на форме?
2. Какие компоненты используются для создания командных кнопок?
3. Какой компонент используется для организации многострочного ввода-вывода?
4. Какой компонент лучше всего использовать для ввода числового данного?
5. Какое свойство отвечает за состояние «выбранности» независимого переключателя?
6. Какие компоненты используются для создании группы зависимых переключателей и чем они отличаются?
7. При помощи какого компонента можно показывать в окнах диалога диаграммы и графики?
8. Каков порядок действий для подготовки к вычерчиванию графика?
9. Какой метод используется для добавления новой точки на график?
10. Какой метод используется для добавлении я точки данных на диаграмму?
11. Какой метод надо использовать, чтобы закрыть окно приложения?
12. Какой метод используется для добавление новой строки в многострочное поле ввода-вывода?
13. Назовите компонент, при помощи которого можно создать главное меню приложения.
14. Как добавить новый пункт в главное меню?
15. Какие компоненты используются для ввода целых чисел из заданного диапазона?
16. Как настроить компонент Счетчик?
17. Как называется компонент, при помощи которого можно создать набор вкладок?

18. Какой компонент предназначен для создания таблиц в окнах приложений?
19. Как создаются пояснительные надписи к строкам и столбцам таблицы, размещенной в окне приложения Delphi?

Примеры решения задач

Построение форм Delphi

Задание: Построить проекты диалоговых окон для различных программ Delphi в соответствии с образцами. В инспекторе объектов придать компонентам необходимые значения свойств.

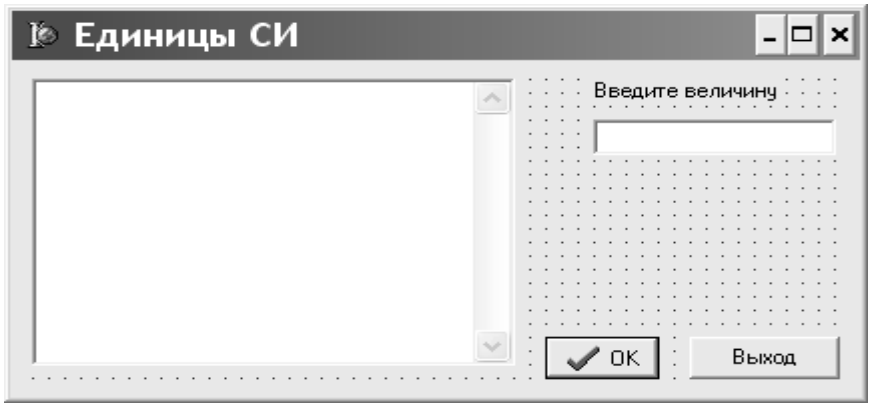
Изучаемые компоненты: Form, Edit, Label, Button, BitBtn, Memo.

Изучаемые свойства: Caption, Text, Kind, Lines, ScrollBars, Au-toSize, Color, Font, BoderWidth.

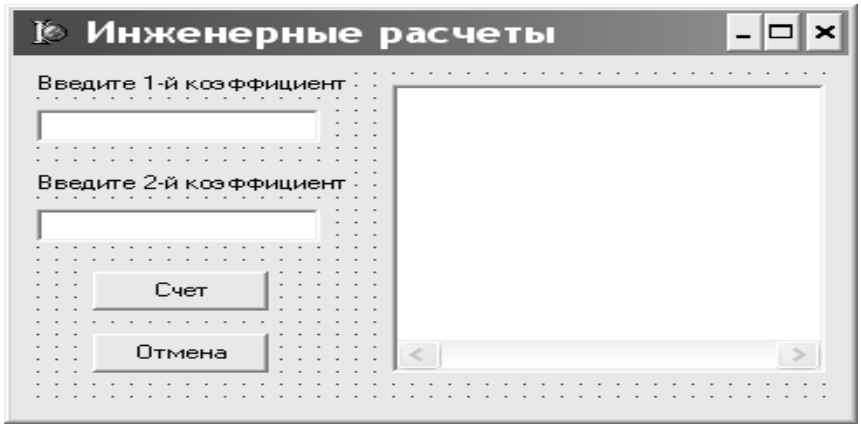
Рекомендации: Вначале нанести нужные элементы на форму, правильно расположить их и придать необходимые значения их свойствам. Затем придать следующие значения свойствам компонента Form: BoderWidth=10, AutoSize=True. В этом случае размеры формы автоматически подстроятся под расположенные на ней элементы.

Чтобы стереть наименование Memo1, появляющееся в правом верхнем углу многострочного поля, перейдите в свойство Lines, сотрите ненужную строку в редакторе. Для появления нужных полос прокрутки используйте свойство ScrollBars данного компонента.

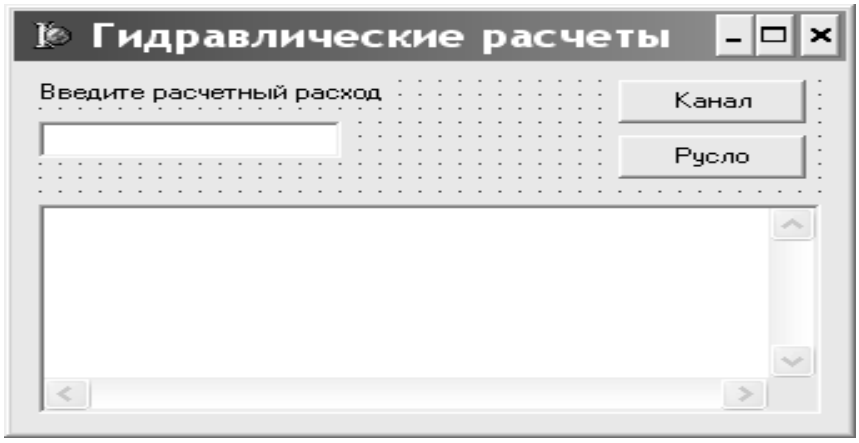
1.


Единицы СИ
Введите величину
OK
Выход

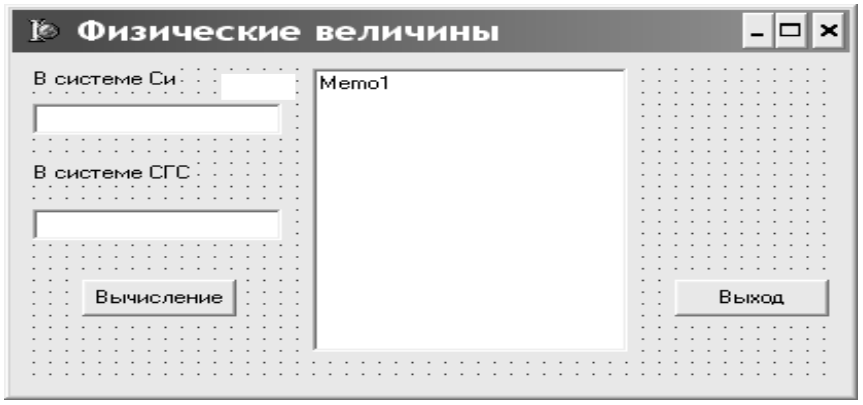
2.


Инженерные расчеты
Введите 1-й коэффициент
Введите 2-й коэффициент
Счет
Отмена

3.


Гидравлические расчеты
Введите расчетный расход
Канал
Русло

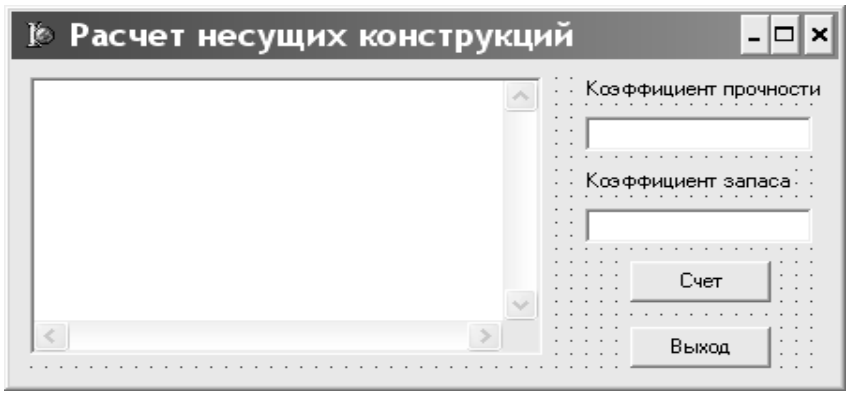
4.


Физические величины
В системе Си
Memo1
В системе СГС
Вычисление
Выход

5.


Гидравлические расчеты
Введите номер створа
Расчет
Выход

6.


Расчет несущих конструкций
Коэффициент прочности
Коэффициент запаса
Счет
Выход

7.

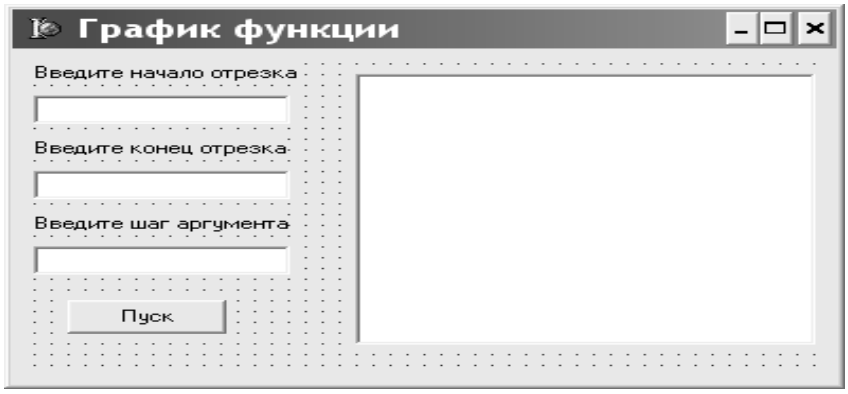
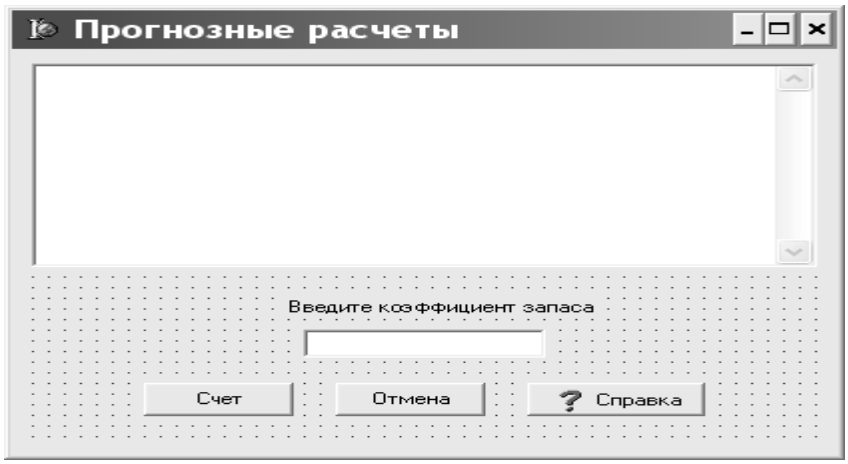
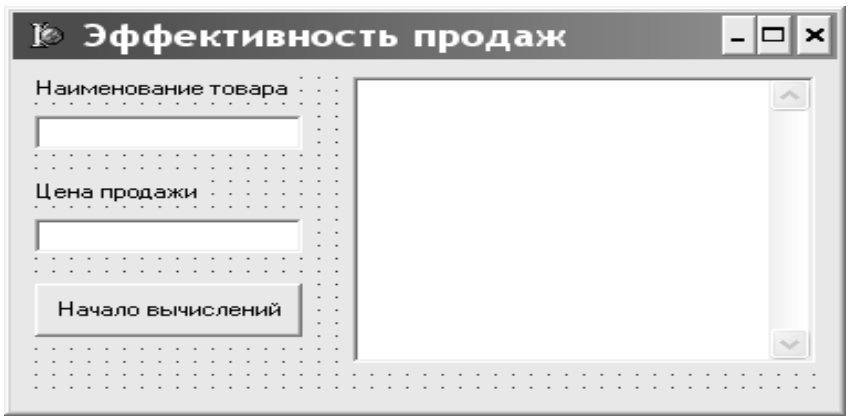


График функции
Введите начало отрезка
Введите конец отрезка
Введите шаг аргумента
Пуск

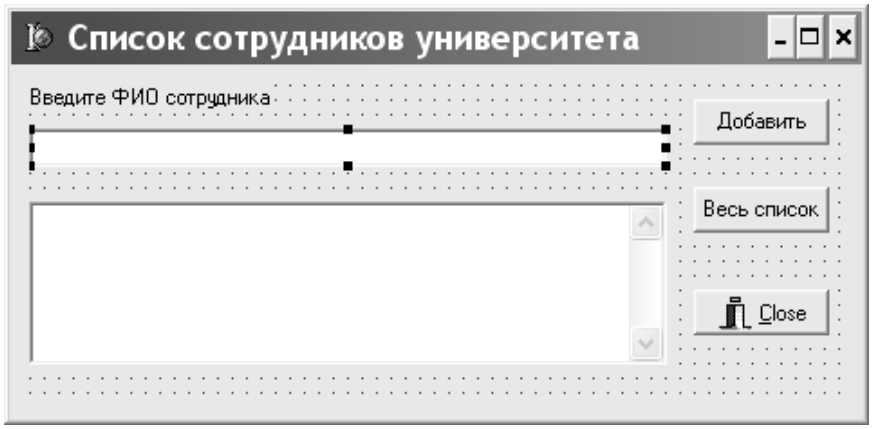
8.


Прогнозные расчеты
Введите коэффициент запаса
Счет
Отмена
Справка

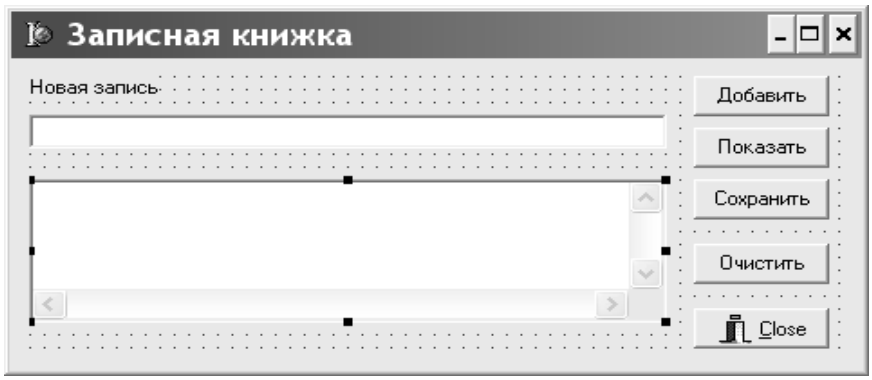
9.


Эффективность продаж
Наименование товара
Цена продажи
Начало вычислений

10.


Список сотрудников университета
Введите ФИО сотрудника
Добавить
Весь список
Close

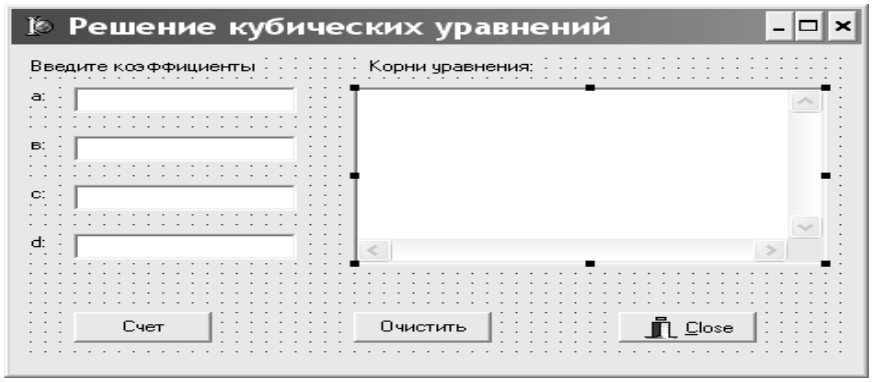
11.


Записная книжка
Новая запись
Добавить
Показать
Сохранить
Очистить
Close

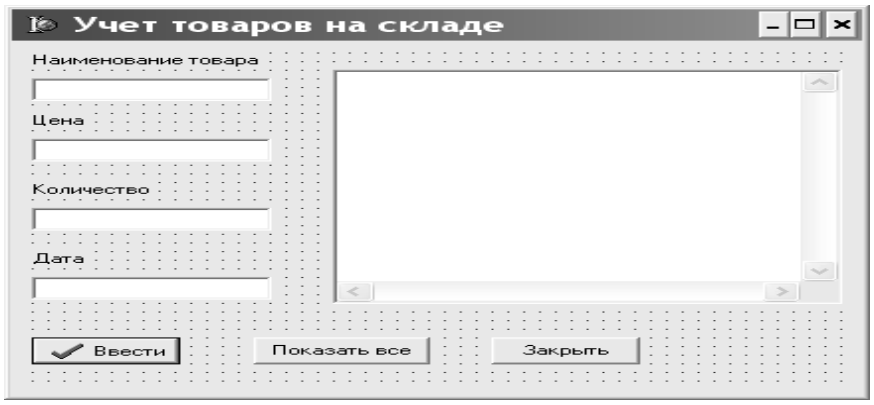
12.


Решение квадратных уравнений
Введите коэффициенты
Решение
a:
b:
c:
Счет
Очистить

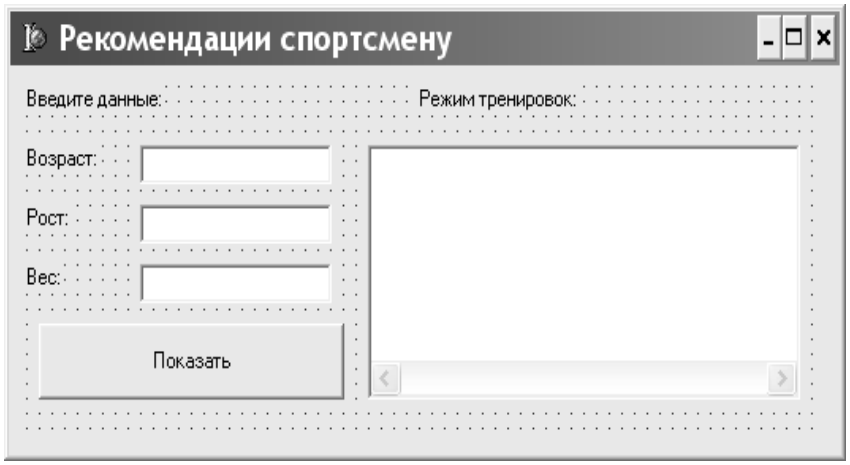
13.



14.



15.



Четыре этапа создания простой расчетной программы

При создании достаточно простых программ у начинающего программиста часто возникают элементарные ошибки, связанные отсутствием понимания процесса создания программы. Избежать этих ошибок поможет знание 4-х эта-

пов, которые необходимы при создании многих расчетных программ. Если хотя бы один из этапов пропущен – программа правильно работать не будет. Перечислим эти этапы:

1. Описание переменных. Если этот этап пропущен, то не избежать синтаксических ошибок на этапе компиляции.
2. Ввод с формы исходных данных. Если вы не ввели данные, то, естественно, ничего рассчитываться не будет. При вводе данных используются функции преобразования типа.
3. Основные расчеты. Этот этап, как правило, не пропускают, т.к. он самый существенный. Программный текст , реализующий расчетный алгоритм может быть очень длинным и в нем возможны ошибки.
4. Вывод полученных результатов на формы. Если этот этап пропущен, по после нажатия на кнопку Счет ничего не происходит. Программа как будто не работает. Программисту необходимо предусмотреть вывод результатов в размещенные на форме компоненты. При реализации этого этапа, как правило, используются функции преобразования типа, т.к. поля ввода выводы работают с символьной информацией, а расчетные результаты имеют один из числовых типов данных.

Покажем выполнение этих этапов на примере программы, рассчитывающей скорость бега спортсмена.

Скорость бега спортсмена можно рассчитать по формуле:

$$V = (s/1000)/(t/3600),$$

где s – дистанция в м, t – время в сек, V – скорость в км/час.

Окно работающего приложения следующее (см. рис. 38):

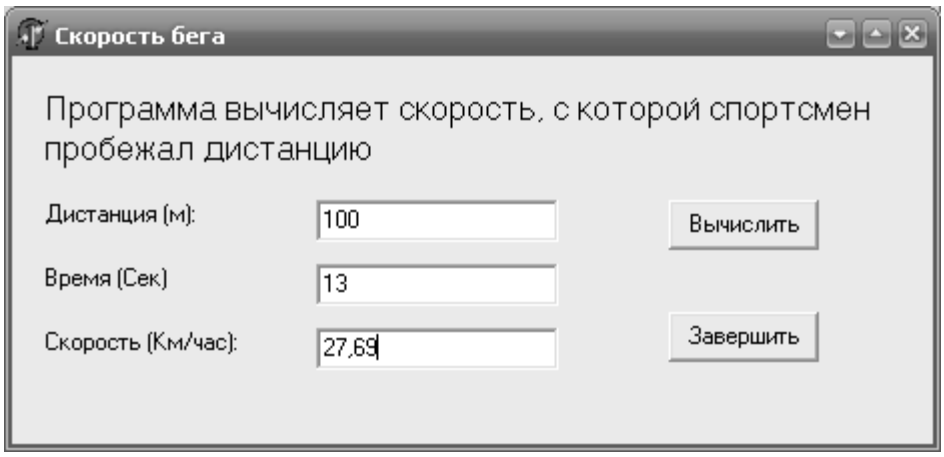


Рис. 38. **Окно программы, расчитывающей скорость бега спортсмена**

Для ввода данных используются компоненты Edit1 и Edit2, для вывода полученного результата - Edit3. На рис. 39 показан текст процедуры реализующей расчет, жирным выделены строки, которые пишет программист:

```
procedure TForm1.Button1Click(Sender: TObject);
var dist : integer;
    t: real;
    v: real;

begin

dist:=StrToInt(Edit1.Text);
t:=strToFloat(Edit2.Text);

v:=(dist/1000)/(t/3600);

Edit3.Text:=FloatToStr(v);
end;
```

Четыре этапа создания программы

1) Описание переменных

2) Ввод исходных данных с формы

3) Основные расчеты

4) Вывод результатов на форму

Рис. 39. **Четыре этапа создания простой расчетной программы**

Функция 2-х переменных

Задание: создать программу, которая бы позволяла вычислять значение функции от двух переменных.

Изучаемые вопросы: Оператор присваивания, описания локальных переменных и констант, стандартные функции.

Проект формы

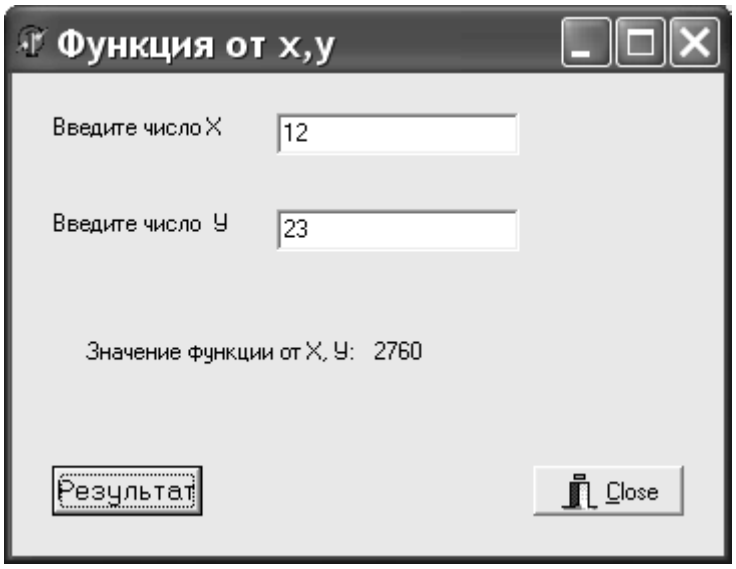


Рис. 40. **Окно программы расчета функции 2-х переменных**

Порядок выполнения: Нанести на форму 4 объекта Label, 2 объекта Edit, 1 - BinBtn, 1 - Button. Отрегулировать размеры формы и объектов. В инспекторе объектов придать всем объектам требуемые свойства. После двойного щелчка на кнопоке Button занести в раскрывшемся редакторе кода необходимые операторы языка. Откомпилировать программу, пустить ее на выполнение, устранить ошибки.

Таблица 12

Свойства компонентов.

Объект	Свойство	Значение свойства
Form1	Caption	Функция от x и y
Label1	Caption	Введите число X
Label2	Caption	Введите число У
Label3	Caption	Значении функции от X,У
Label4	Caption	
Button1	Caption	Результат
BinBtn	Kind	bkClose

Текст программы:

```
unit MultiUnit;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    Edit1: TEdit;
    Edit2: TEdit;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var x,y,pro: real;
const ub=10;
begin
```

```
 x:=StrToFloat(edit1.text);
 y:=StrToFloat(edit2.text);
 pro:=ub*x*y;
 Label4.Caption:= FloatToStr(pro);
end;
end.
```

Варианты заданий для самостоятельного решения:

1. z = 10xy;
2. g =sin(ab)+7,2ab;
3. b=x-1,5√xy;
4. h=(a+c)+ec+c;
5. y=a√c + lna + c;
6. y=c+√(c+a);
7. c=sin2(t+a)/t;
8. c=(arcsinxt + t)/xt;
9. g=ex+t√(x+t);
10. y=(lna+g)/ √(a+g);
11. y=(0,8√(sinx+cos2x))/(g+1);
12. z=√((x+c)/sinx);
13. g=√(|a|+|b|)/sin(a+b);
14. g=p2-arcsin(x+lnx);
15. x=(b2+g)2/(cosb-√(bg)).

Расчет площади земельного участка

Земельный участок имеет форму криволинейной трапеции и расположен вдоль шоссе. Участок может располагаться в любом месте с 165 до 400 км. Шоссе. С трех сторон он ограничен прямыми линиями, а с третьей кривой площадь его описывается интегралом (основание трапеции для примера заключено между 200-м и 210-м км. шоссе):

$$\int_{200}^{210} \frac{x^{6.5}}{200 * x^{Ln(x)}} (-Cos(\frac{x}{100})) dx ;$$

Требуется составить программу в Delphi, которая приближенно рассчитывает его площадь по методу левых прямоугольников и выдает схему участка, в качестве исходных данных вводить:

Xn – начало основания трапеции, км,

Xк – конец основания трапеции, км,

N – число частей, на которые разбивается основание трапеции.

Методические указания: Для построения графика использовать компонент Chart, выбрать тип диаграммы Area, при настройке осей диаграммы на вкладке Axis установить минимальное значение левой оси – 0, максимальное – автоматическое определение (см. рис. 41).

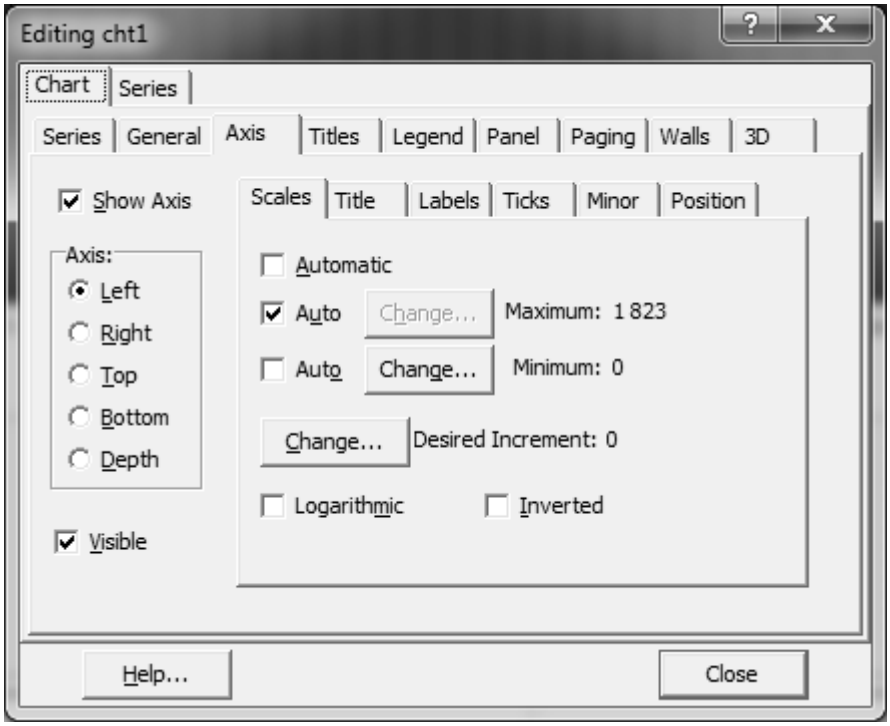


Рис. 41. **Установки по форматированию ось Y диаграммы**

Для возведения в степень переменных использовать функцию **power**, для этого подключить библиотеку **Math**, для чего вставить ее в конец списка uses:

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, TeEngine, Series, StdCtrls, Buttons, ExtCtrls, TeeProcs, Chart, **Math**;

Для проверки условия, что участок расположен между 165 и 400 км. шоссе, использовать логический оператор IF.

Ниже приводится окно программы в работе и ее текст:

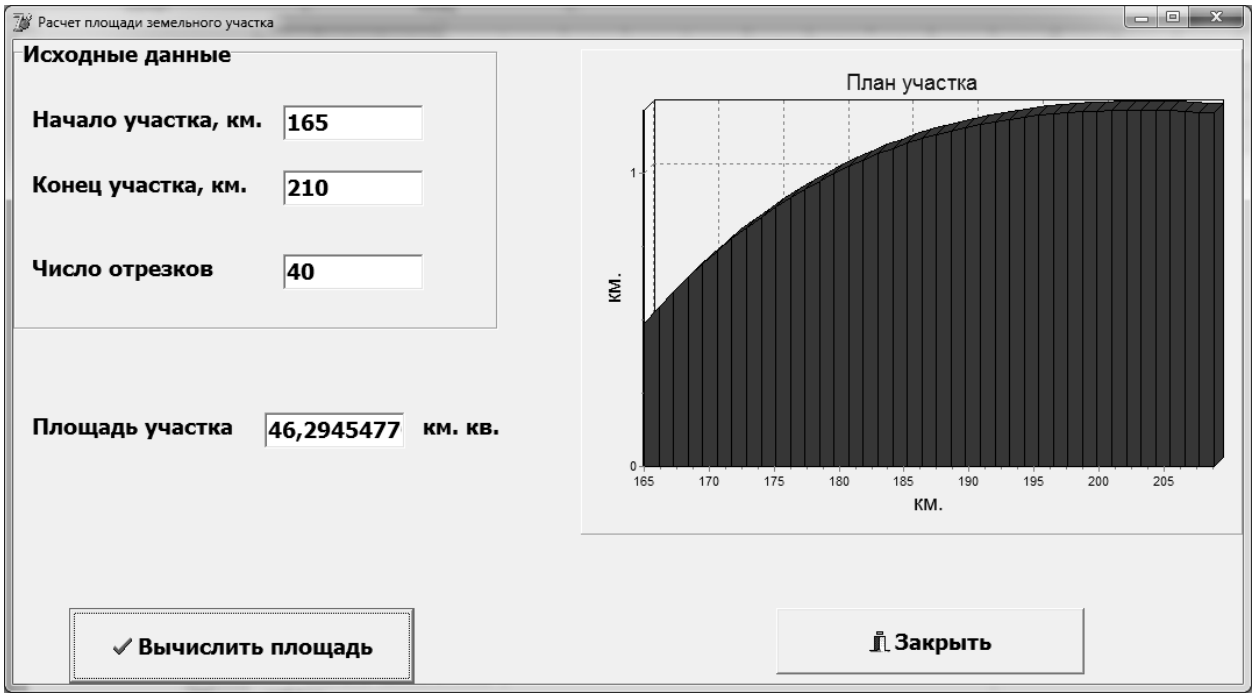


Рис. 41. **Окно программы расчета площади земельного участка**

Текст программы:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, TeEngine, Series, StdCtrls, Buttons, ExtCtrls, TeeProcs, Chart, Math;
type
  TForm1 = class(TForm)
    grp1: TGroupBox;
    label2: TLabel;
```

```
    Edit1: TEdit;
    label3: TLabel;
    Edit2: TEdit;
    label4: TLabel;
    Edit3: TEdit;
    cht1: TChart;
    label5: TLabel;
    Edit4: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Series1: TLineSeries;
    txt1: TStaticText;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.BitBtn1Click(Sender: TObject);
var x,y,s,xn,xk,hx:Real;
i,n:Byte;
begin
  xn:=StrToFloat(Edit1.text);
  xk:=StrToFloat(Edit2.text);
  n:=StrToInt(Edit3.text);
  if (xn<165) or (xk>400) or (xn>=xk) then
```

```
  begin
    ShowMessage('Недопустимые границы участка, повторите ввод!');
    Exit;
  end;
  hx:=(xk-xn)/n;
  Series1.Clear;
  x:=xn;
  s:=0;
  for i:=1 to n do
  begin
    y:=Power(x,6.5)*(-cos(x/100))/(200*power(x,Ln(x)));
    Series1.AddXY(x,y,'',clRed);
    s:=s+hx*y;
    x:=x+hx;
  end;
  Edit4.Text:=FloatToStr(s);
end;
end.
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Анисимов В.В. Решение инженерных задач с использованием MS Excel: Методические указания для студентов заочных форм обучения / В.В. Анисимов. Хабаровск: Изд-во ДВГУПС, 2003 – 29с.
2. Долженков В.А., Стученков А.Б. Microsoft Office Excel 2010 / В.А. Долженков, А.Б. Стученков. СПб: «БХВ-Петербург», 2014 - 792 с.
3. Культин Н.Б. Основы программирования в Delphi 7 / Н.Б. Культин. СПб.: «БХВ-Петербург», 2003 - 598 с.
4. Новиков С.П. Практика инженерно-технических расчетов в среде MS Excel / С.П. Новиков. Брянск: БФ МИИТ, 2012 – 20.с.
5. Соколов А.Л. Сборник заданий по Delphi / А.Л Соколов. М.: МГУП, 2008 – 118 с.

СОДЕРЖАНИЕ

Учебно-методическое издание

Соколов Андрей Львович

Информатика

Учебно-методическое пособие

Обложка художника
Компьютерная верстка
Корректор

fgnu@rosinformagrotech.ru

Подписано в печать Формат 60×84 1/16
Бумага писчая Гарнитура шрифта «Times New Roman» Печать офсетная
Печ. л. Тираж экз. Изд. Заказ Тип заказа

Отпечатано в типографии ФГБНУ «Росинформагротех»,
141261, пос. Правдинский Московской обл., ул. Лесная, 60