Ю.В. Калмыков

embarcadero

Учебно методическое пособие

по обучению программированию на языке **Delphi** для школ

При поддержке лицея 1580 при МГТУ им. Баумана

```
procedure TForm1.Button1Click(Sender: TOb
var Memo: TMemo;
begin
  Memo:=TMemo.Create(Form1);
  Memo.Parent:=Form1;
  Memo.Left:=50;
  Memo.Top:=50;
  Memo.Width:=250;
  Memo.Height:=100;
  Memo.Text:='Здравствуй,мир!'
end;
```

Москва 2014

Оглавление

Об авторе

Калмыков Ю.В., специалист в области программирования, автор научных статей и учебных пособий, среди которых «Учебно-методическое пособие по обучению программированию на языке Delphi для школ». Представленное пособие является результатом 25-летнего педагогического опыта работы в качестве преподавателя кафедры «Информатики и процессов управления» МИФИ и СУНЦ-1 МГТУ им. Н.Э. Баумана; учителя лицея №1511 при МИФИ, а затем лицея №1580 при МГТУ им. Н.Э. Баумана; эксперта ОГЭ и ЕГЭ по информатике; председателя жюри Московской олимпиады школьников по информатике (1995-2003 гг.); руководителя команды учащихся города Москвы, успешно выступавшей на заключительных этапах Всероссийской олимпиады школьников по информатике (1995-2003 гг.).

Занятие 1. Знакомство с Delphi. Простые компоненты.

Развитие персональных компьютеров привело к внедрению многозадачных, многопользовательских систем. Например, операционная система Windows. Наряду с этим создание нового программного обеспечения стало существенно сложнее. Для облегчения работы по взаимодействию с операционной системой, сокращения сроков написания программ и повышения качества программ были разработаны визуальные среды, являющиеся системами быстрой разработки приложений.

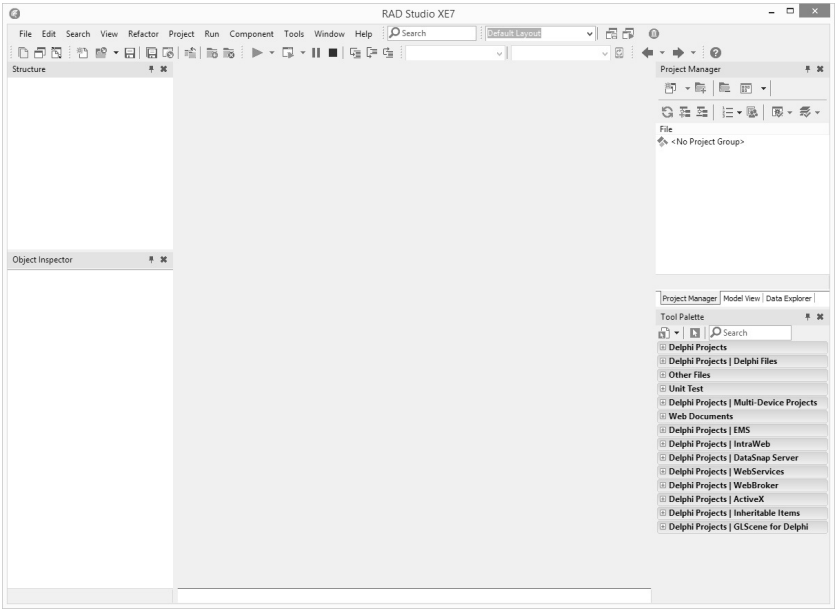
Визуальное программирование представляет собой процесс создания приложений, при котором можно одновременно конструировать, изменять, отлаживать приложение, используя интегрированную среду разработки. Иначе говоря, визуальное программирование – это единство двух взаимосвязанных процессов: наглядного конструирования типового окна приложения и написания кода программы.

В 1996 году был выпущен компилятор нового поколения – Delphi. Прежде всего, это мощный компилятор языка Паскаль, дополненного рядом новых существенных возможностей для создания приложений в среде Windows.

Delphi – это система программирования очень высокого уровня. Она берет на себя значительную часть работы по управлению компьютером. Delphi – это среда, в которой интегрирован набор инструментов для разработки готового приложения, это средство быстрой разработки приложения. В основе идеологии Delphi лежат технологии визуального проектирования и событийного программирования, применение которых позволяет существенно сократить время разработки и процесс созданий приложений.

Технология визуального проектирования позволяет пользователю оформлять будущую программу (приложение) и видеть результаты своей работы ещё до запуска самой программы. А также, например, для задания каких-нибудь свойств элементу разрабатываемого приложения не обязательно писать массивные строки программы, достаточно изменить свойство в нужном окне. Это изменение автоматически дополнит или модифицирует код программы.

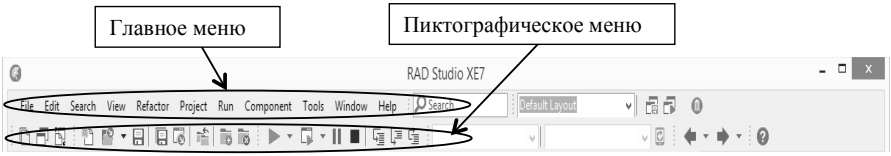
Запустим Delphi.



Интегрированная среда Delphi представляет собой многооконную систему, определяемую настройками пользовательского интерфейса.

При любых действиях программиста в каком-либо из окон, Delphi автоматически вносит изменения в код программы.

Главное окно Delphi содержит главное меню, пиктографическое меню.

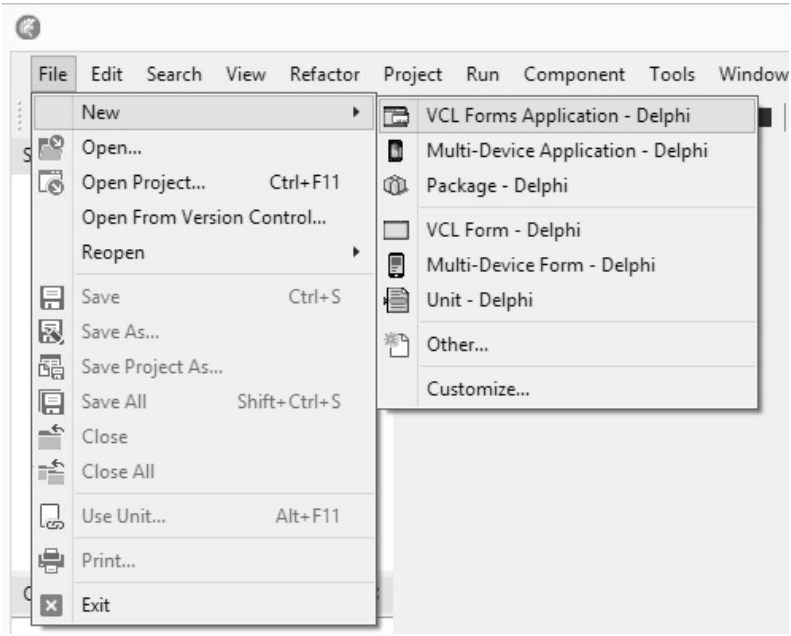


Главное меню содержит все необходимые средства для управления проектом.

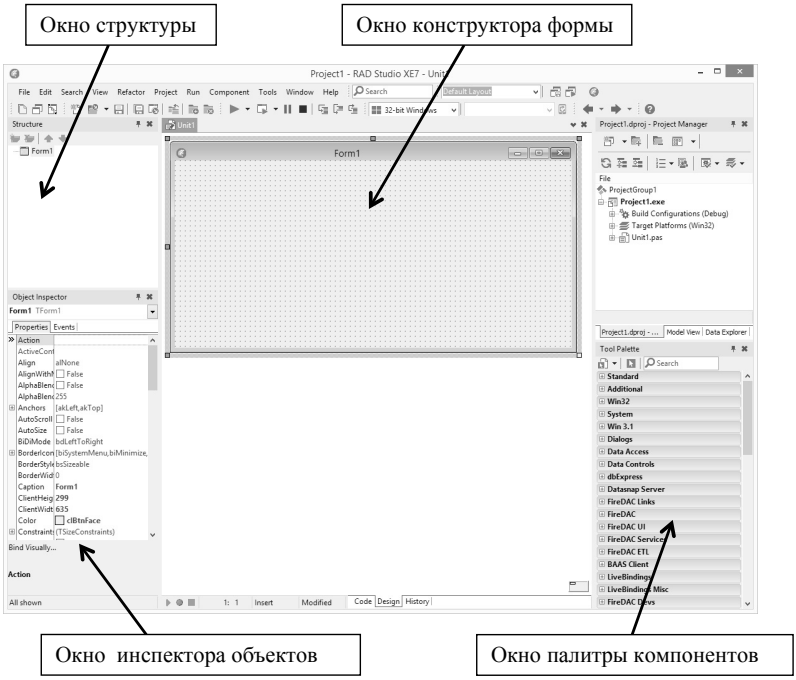
Пиктографическое меню содержит кнопки быстрого доступа к наиболее часто вызываемым опциям главного меню.

Например, для того, чтобы запустить программу на выполнение мы должны выбрать в главном меню опцию ***Run*** – ***Run***, или нажать клавишу ***F9***, или выбрать в пиктографическом меню зелёный треугольник.

Чтобы начать писать новую программу необходимо в меню File выбрать пункт New и в открывшемся списке выбрать строчку VLC Forms Application – Delphi.



Комбинация окон Delphi примет следующий вид:



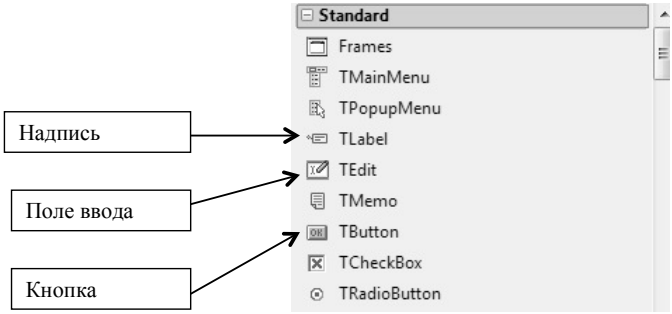
Разберёмся с появившимися окнами.

Delphi – среда объектно-ориентированного программирования. Под объектом понимается целостный, неделимый элемент, имеющий свои свойства (характеристики или отличительные признаки) и варианты действий. Сформированный объект можно переносить из программы в программу. В Delphi включены сотни готовых объектов, которые представлены в ***Окне Tool Palette (Палитра компонентов)***. Эти объекты (компоненты) сгруппированы по вкладкам).

К числу основных вкладок Палитры компонентов можно отнести следующие:

1. Standart (Стандартная)
2. Additional (Дополнительная)
3. System (Доступ к системным функциям)
4. Data Access (Работа с информацией из баз данных)
5. Data Controls (Создание элементов управления данными)
6. Dialogs (Создание стандартных диалоговых окон)

Пока мы будем работать с вкладкой ***Standart*** в палитре компонентов, в которой сосредоточены стандартные интерфейсные элементы, без которых не может обойтись практически ни одна программа.



Окно формы – проект будущего приложения. Как правило, визуальное программирование сводится к выбору нужного компонента и размещению его на поле формы.

Delphi при запуске автоматически предлагает пользователю новый проект (так называется в Delphi разрабатываемое приложение), открывая при этом пустое (незаполненное) окно (форму) под названием Form1. Конечно, данная форма содержит основные элементы окна Windows: заголовок ***Form1***, кнопки минимизации и закрытия окна, изменения размеров окна и кнопку вызова системного меню окна. Это основное окно нашей программы. На него мы будем переносить элементы из Палитры компонентов.

В Delphi поля ввода (редактирования), командные кнопки и прочие элементы управления, находящиеся на форме называются компонентами (компонентами формы). В программе форма и компоненты рассматриваются как объекты. Этим можно объяснить, что окно, в котором находятся свойства компонентов, называется ***Object Inspector (Инспектор объектов).***

Окно ***Инспектора объектов*** содержит две вкладки. На первой ***Properties (Свойства)*** постоянно отображаются все доступные свойства выбранного компонента. В левой колонке содержится список, в правой – текущие значения по умолчанию. На второй вкладке, ***Events (События)***, возможные обработчики событий для выбранного компонента. В левой колонке – названия, в правой – соответствующие свойства или процедуры.

В ***окне Структуры (Structure)*** отображаются все объекты (компоненты), которые мы размещаем на форме.

Завершение работы приложения

- Нажатие кнопки закрытия экрана
- Меню ***Запуск*** - Команда ***Остановить (Ctrl+F2)***

Сохранение проекта

Любая программа Delphi состоит из большого количества различных файлов - это файл проекта, одного или нескольких модулей и т.д. Файл проекта формируется автоматически средой Delphi и не предназначен для редактирования. Поэтому он имеет своё расширение (*.dpr) и не отображается в окне кода программы. Модули – это коды программы. Каждый проект есть смысл сохранять в отдельную папку.

Сохранять проект надо с помощью команды *Сохранить все*.

При сохранении под каждую задачу (проект) необходимо создавать новую папку.

Папке и файлам проекта дать соответствующие имена. Не нужно соглашаться с именами по умолчанию. Имена файлов состоят из английских букв, цифры допустимы со второго символа. Остальные символы недопустимы.

Файлам ***Project1.dpr*** и ***Unit1.pas*** нужно будет дать свои имена (они разные), остальным файлам имена будут присвоены по умолчанию.

При переименовании файлов ***Project1.dpr*** и ***Unit1.pas*** надо следовать стандарту стилевого оформления исходного кода DELPHI, согласно которому все файлы должны иметь ***префикс*** XXX, например ***prjMyTask1.dpr*** **и** ***untMyTask1.pas***. (XXX – это буквы, соответствующие сокращенному названию того, что Вы сохраняете: ***Project1 - prj и Unit1 - unt***). Данное правило должно использоваться и для переименования простых компонентов.

А теперь поговорим о некоторых простых компонентах, которые есть в среде Delphi.

Рассматриваемые компоненты считаются простыми не потому, что их легче использовать, чем другие, а потому, что при создании графического интерфейса они часто применяются в довольно сложных сочетаниях с другими компонентами. Здесь рассматриваются следующие простые компоненты: *форма*, *надпись*, *поле ввода* и *кнопка*.

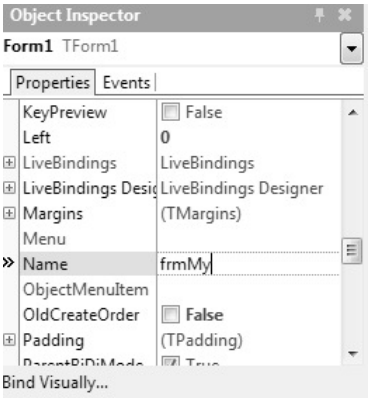
Форма

Из перечисленных компонентов только **форма** является объектом, отсутствующим на палитре компонентов. Форма создаётся сама при создании нового приложения. Если в приложении необходимо использовать несколько форм, то можно добавить к проекту формы дополнительно, но мы пока этого делать не будем. Работая в Delphi, представьте себе, что вы художник, а форма — это ваш холст. Вы рисуете пользовательский интерфейс приложения путём размещения на форме компонентов. Затем, изменяя свойства компонентов, вы изменяете их внешний вид. После этого вы разрабатываете средства взаимодействия компонентов с пользователем. Взаимодействие компонентов формы с пользователем означает некоторую реакцию компонентов на действия пользователя.

Свойства компонентов – это их характеристики, например, имя, заголовок, размер или цвет компонента, размер или цвет надписи на компоненте и т.д.

Рассмотрим некоторые свойства компонента ***Форма***. Многие свойства одинаковы у разных компонентов, но есть свойства, присущие только некоторым или одному компоненту.

Первое свойство, которое мы рассмотрим – это свойство ***Name (Имя).*** Это свойство есть обязательно у всех компонентов, так как обращаться к компоненту в программе приходится по имени. Будем при назначении имён обязательно использовать трёхбуквенный префикс, который говорит о типе компонента. Для компонента ***Форма*** будем использовать префикс ***frm*** (сокращение от ***Form***). Так если мы хотим назвать форму ***My*** (моя), то мы должны свойству ***Name*** присвоить значение ***frmMy***.



Рассмотрим три простых компонента, которые находятся во вкладке ***Standard*** (Стандартная) палитры компонентов.

Надпись

Компонент ***Надпись*** (*TLabel*) используется для вывода на форму текста, который пользователь не может изменить непосредственно (естественно, в программе может быть предусмотрено изменение надписи "изнутри" программы). Рассмотрим методику работы с надписями на конкретном примере. Выполните ряд действий.

1. Создайте новый проект.

2. Поместите надпись на форму. Для этого дважды щёлкните кнопкой мыши на пиктограмме надписи на палитре компонентов. Другой способ размещения надписи на форме — однократно щёлкнуть кнопкой мыши на пиктограмме надписи на палитре компонентов, а затем однократно щёлкнуть в произвольном месте формы. Такой способ удобен тем, что надпись в этом случае размещается там, где вы хотите. Чтобы удалить надпись с формы, нужно выделить её (щёлкнуть на ней кнопкой мыши, при этом она выделяется черными квадратиками) и нажать клавишу <***Delete***>. Чтобы отменить выделение, нужно щёлкнуть кнопкой мыши в любом месте за пределами надписи. Поэкспериментируйте с размещением и удалением надписей.

3. Переместите надпись в другое место на форме методом перетаскивания. Для этого установите указатель мыши на надписи, нажмите кнопку мыши и, удерживая её нажатой, передвиньте надпись в другое место. Когда надпись займёт нужное положение, отпустите кнопку мыши.

4. Измените свойство надписи ***Name*** **(**имя надписи) на ***IblMyLabel*** (по умолчанию она называлась *Label1*). Для этого в инспекторе объектов щёлкните на свойстве ***Name*** и введите ***IblMyLabel***. Убедитесь, что вы изменяете свойство надписи, а не формы (это типичная ошибка новичков). Для этого надпись на форме должна быть выделена, а в заголовке раскрывающегося списка в верхней части инспектора объектов должно быть написано ***Label1***: ***TLabel*** (когда вы измените имя надписи, там будет написано ***IblMyLabel: TLabel***). После ввода нужного имени надписи, зафиксируйте его, нажав клавишу <***Enter***>.

5. Обратите внимание: заголовок надписи изменился на ***IblMyLabel***. Это объясняется тем, что по умолчанию заголовок надписи совпадает с её именем. Измените заголовок явно. Для этого выберите в инспекторе объектов свойство ***Caption*** (заголовок надписи), введите новый заголовок «**Это моя первая надпись!**» и нажмите клавишу <***Enter***>. Введённый текст должен появиться на форме.

6. Измените цвет фона надписи. Для этого выберите свойство ***Color*** (цвет фона), щёлкните на стрелке, выберите в раскрывшемся списке жёлтый цвет и щёлкните на нем.

7. Измените шрифт и цвет текста надписи. Для этого выберите свойство ***Font*** (шрифт) и щёлкните на трёх точках. В окне ***Font*** измените шрифт на ***Arial***, стиль на ***Bold Italic*** (полужирный курсив), а размер — на 20. В раскрывающемся списке выберите красный цвет и щёлкните на кнопке ***OK***.

8. Добавьте на форму ещё одну надпись. На этот раз воспользуйтесь другим методом — щёлкните на пиктограмме *надписи* на палитре компонентов, переместите указатель мыши в произвольное место формы и ещё раз щёлкните кнопкой мыши. При этом на форме в указанном вами месте должна появиться новая надпись.

9. Измените свойство ***Name*** новой надписи на ***IblAnother***, а свойство ***Caption*** — на «Другая надпись».

10. Теперь выделите форму. Это можно сделать двумя способами: щёлкнуть в любом месте формы за пределами надписей или выбрать ***Form1*** в окне ***Структура*** (***Structure***). Если форма видна, то первый способ, конечно, удобнее, однако если в проекте есть много форм, причём нужная форма закрыта другими окнами, то более удобен второй способ.

11. Измените свойства формы: свойству **Name** задайте значение ***frmLabelExample***, а свойству ***Caption*** — **«Пример надписи».**

12. Итак, сами того не заметив, вы создали простое приложение, которое, правда, пока что ничего не делает. Выполните его. Это можно сделать одним из трёх способов: щёлкнув на пиктограмме ***Запустить*** (Зелёный треугольник), выбрав в главном меню команду ***Run=> Run*** или нажав клавишу ***<F9>.***

13. Щёлкнув на крестике в верхнем правом углу формы, завершите приложение.

Поле ввода

В компоненте ***Поле ввода*** (*TEdit*) хранится текст, который можно помещать в данный компонент, как во время разработки, так и во время выполнения. Текст, видимый в поле ввода, находится в свойстве ***Text***. Свойство ***MaxLength*** определяет максимальное количество символов в поле ввода. Если значение свойства ***MaxLength*** равно ***0***, то количество символов ничем не ограничено. С помощью свойства ***Font*** можно устанавливать шрифт текста. Если свойство ***Readonly*** (только чтение) установить в ***True*** (истина), то во время выполнения программы пользователь не сможет изменять текст поля ввода. Чтобы лучше усвоить приёмы работы с полями ввода, выполните ряд действий.

1. Создайте новый проект.

2. Разместите ***поле ввода*** на форме. Как и в случае надписи, это можно сделать одним из двух способов: дважды щёлкнуть на пиктограмме поля ввода на палитре компонентов или щёлкнуть на ней один раз, а затем щёлкнуть в произвольном месте формы.

3. Измените размер поля ввода. Для этого установите указатель мыши на одном из черных квадратиков и, удерживая кнопку мыши нажатой, переместите чёрный квадратик (а с ним и границу поля ввода) в нужном направлении. Установив необходимый размер, отпустите кнопку мыши. Если чёрных квадратиков вокруг поля ввода нет, значит, оно не выделено. В этом случае выделите поле ввода, щёлкнув на нем кнопкой мыши.

4. Переместите поле ввода в другое место методом **перетаскивания.** Для этого установите указатель мыши на поле ввода, нажмите кнопку мыши и, удерживая её нажатой, передвиньте поле ввода в новое место. Когда поле ввода займёт нужное положение, отпустите кнопку мыши.

5. Установите значение свойства ***Name*** в ***edtMyText***. Для этого в инспекторе объектов щёлкните на свойстве ***Name*** и введите ***edtMyText.*** Как и в случае с надписью, убедитесь, что вы изменяете свойство поля ввода, а не формы. Для этого в окне ***Структура*** (***Structure***) должно быть написано ***Edit1*: *TEdit*** (после изменения свойства ***Name*** здесь будет написано ***edtMyText*: *TEdit***).

6. Выберите в инспекторе объектов свойство ***Text*** и введите его новое значение: *Это элемент управления "поле ввода"*. Нажав клавишу <***Enter***>, зафиксируйте введённый текст. Обратите внимание: во время ввода изменяется текст в поле ввода на форме.

7. Измените цвет текста в поле ввода на синий. Для этого в инспекторе объектов щёлкните на значке «+» рядом со свойством ***Font***. При этом появляется список свойств объекта ***Font.*** Выберите свойство ***Color*** и щёлкните на стрелке, расположенной в этом поле. При этом раскрывается список доступных цветов. Найдите в нем синий цвет и щёлкните на нем.

8. Выделите форму. Это можно сделать одним из двух способов: щёлкнув в любом месте формы за пределами поля ввода или выбрав имя формы в окне ***Структура*** (***Structure***). Измените свойство формы ***Name*** на ***frmEditBoxExample***, а свойство ***Caption*** — на **Пример поля ввода**.

9. Нажав клавишу <***F9***>, запустите разработанную программу на выполнение. Поэкспериментируйте с полем ввода. Введите в него какой-либо текст.

10. Для завершения программы щёлкните на крестике в правом верхнем углу формы.

11. Установите значение свойства ***Readonly*** в ***True***.

12. Нажав клавишу <***F9***>, запустите программу на выполнение ещё раз. Попытайтесь изменить содержимое поля ввода: как видите, теперь изменить его содержимое во время выполнения нельзя. Возможно, вы удивитесь: зачем может понадобиться поле ввода, в которое нельзя ничего ввести? Однако в дальнейшем вы увидите, что это довольно полезное средство, так как значение свойства ***Readonly*** можно менять из программы, запрещая или разрешая, таким образом, пользователю вводить данные.

13. Щёлкнув на крестике в правом верхнем углу формы, завершите выполнение программы.

Кнопка

Обычно с помощью компонента **кнопка** (***TButton***) пользователь инициирует выполнение какого-либо фрагмента кода или целой программы. Другими словами, если щёлкнуть на элементе управления ***TButton***, то программа выполняет определённое действие. При этом кнопка принимает такой вид, будто она вдавлена.

Кнопкам можно присваивать комбинации клавиш быстрого вызова. Во время выполнения нажатие такой комбинации клавиш эквивалентно щелчку на кнопке левой кнопкой мыши. Выполните ряд действий:

1. Создайте новый проект.

2. В инспекторе объектов измените значение свойства формы ***Name*** на ***frmButtonExample***, а свойства ***Caption*** — на *Пример кнопки*.

3. Поместите кнопку на форму. Для этого дважды щёлкните на пиктограмме кнопки на палитре компонентов или однократно щёлкните на её пиктограмме, а затем в любом месте формы.

4. Измените значение свойства ***Name*** кнопки на ***btnMyButton***. Для этого в инспекторе объектов щёлкните на свойстве ***Name*** и введите ***btnMyButton***. Убедитесь, что вы изменили свойство кнопки, а не формы. В окне ***Структура*** (***Structure***) должно быть написано ***Button1: TButton***, а после изменения имени — ***btnMyButton: TButton***.

5. Измените значение свойства ***Caption*** кнопки на ***&Run***. Обратите внимание: в заголовке кнопки буква, перед которой стоит символ &, оказалась подчёркнутой. В данном случае это буква ***R***.

6. Измените размер и положение кнопки.

7. Нажав клавишу <***F9***>, запустите программу на выполнение.

8. Щёлкните на кнопке. При этом кнопка принимает такой вид, будто она "вдавлена".

9. Активизируйте кнопку, нажав клавишу <R>. Как видите, при активизации клавишами быстрого вызова кнопка не принимает вид вдавленной. Пока ещё с кнопкой не связан какой-либо фрагмент кода, поэтому никакой реакции кнопки не видно. Тем не менее, можете поверить, она активизируется.

10. Щёлкнув на крестике в правом верхнем углу формы, завершите работу программы.

Заголовок кнопки *btnMyButton* выглядит как **R**un, а не **Run**. Символ ***&***, расположенный перед любой буквой значения свойства *Caption*, присваивает кнопке комбинацию клавиш быстрого вызова. В заголовке кнопки буква, перед которой стоит символ ***&*** подчеркнута. Это сообщает пользователю о том, что с кнопкой связана клавиша быстрого вызова. Во время выполнения пользователь может активизировать кнопку с помощью клавиатуры. Для этого нужно нажать клавишу с подчёркнутой буквой.

В данном примере клавиатура должна быть переключена на английский язык (С русскими буквами клавиатура должна быть переключена на русский язык).

Что делать, если в заголовке кнопки должен присутствовать символ ***&***? Ведь если поместить его в заголовок, то он сделает следующую букву подчёркнутой, а сам виден не будет. Чтобы решить эту проблему, используется следующее правило: символ ***&*** отображается в заголовке кнопки, если в свойстве ***Caption*** записаны два стоящих подряд символа — ***&&***. Например, чтобы заголовок кнопки имел вид *This & That*, в свойстве ***Caption*** должно быть написано *This && That*. При этом никакая комбинация клавиш быстрого вызова кнопке не присваивается.

Как видите, с помощью среды разработки Delphi можно буквально за минуту создать простую форму (и программу). Аналогично создаются также самые сложные пользовательские интерфейсы. Теперь вы знакомы со свойствами некоторых наиболее распространённых компонентов Delphi — формами, надписями, полями ввода, областями просмотра и кнопками. Чтобы закрепить полученные знания, поэкспериментируйте с этими компонентами. Попробуйте изменять другие их свойства.

Занятие 2. Обработка события нажатия на кнопку

Визуальные компоненты способны генерировать и обрабатывать достаточно большое число (несколько десятков) событий различного вида: щелчок мыши, нажатие на кнопку, нажатие клавиш клавиатуры, открытие окна.

Когда пользователь нажимает на любую кнопку, Windows посылает сообщение приложению: *«По такой-то кнопке щёлкнули»*. Если пользователь использовал это сообщение и написал реакцию на него, то программа выполнит это.

Реакции на событие - это результат произошедшего системного события.

Реакцию на событие назначают программно, указывая список действий, которые необходимо произвести, в **окне редактора исходного кода.** (Чтобы заставить программу реагировать на щелчок, необходимо написать фрагмент программы, который называется *обработчиком события*).

Окно редактора исходного кода.

На момент запуска окно имеет заголовок ***Unit1***.

В редакторе кода могут быть открыты сразу несколько файлов. Каждый открытый файл размещается на отдельной странице, а его название отображается в верхней части (на вкладке). Если в программе будут три окна, то они в процессе работы будут взаимодействовать с тремя модулями (Unit). Все эти модули и отображаются в редакторе кода.

Простые события содержат только источник события, на который указывает параметр Sender в *Процедуре обработки события*. В сложных событиях в *Процедуре обработки события* требуются дополнительные параметры (например, передача координат указателя мыши).

При выборе управляющего элемента возникает событие ***OnClick***, которое также называют *Событие нажатия*. Обычно оно возникает при щелчке мышью на компоненте. При разработке приложений событие ***OnClick*** является одним из наиболее часто используемых. (Для некоторых компонентов событие ***OnClick*** может возникать и при других способах нажатия на управляющий элемент).

В качестве примера напишем реакцию нажатия на командную кнопку **btnMy** расположенную на форме **frmMy**.

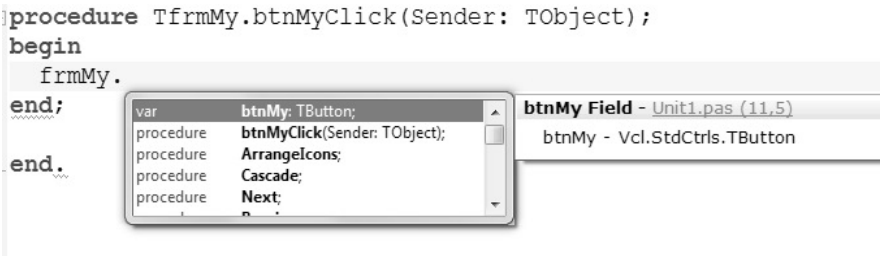
Дважды щёлкаем по этой кнопке.

Delphi автоматически подготавливает редактор кода для минимизации ввода данных с клавиатуры, набирая за нас заголовок процедуры реакции нажатия на кнопку.

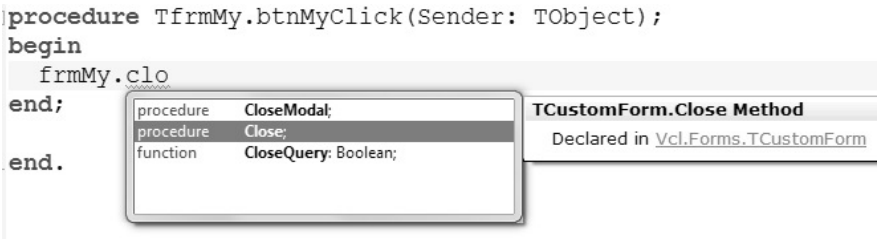
```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
end;
```

Курсор будет находиться между словами **begin** и **end.** Именно здесь мы и напишем команду программного закрытия окна формы (аналогично действию системной кнопки в заголовке этого окна). Поскольку это окно единственное и главное, то будет закрыта и вся программа.

Набираем вручную **frmMy** и ставим точку. Через несколько мгновений Delphi выводит для нас дополнение кода. Это список доступных свойств и методов для компонента **frmMy**.



Перемещаясь по списку можно найти соответствующее свойство или процедуру. Однако список довольно велик, поэтому для ускорения поиска можно набрать несколько символов искомого слова и количество элементов списка резко сократится.



Завершение ввода из списка дополнения кода осуществляется нажатием на клавишу Enter.

Окончательно наше описание реакции нажатия на командную кнопку **btnMy** выглядит следующим образом:

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
   frmMy.Close;
end;
```

Если нажать на командную кнопку **btnMy** после запуска работы программы, то окно формы закроется. В данном примере мы использовали метод (процедуру) Close. Метод показывает, что необходимо сделать с компонентом. Из примера видно, что формат обращения к методу следующий: сначала идёт имя компонента (свойство Name), затем ставится разделитель – точка, и после разделителя имя метода, т.е. что необходимо сделать с компонентом.

```
<Имя_компонента>.<Метод>;
```

У каждого компонента свой набор методов. Иногда некоторые методы могут встречаться у нескольких компонентов.

Приведём пример ещё одного метода – ***Clear***. Данный метод очищает содержимое и может применяться, например, к компоненту ***Edit***.

Edit1.Clear

Как мы уже говорили, компоненты имеют не только методы, но и свойства. Свойства - это характеристики компонентов. Свойство можно изменить или задать следующим образом: сначала указывается имя компонента, потом ставится разделитель – точка, после неё ставиться имя свойства, причём система даёт нам возможность выбрать необходимое нам свойство из меню, потом ставится значок присваивания и далее значение свойства.

```
<Имя_ компонента>.<Свойство>:=<Значение>;
```

Большинство свойств компонента совпадают с теми, что отображены в инспекторе объектов, но бывают и свойства, которые не отображены в нём. С ними мы познакомимся позже. А пока давайте теперь напишем программу, в которой при нажатии на командную кнопку окно формы закрасится белым цветом. В данной программе необходимо использовать свойство ***Color***.

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
   frmMy.Color:=clWhite;
end;
```

Свойства компонентов можно не только задавать, но и считывать, например, для того чтобы присвоить свойству другого компонента. Формат записи выглядит следующим образом:

```
<Имя_компонента1>.<Свойство1>:=<Имя_компонента2>.<Свойство2>;
```

Например:

```
lblMy.Caption:=edtMy.Text;
```

По данному предложению текст окна ввода ***edtMy*** будет помещён на заголовок надписи ***lblMy***.

Если необходимо выполнить несколько действий, то они отделяются в Delphi точкой с запятой.

Например:

```
procedure TfrmMy.btnMyClick(Sender: TObject);
begin
  edit1.Clear;
  edit1.Color:=clBlue;
end;
```

При нажатии на кнопку выполнится два действия – очистится поле ввода и его фон окрасится в синий цвет.

ЗАДАНИЯ

Задание 1.

Создать форму, содержащую надпись и две кнопки. Первая кнопка включает надпись, а вторая выключает.

Задание 2.

Создать форму, содержащую надпись, поле ввода и кнопку. При нажатии на кнопку текст из поля ввода переносится на надпись, а поле ввода очищается.

Задание 3.

Светофор. Создать форму, содержащую три надписи и три кнопки. Каждая кнопка включает свою надпись и выключает остальные.

Задание 4.

Создать форму, содержащую надпись и четыре группы кнопок по три штуки. Первая группа кнопок (три кнопки) меняет цвет фона надписи. Вторая группа кнопок (три кнопки) меняет цвет шрифта надписи. Третья группа кнопок (три кнопки) меняет размер шрифта надписи. И последняя группа кнопок (три кнопки) меняет название шрифта надписи.

Занятие 3. Переменные и их типы. Преобразование типов

Если нам необходимо один раз вычислить значение по формуле, причём исходные данные вполне конкретно заданы, для этого совсем не обязательно разрабатывать программу.

Но, если вычисления нужно производить достаточно часто и с различными значениями исходных данных, имеет смысл запрограммировать этот процесс.

Инструкция присваивания

Пусть величина **DEGR** задаёт число градусов, а переменная **RAD** — число радиан. Тогда формула перевода градусов в радианы на языке Delphi будет выглядеть следующим образом:

```
RAD := ( DEGR * 3.14 ) / 180 ;
```

Инструкция присваивания - основная в любом языке программирования. Эта инструкция позволяет присвоить переменной значение вычисленного выражения.

Инструкция присваивания записывается следующим образом. Слева стоит имя переменной, справа — выражение, значение которого должно быть вычислено. Разделяет левую и правую части составной символ "**:=**", который интерпретируется как один

элемент. Выполнение инструкции присваивания заключается в следующем: вычисляется результат выражения, стоящего справа, и этот результат становится значением переменной, имя которой стоит слева.

Формат инструкции присваивания:

```
<имя переменной> := <арифметическое выражение>;
```

Давайте разберём, что такое переменная. Это ячейка памяти, которая имеет своё имя. В зависимости от того, какое значение надо записать в переменную, в памяти выделяется ячейка того или иного размера. В принципе, можно было бы выделить для переменных ячейки только большого размера. Но тогда при большом количестве переменных памяти может просто не хватить.

Для того, чтобы знать какую ячейку памяти необходимо отвести под переменную, существует понятие тип переменной.

В Delphi, для решения различных задач можно использовать переменные. Для этого их нужно описать в блоке объявления, который расположен после заголовка процедуры перед словом begin, после служебного слова Var (переменная). Формат описания переменных следующий:

```
Var
   <имя переменной>: <тип переменной>;
```

Имя переменной представляет собой последовательность символов, составленную по следующим правилам:

- в имени используются только латинские буквы, либо цифры (от 0 до 9), либо значок подчёркивания;
- первый символ в имени может быть только буквой;
- количество символов в имени не ограничено.

Различия между заглавными и прописными буквами не существует.

Когда Вы раздумываете над именами переменных, то имейте в виду, что нужно избегать однобуквенных имён, кроме как для временных переменных и переменных цикла.

Переменные цикла именуются **I** и **J**. Другие случаи использования однобуквенных переменных это **S** (строка) и **R** (радиус). Однобуквенные имена должны всегда использовать символ в верхнем регистре, но лучше использовать более значимые имена. Не рекомендуется использовать переменную l (эль), потому что она похожа на 1 (единицу).

Пример:

```
procedure TfrmMy.btnMyClick(Sender: TObject);
 Var
      k : Integer;
begin
end;
```

В этом примере описана переменная ***k***, которая имеет тип ***Integer*** (целое).

Типов переменных в среде Delphi очень много. Рассмотрим только некоторые из них.

Целые числа могут быть описаны несколькими типами, представленными в таблице.

Название типа	**Диапазон изменения чисел**	**Формат представления в компьютере**
Integer	От -2147483648 до 2147483647	32-bit, число со знаком
Cardinal	От 0 до 4294967295	32-bit, положительные числа
ShortInt	От -128 до 127	8-bit, число со знаком
SmallInt	От -32768 до 32767	16-bit, число со знаком
LongInt	От -2147483648 до 2147483647	32-bit, число со знаком
Int64	От -2^{63} до $2^{63}-1$	64-bit, число со знаком
Byte	От 0 до 255	8-bit без знаковый тип
Word	От 0 до 65535	16-bit без знаковый тип
LongWord	От 0 до 4294967295	32-bit без знаковый тип

Мы, как правило, будем пользоваться типом **Integer**.

Вещественные числа также могут быть представлены несколькими типами. Мы будем пользоваться типом **Real**.

Название типа	Диапазон изменения	Кол-во значащих цифр	Занимаемый объем в байтах
Real48	От $\pm 2.9 \times 10^{-39}$ до $\pm 1.7 \times 10^{38}$	11-12	6
Real	От $\pm 5.0 \times 10^{-324}$ до $\pm 1.7 \times 10^{308}$	15-16	8
Single	От $\pm 1.5 \times 10^{-45}$ до $\pm 3.4 \times 10^{38}$	7-8	4
Double	От $\pm 5.0 \times 10^{-324}$ до $\pm 1.7 \times 10^{308}$	15-16	8
Extended	От $\pm 3.6 \times 10^{-4951}$ до $\pm 1.1 \times 10^{4932}$	19-20	10
Comp	От -2^{63} до $2^{63}-1$	19-20	8
Currency	От -922337203685477.5808 до +922337203685477.5807	19-20	8

Из математики Вы знаете нормализованную запись действительного числа, например:

$3{,}28 \cdot 10^{17}$ $1{,}4 \cdot 10^{-9}$ $-5{,}101 \cdot 10^{4}$

В Delphi эти числа запишутся следующим образом:

3.28e+17	1.4e-09	-5.101e+4
328e15	0.14e-8	-5101e+1
0.328e+18	140e-11	-510100e-1

Арифметическое выражение

Вернёмся к инструкции присваивания. Тип переменной, стоящей в левой части, должен соответствовать типу значения, полученного при вычислении арифметического выражения, стоящего в правой части.

Арифметическое выражение — это совокупность констант, переменных, функций, объединённых знаками арифметических действий и круглыми скобками, имеющая математический смысл.

Операции, используемые в арифметических выражениях:

Знак	Операция	Тип операндов	Тип результата
+	сложение	хотя бы один из операндов Real	Real
		Integer	Integer
-	вычитание	хотя бы один из операндов Real	Real
		Integer	Integer
*	умножение	хотя бы один из операндов Real	Real
		Integer	Integer
/	деление	Real, Integer	Real
Div	целочисленное деление	Integer	Integer
Mod	остаток от целочисленного деления	Integer	Integer

Например:

13 div 4 = 3 **13 mod 4 = 1**

-13 div 4 = -3 **-13 mod 4 = -1**

13 div -4 = -3 **13 mod -4 = 1**

-13 div -4 = 3	**-13 mod -4 = -1**
0 div 2 = 0	**1 mod 2 = 1**
2 div 5 = 0	**2 mod 5 = 2**

В случае если значение выражения имеет тип ***Integer***, его можно присвоить переменной типа ***Real***, но никак не наоборот.

Рассмотрим задачу сложения двух чисел. Создадим форму **frmCalc**, имеющую два окна ввода **edtA** и **edtB**, одну кнопку **btnPlus** и одну надпись **lblOtvet**.

Зададим свойство ***Caption*** для кнопки, свойство ***Text*** для полей ввода оставим пустым. Нажав два раза на кнопку, начнём писать программу обработки события.

Сначала опишем необходимые нам переменные.

```
Var
   a, b, c: Single;
```

Значений переменных a и b будем брать из полей ввода edtA и edtB. При этом необходимо помнить, что в поле ввода вводятся строки, значит, нам необходимо перевести их в вещественный тип. Для этого воспользуемся функцией **StrToFloat (строка).**

```
Begin
   a:=StrToFloat(edtA.Text); //Ввести число A
   b:=StrToFloat(edtB.Text); //Ввести число B
   c:=a+b; //Сложить два числа
   lblOtvet.Visible:=true; //Сделать видимым окно вывода
   lblOtvet.Caption:=FloatToStr(c); //Вывести ответ
end;
```

Результат сложения двух переменных положим в третью и выведем ответ в надпись, воспользовавшись функцией перевода вещественного числа в строку **FloatToStr(число)**. Для более красивой записи ответа напишем его в развёрнутой форме:

```
lblOtvet.Caption:=edtA.Text+'+'+edtB.Text+'='+FloatToStr(c);
```

При работе с функциями **StrToFloat** и **FloatToStr** в среде Delphi необходимо следить за разделителем целой и дробной части. В программе (Unit) в качестве разделителя используется только точка. А в окнах ввода и вывода на форме символ разделителя зависит от настроек Windows. В русифицированных версиях в качестве разделителя целой и дробной части числа используется запятая.

Для перевода целых чисел в строку и обратно используются соответственно функции **IntToStr(число)** и **StrToInt(строка).**

ЗАДАНИЯ

Задание 1.

Из двух полей ввода, одной надписи и четырёх кнопок создать калькулятор на четыре основных действия.

Задание 2.

Написать программу для перевода температуры по шкале Фаренгейта в шкалу Цельсия и обратно. (Tf=9/5*Tc+32).

Задание 3.

Написать программу для перевода скорости из км/ч в м/с и обратно.

Занятие 4. Стандартные математические функции.

Кроме операций в арифметических выражениях возможно использование стандартных функций. Необходимо следить за типом аргумента функции и типом её значения (см. таблицу).

Таблица стандартных функций языка Delphi

Функция	Назначение	Входные параметры	Тип результата	Примеры
`ABS( X )`	Возвращает абсолютное значение (модуль) аргумента	X - значение типа REAL или INTEGER	Такой же как и тип аргумента	`ABS(2.0) = 2.0000e+00;`
`SQR( X )`	Возвращает квадрат аргумента	X - значение типа REAL или INTEGER	Такой же как и тип аргумента	`SQR( 3 ) = 9; SQR(-2.0)= 4.0000e+00;`
`SQRT( X )`	Возвращает квадратный корень аргумента	X - значение типа REAL или INTEGER	REAL	`SQRT(16) =4.0000e+00; SQRT(25.0)=5.0000e+00;`
`EXP( X )`	Возвращает экспоненту аргумента	X - значение типа REAL и INTEGER	REAL	`EXP( 0 )=1.00000e+00; EXP(-1.0)=3.67879e-01;`
`LN( X )`	Возвращает натуральный логарифм аргумента	X - значение типа REAL или NTEGER	REAL	`LN( 1 ) = 0.00000e+00 LN(7.5) = 2.01490e+00`

SIN(X)	Возвращает значение синуса аргумента.	X - значение типа REAL или INTEGER, выраженного в радианах	REAL	SIN(0)=0.00000e+00; SIN(1.0)=8.41471e-01
COS(X)	Возвращает значение косинуса аргумента, выраженного в радианах	X - значение типа REAL или INTEGER	REAL	COS(0)=1.00000e+00; COS(1.0)=8.41471e-01
ARCTAN(X)	Возвращает арктангенс аргумента	X - значение типа REAL или INTEGER	REAL	ARCTAN(0)=0.0000e+00 ARCTAN(-1.0)=-7.8539e-01
ROUND(X)	Преобразует вещественное в целое. Возвращает округленное по модулю до ближайшего целого значение числа X	X - REAL	INTEGER	ROUND(3.1) = 3; ROUND(-3.1)= -3; ROUND(3.8) = 4; ROUND(-3.8)= -4; Внимание: числа, имеющие дробную часть, равную 0.5, округляются до ближайшего чётного. ROUND(3.5) = 4; ROUND(2.5) = 2;
TRUNC(X)	Возвращает целое значение, отбросив дробную часть числа X	X - REAL	INTEGER	TRUNC(3.1) = 3; TRUNC(-3.1)= -3; TRUNC(3.8) = 3;
INT(X)	Возвращает вещественное значение, отбросив дробную часть числа X	X - REAL	REAL	INT(3.1) = 3.00000E+00 INT(-3.1) = -3.00000E+00 INT(3.8) = 3.00000E+00

ЗАДАНИЯ

Задание 1.

Дано вещественное число. Вывести отдельно его целую и дробную часть.

Задание 2.

Считая, что Земля - идеальная сфера с радиусом R=6350 км, определить расстояние до линии горизонта от точки с заданной высотой над Землёй.

Задание 3.

Найти и вывести сумму и произведение трёх введённых с клавиатуры чисел.

Задание 4.

Треугольник задан координатами своих вершин. Найти и вывести периметр и площадь треугольника.

Задание 5.

Вычислите высоту дерева, если известны: расстояние до дерева и угол, под которым его видно. Полученный результат вывести в виде:

Высота дерева равна 2 м 87 см

Занятие 5. Логические выражения. Переменные булевского типа. Логические операции.

В правой части инструкции присваивания может стоять не только арифметическое выражение, но и выражение другого типа, например, логического.

Логическое выражение (или булевское) — это выражение, результатом которого является значение **TRUE** (истина) или **FALSE** (ложь). Наименование "булевский" выбрано в честь английского математика Джорджа Буля, заложившего основы математической логики. Термины булевский и логический обычно употребляются как синонимы.

Значение логического выражения можно присвоить переменной типа **Boolean**.

Пример описания переменной логического типа:

```
Var
   Exist : Boolean;
```

Логическое выражение может включать в себя: арифметические выражения, операции отношения и логические операции.

Операции отношения

Операции отношения предназначены для сравнения двух величин. Результат сравнения имеет значение **TRUE** или **FALSE**.

=	—	равно
<>	—	не равно
<	—	меньше
<=	—	меньше или равно
>	—	больше
>=	—	больше или равно

Пример:

```
Var
   X : Real ;
   Exist, Ok : Boolean;
begin
   X := 2.5 ;
   Ok:= X > 0 ;
   Exist:= X = 3 - 27 ;
end.
```

В результате выполнения этой программы переменная **Ok** примет значение **TRUE**, а переменная **Exist** — значение **FALSE**.

Логические операции

Логические операции применяются к величинам логического типа, результатом выполнения операции тоже является величина логического типа.

Рассмотрим следующие логические операции:

- **NOT** (отрицание, унарная операция)
- **AND** (и)
- **OR** (или)

Таблица значений логических операций

X	Y	Not X	X And Y	X Or Y
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Значение выражения вычисляется в определённом порядке.

Таблица приоритета выполнения операций

Тип действий	Операции
Вычисления в круглых скобках	**( )**
Вычисления значений функций	**функции**
Унарные операции	**not, унарный “-”**
Операции типа умножения	*** / div mod and**
Операции типа сложения	**+ - or**
Операции отношения	**= <> < > <= >=**

Операции одинакового приоритета выполняются слева направо в порядке их следования в выражении.

В качестве примера рассмотрим, в каком порядке выполняются операции и какое значение принимает следующее выражение

```
(a*2>b)  or not (c=7) and (d-1<=3), при a=2, b=4, c=6, d=4.
(2*2>4)  or not (6=7) and (4-1<=3)
(4>4)    or not (6=7) and (3<=3)
 false   or not  false  and   true
 false   or   true   and   true
 false   or   true
         true
```

Математическая запись $-4 < x \le 18.3$ на языке Delphi запишется в виде:

```
(x > -4) and (x<=18.3)
```

ЗАДАНИЯ

Задание 1.

Создать форму, содержащую надпись и одну кнопку, которая включает и выключает надпись.

Задание 2.

Светофор. Создать форму, содержащую три надписи и три кнопки. Каждая кнопка включает свою надпись (красный, жёлтый, зелёный) соответствующего цвета и выключает остальные.

Задание 3.

Создать форму, содержащую два поля ввода и одну кнопку. При нажатии на кнопку должна загореться надпись Истина, если число в первом поле ввода больше, чем во втором, и Ложь в противном случае.

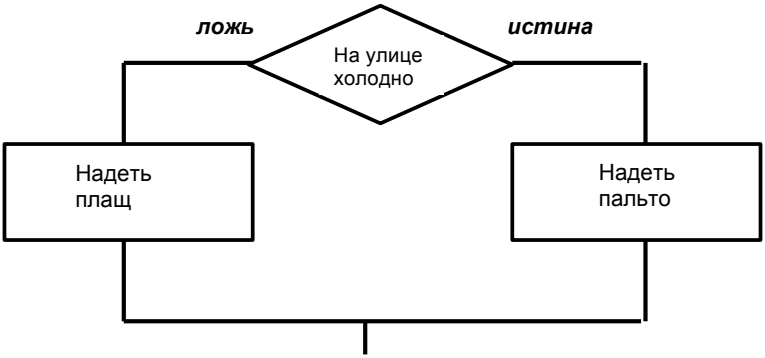
Задание 4.

В поле ввода вводится целое трёхзначное число. Написать программу, которая при нажатии на кнопку выводит надпись Истина, если сумма каких-либо двух цифр числа равна третьей цифр, и Ложь в противном случае.

Занятие 6. Организация ветвлений в программе. Инструкция IF...THEN...ELSE.

До сих пор мы писали программы, в которых действия выполнялись последовательно одно за другим, так как они написаны. Такая структура программы называется *линейной структурой*.

В жизни мы часто встречаемся с проблемой выбора. Например, если на улице холодно, мы надеваем пальто, иначе — надеваем плащ. Эту ситуацию можно представить в виде следующей схемы:



В зависимости от истинности утверждения в ромбе “на улице холодно” выполняется либо действие “надеть пальто”, либо действие “надеть плащ”.

Такая структура называется структурой ветвления.

Ситуация, связанная с выбором одной из двух альтернатив, встречается в программировании довольно часто. В языке Delphi структуре выбора соответствует инструкция ***If...Then...Else***.

Формат описания:

```
If <логическое выражение> Then
   <инструкция 1>
Else
   <инструкция 2> ;
```

Порядок выполнения инструкции ***If...Then...Else***:

- сначала вычисляется значение логического выражения,
- если значение выражения ***True***, то выполняется *инструкция 1*,
- иначе (если значение выражения ***False***), выполняется *инструкция 2*.

Пример 1.

Даны два целых числа. Найти среди этих чисел максимальное значение.

Решение:

```
procedure TfrmMy.btnMaxClick(Sender: TObject);
 var
   a,b,m : integer;
begin
   a:=StrToInt(edtA.Text);
   b:=StrToInt(edtB.Text);
   if a>b then
     m:=a
   else
     m:=b;
   lblMax.Caption:=IntToStr(m)
end;
```

Заметим, что в инструкции ***If...Then...Else*** точка с запятой перед словом ***Else*** никогда не ставится, так как эта инструкция представляет собой одно целое (***Else*** без ***If*** существовать не может).

В тех случаях, когда по правилам языка Delphi необходимо использовать только одну инструкцию, а выполнить нужно несколько действий, применяется составная инструкция. Такая составная инструкция состоит из нескольких инструкций, объединённых операторными скобками ***begin ... end***.

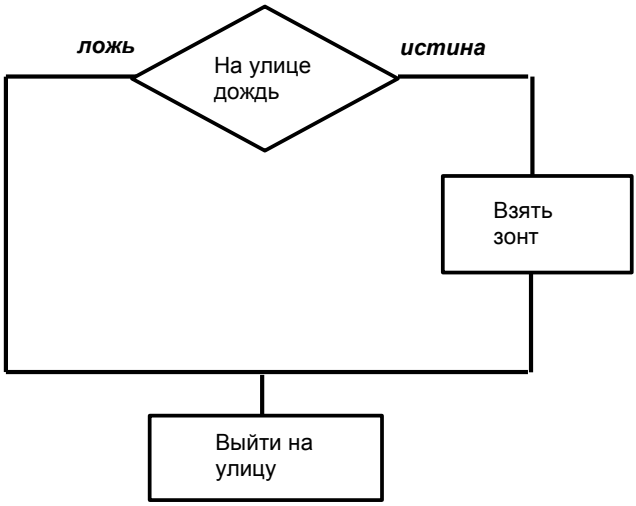
Например, если в ветви ***Then*** или ветви ***Else*** необходимо выполнить более одной инструкции, то эти инструкции заключаются в операторные скобки ***begin ... end***:

```
...
If x<0 Then
   begin
       y:=1;
       k:=k+1;
   end
Else
   begin
       y:=2*x;
        k:=x;
   end;
```

составная инструкция

составная инструкция

Иногда структура выбора выглядит несколько иначе:



Такой структуре в Delphi соответствует краткая инструкция ***If***...***Then***.

Формат описания:

```
If <логическое выражение> Then
   <инструкция 1>;
```

Краткая инструкция выполняется аналогично полной, то есть вычисляется значение логического выражения, если его значение ***True***, то выполняется инструкция 1, иначе выполняется следующая за ***If***...***Then*** инструкция.

ЗАДАНИЯ

Задание 1.

Вводится два целых числа m и n.

Если число m делится нацело на число n, то вывести частное от деления, в противном случае вывести сообщение "m на n нацело не делится".

Задание 2.

Дано трёхзначное число. Выяснить, является ли оно **палиндромом** ("перевёртышем"), то есть таким числом, десятичная запись которого читается одинаково слева направо и справа налево.

Задание 3.

Определить максимальное и минимальное значение из трёх различных вещественных чисел.

Задание 4.

В подъезде жилого дома имеется ***n*** квартир, пронумерованных подряд, начиная с номера ***a***. Определить, является ли сумма номеров всех квартир чётным числом.

Занятие 7. Вложенные инструкции IF...THEN...ELSE. Практика решения задач.

Инструкции, стоящие в структуре выбора после слов ***Then*** и ***Else***, сами могут быть инструкциями выбора, в этом случае инструкция ***If***...***Then***...***Else*** называется вложенной.

Например:

```
If <логическое выражение 1> Then
    <инструкция 1>
Else
    If <логическое выражение 2> Then
        <инструкция 2>
    Else
        <инструкция 3> ;
```

Каждая из инструкций (инструкция 1, инструкция 2, инструкция 3) может быть составной, т.е. в пункт ***Then*** или ***Else*** можно поместить более одной инструкции, заключив их в скобки **begin**... **end**.

Например:

```
If <логическое выражение 1> Then
    begin
        <инструкция 1>
        If < логическое выражение 2> Then
            begin
                <инструкция 2>
                <инструкция 3>
            end
        Else
            <инструкция 4>;
    END
Else
    If < логическое выражение 3> Then
        <инструкция 5>
    Else
        begin
            <инструкция 6>;
            <инструкция 7>;
        end;
```

Пусть задан следующий фрагмент:

```
If <логическое выражение 1> Then
    If <логическое выражение 2> Then
        <инструкция 1>
    Else
        <инструкция 2>;
```

Он может показаться двусмысленным, поскольку не ясно, к какому ***Then*** относится ***Else*** (к первому или ко второму). На этот счёт существует правило: любой пункт ***Else*** относится к ближайшему IF.

Чтобы избежать путаницы и возможных ошибок, желательно вложенные конструкции ***If***... ***Then***... ***Else*** заключать в операторные скобки ***begin***... ***end***.

ЗАДАНИЯ

Задание 1.

Найти корни квадратного уравнения $Ax^2+Bx+C=0$ ($A \neq 0$). Если уравнение не имеет действительных корней, вывести на экран соответствующее сообщение.

Задание 2.

Определить, является ли заданный год високосным. Год является високосным, если он кратен 4, но из лет, кратных 100, високосными считаются кратные 400. (Например, 1700, 1800 и 1900 — не високосные годы, а 2000 год — високосный).

Задание 3.

Даны действительные числа X, Y ($X \neq Y$). Меньшее из этих двух чисел заменить их средним арифметическим, а большее — их утроенным произведением.

Задание 4*.

Дано целое число k (1 <= k <= 180). Определить, какая цифра находится в k-ой позиции последовательности 10111213 ... 9899; в которой записаны подряд все двухзначные числа.

Занятие 8. Процедуры

Для того, чтобы писать серьёзные сложные программы с малыми затратами сил и времени, избегая множества ошибок, мало знать операторы какого-либо языка программирования. Необходимо овладеть навыками структурного программирования (навыками разработки и написания программ).

Применение их на практике позволит Вам:

- просто и легко писать программы;
- получать понятные и легко читаемые программы;
- быстро находить ошибки в программе;
- быстро изменять и совершенствовать программу.

Одна из главных идей структурного программирования — это, так называемое, ***программирование сверху вниз***. То есть поставленная задача разбивается на несколько более простых подзадач. Далее каждая из подзадач также разбивается на части. Этот процесс продолжается до тех пор, пока в итоге Вы не получите несколько элементарных и простых подзадач, которые нужно решить.

Отдельные подзадачи в языке Delphi называются ***процедурами***.

Решим следующую задачу: найти корни квадратного уравнения заданного своими коэффициентами.

Сначала создадим новую форму, которой присвоим имя *frmKvUr,* в свойстве *Caption* напишем «*Решение квадратного уравнения*».

На данной форме разместим поля ввода ***edtA***, ***edtB***. *edtC* для ввода переменных соответственно A,B и C. Свойство ***Text*** этих полей зададим пустое, чтобы удобно и быстро вводить туда значения коэффициентов.

Над этими полями ввода разместим надпись ***lblZag***, в свойстве ***Caption*** которой будет написано: «*Ввести коэффициенты квадратного уравнения*».

Ещё нам понадобятся надписи ***lblX1***, ***lblX2*** для вывода корней квадратного уравнения и надпись ***lblNo***, в свойстве ***Caption*** которой напишем «*Действительных корней нет*». И в свойству *Visible* этих надписей присвоим значение «*False*», чтобы надписи не были видны при запуске программы.

Также нам понадобится кнопка ***btnFind***, при нажатии на которую уравнение будет решено.

Нажмём два раза на кнопку и начнём писать программу.

Сначала напишем алгоритм решения этой задачи, используя комментарии, которые в Delphi можно заключать в фигурные скобки или, что более удобно, отделять от текста программы двумя слешами //:

```
//Вводим  коэффициенты квадратного уравнения A, B,C
//Считаем дискриминант D
//Если D>=0 тогда
      //Считаем корни квадратного уравнения X1 и X2
```

```
      //Выводим значения корней X1 и X2
  //иначе
      //Сообщаем, что действительных корней нет
```

В такой записи решения задачи сразу прослеживается количество нужных нам переменных, процедур и функций, которые мы должны описать. Если одну строку алгоритма решения задачи нужно заменить несколькими операторами языка Delphi, то удобно описать процедуру, а если одна строка заменяется одним оператором, то процедура не нужна.

Исходя из сказанного, запишем каждый пункт алгоритма на языке Delphi:

```
procedure TfrmKvUr.btnFindClick(Sender: TObject);
   var
     A,B,C,D,X1,X2:real;
  begin
    vvod(A,B,C); //Вводим коэффициенты A,B,C
    D:=sqr(B)-4*A*C; //Считаем дискриминант D
    if D>=0 then //Если дискриминант больше или равен 0, то
     begin
       calc(A,B,D,X1,X2); //Считаем корни x1,x2 (в
                          зависимости от A, B и D)
       prn(X1,X2); //Выводим значение корней x1, x2
     end
    else //Иначе
      lblNo.Visible:=true; //Сообщаем, что действительных
                           корней нет
 end;
```

В теле программы мы видим названия действий, которые необходимо выполнить для решения задачи. Эти действия, как было сказано выше, называются процедурами, величины в круглых скобках называются параметрами процедур.

Когда Delphi встречает название действия, которое не является его собственной инструкцией, он рассматривает это как вызов процедуры и передаёт управление в процедуру с указанным именем (в нашем случае vvod, calc и prn являются именами процедур). После выполнения всех действий в процедуре управление возвращается в основную программу на инструкцию, следующую после вызова процедуры.

Формат вызова процедуры:

```
<имя процедуры> ( <список фактических параметров> ) ;
```

Процедуры должны быть описаны в разделе объявлений ***Implementation*** до основной программы, то есть прежде чем вызвать процедуру, её необходимо описать.

Формат описания процедуры:

```
procedure <имя процедуры> ( <список формальных параметров> );
    <блок объявлений>
    begin
      <действия>
    end;
```

Как мы видим, при вызове процедуры указываются фактические параметры, а при описании формальные. В списке формальных параметров обязательно указывается их тип. Существует однозначное соответствие между формальными и фактическими параметрами.

Число, порядок следования и тип формальных и фактических параметров должны совпадать.

Давайте опишем процедуры и начнём с процедуры prn:

В этой процедуре формальные параметры Xf и Xs соответствуют фактическим параметрам X1 и X2. Формальные параметры Xf и Xs – параметры значения, т.е. они передаются из программы в процедуру, а обратно не возвращаются (Нам это и не нужно).

```
Procedure Prn(Xf,Xs:real);
   begin
     frmKvUr.lblX1.Visible:=true;
     frmKvUr.lblX1.Caption:='x1='+FloatToStr(xf);
     frmKvUr.lblX2.Visible:=true;
     frmKvUr.lblX2.Caption:='x2='+FloatToStr(xs);
   end;
```

В данной процедуре мы делаем надписи с решением видимыми и заносим туда ответ. Если бы мы просто написали ***lblX1.Visible:=true***, то получили бы внизу экрана сообщение об ошибке. В процедурах, описываемых пользователем, надо обязательно указывать полное название компонента, т.е. на какой форме он расположен, так как в общем случае в одном проекте может быть несколько форм.

Теперь перейдём к написанию процедуры ввода коэффициентов. Значение коэффициентов надо взять из окон ввода и положить в соответствующие переменные. Чтобы значения переменных были переданы из процедуры в программу надо использовать параметры-переменные, которые как раз и указывают откуда значение взять и куда вернуть. При описании процедур перед формальными параметрами-переменными ставится служебное слово ***Var*** (variable), что в переводе и означает переменная. В списке фактических параметров в качестве параметров-переменных могут быть использованы только переменные.

```
Procedure vvod(var k1,k2,k3:real);
  begin
      k1:=StrToFloat(frmKvUr.edtA.Text);
      k2:=StrToFloat(frmKvUr.edtB.Text);
      k3:=StrToFloat(frmKvUr.edtC.Text);
  end;
```

Формальные параметры-переменные *k1,k2,k3* соответствуют фактическим параметрам a, b, c. Обращаем ваше внимание на использование функции ***StrToFloat,*** если её не поставить сразу, то Вам не будет предложено выбрать свойство *Text* у компонента Edit, т.к. это свойство типа строка, а переменные, отвечающие за коэффициенты уравнения вещественные.

Опишем последнюю процедуру для расчёта корней уравнения. Значения корней нам необходимо вернуть в программу, поэтому это будут параметры-переменные *Xf, Xs*, а

значения первого, второго коэффициентов и дискриминанта мы только передадим в процедуру, возвращать их не надо. Значит, это будут параметры-значения.

```
procedure Calc(k1,k2,dis:real; var Xf,Xs:Real);
 begin
   Xf:=(-k2+Sqrt(dis))/(2*k1);
   Xs:=(-k2-Sqrt(dis))/(2*k1);
 end;
```

После того, как мы запустим программу на выполнение и введём значения коэффициентов уравнения, нам будет выведена та или иная надпись. Если мы изменим значения коэффициентов и получим другую надпись, то увидим, что первая надпись тоже осталось на экране. Поэтому, чтобы этого не происходило, опишем ещё одну процедуру, которая будет приводить надписи в начальное состояние. Эта будет процедура без параметров. Назовём её ***Init***.

```
procedure Init;
begin
    frmKvUr.lblX1.Visible:=false;
    frmKvUr.lblX2.Visible:=false;
    frmKvUr.lblNo.Visible:=false;
end;
```

ЗАДАНИЯ

Задание 1

Составить программу, в результате выполнения которой переменная **A** меняется значением с переменной **B**, а переменная **C** - с переменной **D**. При решении определите процедуру, осуществляющую обмен значениями двух переменных.

Задание 2

Даны стороны двух треугольников. Найти сумму их периметров и сумму их площадей. Определить процедуры для расчёта периметра и площади треугольника по его сторонам.

Задание 3

Найти площадь кольца, внутренний радиус которого равен **R1**, а внешний - **R2**. Определить и использовать процедуру вычисления площади круга.

Задание 4

Найти площадь выпуклого четырёхугольника, вершины которого заданы своими координатами.

Занятия 9. Функции.

Рассмотрим теперь функции. Процедура – это частный случай функции, т.к. функция может делать то же самое, что и процедура, но обязательно выдаёт хотя бы одно значение – это значение функции.

В среде Delphi существует много встроенных функций, но часто встречаются задачи, в которых надо описывать функции для простоты решения. Например, нам надо при решении задачи много раз использовать функцию Tg(A), а такой встроенной функции нет. Давайте разберём простой пример

Имеется прямоугольный треугольник. Для него задаётся величина одного из катетов и прилегающего угла (в градусах). Необходимо рассчитать величину другого катета.

При двойном нажатии на кнопку напишем программу.

```
procedure TfrmKatet.btnRunClick(Sender: TObject);
   var
      a,b.alfa:real;
begin
      alfa:=StrToFloat(edtAlfa.Text); //Вводим угол alfa
      a:=StrToFloat(edtAlfa.Text); //Вводим прилежащий катет a
      b:=a*tg(alfa); //Рассчитаем величину другого катета b
      lblB.Caption:= FloatToStr(b); //Выводим значение b
end;
```

Нам осталось только описать нашу функцию. Функции описываются в разделе объявлений. После служебного слова ***Function*** указывается имя функции, список формальных параметров и далее тип результата, выдаваемого функцией.

Формат описания функции:

```
Function <имя функции> ( <список формальных параметров> ):
<тип результата>;
   <блок объявлений>
   begin
       <действия>
   end;
```

Функция имеет как параметры-значения, так и параметры-переменные. Работают параметры такпже, как в процедуре.

Внутри тела функции должен быть хотя бы один оператор присваивания, который имени функции присваивает значение.

Вызывается функция по имени, но обязательно в выражении.

```
Function tg(x:real):real;
  var
      y : real;
Begin
```

```
    y:=x/180*pi;
    tg:=sin(y)/cos(y);
End;
```

Здесь при переводе градусов в радианы мы использовали одну дополнительную переменную, которая называется локальной, описана внутри функции и видна только при работе функции.

ЗАДАНИЯ

Задание 1.

Вычислить значение выражения: $x = \frac{\sqrt{6}+6}{2} + \frac{\sqrt{13}+13}{2} + \frac{\sqrt{21}+21}{2}$, используя функцию $y = \sqrt{x} + x$

Задание 2.

Вычислить значение выражения $x = \frac{15+\sqrt{8}}{8+\sqrt{15}} + \frac{6+\sqrt{12}}{12+\sqrt{6}} + \frac{7+\sqrt{21}}{21+\sqrt{7}}$, используя функцию $y = a + \sqrt{b}$

Задание 3.

Вводятся координаты трёх вершин треугольника, вычислить его периметр.

Задание 4.

Вводятся a, b и c. Вычислить значение выражения $t = \frac{\max(a,b,c) - \min(a,b,c)}{2 + \max(a,b,c) \cdot \min(a,b,c)}$, где функции min(a, b, c) и max(a, b, c) возвращают, соответственно, минимальное и максимальное из трёх чисел.

Задание 5.

Вычислить значение выражения $z = (sign(x) + sign(y)) \cdot sign(x + y)$, где

$$sign(a) = \begin{cases} -1 & a < 0 \\ 0 & a = 0 \\ 1 & a > 0 \end{cases}$$

Задание 6.

С клавиатуры вводятся координаты точек A, B, и C, лежащих на плоскости. Определить, возможно ли по этим точкам построить невырожденный треугольник. Если возможно, то рассчитать для него длины сторон, величины высот, площадь, периметр и величины углов, выраженные в градусах. Вывести результат на экран, указав какие именно углы, стороны, высоты были рассчитаны. Например, сторона AB=…, угол BCA=… и т.д. При написании программы определить функции расчёта длины стороны, величины угла, величины высоты.

Если треугольник построить невозможно – выдать соответствующее сообщение.

Занятие 10. Графика в Delphi

Для того, чтобы работать с графикой в среде Delphi существует много возможностей. Рассмотрим некоторые из них. Рисунки будем помещать на объекте ***PaintBox***.

Выберем на палитре компонентов вкладку *Additional* (дополнительная). Мы увидим пиктографическое меню для этой вкладки и выберем из него пиктограмму ***TPaintBox***. Растянув данный компонент на форме, мы получим место для размещения рисунков, как будто мы изготовили раму для рисования картины, а теперь на раму необходимо натянуть холст, полотно. Это мы будем делать при помощи свойства ***Canvas*** (полотно). Начало координат нашего рисунка находится в левом верхнем углу объекта, ось X направлении горизонтально слева направо, а ость Y направлена сверху вниз. Размер рисунка можно посмотреть поставив курсор на компонент ***PaintBox***, или .используя свойства ***Height*** (высота) и ***Width*** (ширина). Например, изменив свойство имя (Name) объекта ***PaintBox,*** например, на pbxEx, определим центр компонента ***PaintBox.***

```
x0:= pbxEx.Width div 2;
y0:= pbxEx.Height div 2;
```

Для создания изображения на холсте нам надо иметь инструменты для рисования. Также и в Delphi рисовать на холсте можно с помощью пера, закрашивать фигуры кистью, текст выводить с помощью свойства «шрифт».

Для изменения цвета и толщины линии любой фигуры (т.е. рисования фигуры) используется объект ***Pen*** (перо).

```
pbxEx.Canvas.Pen.Color:=clRed;
pbxEx.Canvas.Pen.Width:=3;
```

После такого задания свойств «пера» фигуры будут нарисованы красным цветом и чуть более толстой линией, чем по умолчанию.

Объект ***Brush*** (кисть) служит для заполнения внутреннего пространства замкнутых фигур.

```
pbxEx.Canvas.Brush.Color:=clGreen;
```

Цвет кисти станет зелёным.

Рассмотрим, как установить цвет кисти белый и произвести заливку всей рабочей области белым цветом.

```
pbxEx.Canvas.Brush.Color := clwhite;
pbxEx.Canvas.FillRect(ClientRect);
```

Стиль закраски и стиль линии можно тоже менять при помощи свойства стиль. Формат обращения к этому свойству предлагаем найти самостоятельно.

Рассмотрим стандартные графические примитивы, которые мы можем разместить на нашем холсте.

Точка.

Для окрашивания точки (X,Y) в определённый цвет используется следующее свойство - точка. Свойство ***Pixels[x,y]*** – задаёт цвет точки с координатами X,Y.

```
pbxEx.Canvas.Pixels[x,y]:=clRed;
```

То есть, пиксель с координатами (X,Y) закрашивается в данном случае красным цветом. Обращаем ваше внимание на то, что квадратные скобки используются только для окрашивания точки.

Линия.

Линия рисуется с помощью метода

```
pbxEx.Canvas.LineTo(X,Y)
```

Линия рисуется от текущего местоположения графического курсора в точку, координаты которой указаны в скобках.

Перо (графический курсор) можно перемещать без прорисовки следа в точку (X.Y) с помощью процедуры

```
pbxEx.Canvas.MoveTo(X,Y)
```

Любая фигура не перемещает графический курсор.

Прямоугольник.

Для рисования прямоугольника используется метод

```
pbxEx.Canvas.Rectangle(X1,Y1,X2,Y2)
```

Данная процедура рисует закрашенный прямоугольник со сторонами параллельными экрану и с диагональю с координатами (X1,Y1,X2,Y2).

Граница прямоугольника по умолчанию чёрного цвета, а закраска белая.

Эллипс.

Задаётся прямоугольником, в который он вписывается. Оси эллипса параллельны границам экрана. Эллипс рисуется закрашенным.

```
pbxEx.Canvas.Ellipse(X1,Y1,X2,Y2)
```

ЗАДАНИЯ

Задание 1.

Нарисовать снеговика.

Задание 2.

Нарисовать дом.

Задание 3.

Придумать свою картинку и нарисовать. Обязательно использовать различные фигуры, линии, толщину линий и цвета закраски.

Занятие 11. Циклы.

Организация циклов в программах

Прежде чем начать изучать графику повторим алгоритмическую конструкцию «цикл» и операторы языка Delphi, соответствующие этой конструкции.

Цикл — это многократно выполняемая последовательность действий. Существует два вида цикла — цикл с предусловием (цикл ”пока”) и цикл с постусловием (цикл “до”).

Циклу с предусловием (цикл “ПОКА”) в языке Delphi соответствует инструкция ***While...do***... .

Формат описания:

```
While <логическое выражение> do
      <инструкция> ;
```

Цикл ***While...do*** многократно выполняет одни и те же действия при истинности выражения, которое изменяется обычно внутри цикла. Значение логического выражения вычисляется перед выполнением инструкции. Таким образом, если значение выражения с самого начала оказалось FALSE, то инструкция не будет выполнена ни разу (значение логического выражения является условием продолжения цикла).

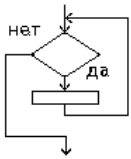


Схема цикла ***While...do***

Если в цикле необходимо выполнить несколько инструкций, то после служебного слова ***do*** используются операторные скобки ***begin... end***.

Формат описания:

```
While <логическое выражение> do
      begin
         <инструкция 1> ;
         <инструкция 2> ;
         <инструкция 3> ;
      end;
```

Циклу с постусловием (цикл “ДО”) в языке Delphi соответствует инструкция ***Repeat... Until...*** .

Формат описания:

```
Repeat
   <инструкция 1> ;
   <инструкция 2> ;
   . . .
```

```
    <инструкция n> ;
Until <логическое выражение>
```

В данном цикле сначала выполняются инструкции, составляющие тело цикла (часть текста программы, заключённая между словами ***Repeat*** и ***Until***), затем вычисляется значение логического выражения, если оно ***False***, то инструкции в теле цикла повторяются, иначе (значение выражения — ***True***) цикл завершается. Таким образом, в инструкции ***Repeat... Until*** тело цикла всегда выполняется хотя бы один раз, а значение логического выражения является условием завершения цикла.

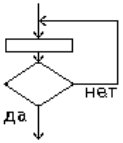


Схема цикла ***Repeat... Until***

В том случае, если заранее известно число повторений тела цикла, в Delphi используется цикл со счётчиком — цикл ***For***.

Формат описания:

```
For <перем. цикла> := <нач. значение> To <кон.значение> do
  <инструкция> ;
```

или

```
For <перем. цикла> := <нач. значение> DownTo <кон. значение> Do
   <инструкция> ;
```

где *<переменная цикла>* — переменная типа Integer, принимающая значение с **начального** до **конечного** с шагом **+1** — в случае ***To***, **-1** — в случае ***DownTo***. Значение переменной цикла изменяется автоматически;

<начальное значение> — любое целочисленное арифметическое выражение, значение которого является начальным значением переменной цикла;

<конечное значение> — любое целочисленное арифметическое выражение, значение которого является конечным значением параметра цикла;

<инструкция> — действие, повторяемое до тех пор, пока начальное значение переменной цикла не станет больше — в случае ***To*** (меньше — в случае ***DownTo***) конечного значения. Если в цикле необходимо выполнить несколько инструкций, то после служебного слова ***do*** используются операторные скобки ***begin... end***..

Значения арифметических выражений вычисляются только один раз при входе в цикл.

Инструкция ***FOR*** соответствует циклу с предусловием.

ЗАДАНИЯ

Задание 1.

Нарисовать ***N*** горизонтальных линий, расположенных на одинаковом расстоянии друг от друга. Число ***N*** задаётся через соответствующее поле ввода.

Задание 2.

Нарисовать ***N*** квадратов со стороной ***S***, расположенных горизонтально на одинаковом расстоянии друг от друга. Числа ***N*** и ***S*** задаются через соответствующие поля ввода.

Задание 3.

Нарисовать клетчатое поле, состоящее из ***N*** строк и ***M*** столбцов. Числа ***N*** и ***M*** задаются через соответствующие поля ввода.

Задание 4.

Нарисовать шахматную доску, состоящую из ***N*** строк и ***M*** столбцов. Числа ***N*** и ***M*** задаются через соответствующие поля ввода.

Задание 5.

Нарисовать ***N*** концентрических окружностей. Радиус минимальной окружности - ***r***, радиус максимальной окружности - ***R***. Числа ***N*** ***r*** и ***R*** задаются через соответствующие поля ввода.

Задание 6.

Нарисовать окружность, состоящую из N точек радиуса R с заданными координатами центра Xc, Yc. Все параметры вводятся в поле ввода. Координаты точки, лежащей на окружности, определяются следующим образом:

```
X = Xc + R cos(α)
Y = Yc - R sin α
```

Угол ***α*** меняется от 0 до 2π на величину 2π /n.

Задание 7.

Вводятся 3 числа – это длины отрезков. Если из этих отрезков можно построить треугольник, то сделать это, если нельзя, то сообщить об этом.

Задание 8.

Нарисовать N вложенных друг в друга квадратов, определив процедуру рисования квадрата из линий.

Задание 9.

Нарисовать упрощённые часы (без цифр, но с чёрточками). В поле ввода вводится время. По нажатию кнопки рисуются стрелки в нужном положении.

Занятие 12. Строки

Мы уже работали со строками в среде Delphi. Например, свойство *Caption* или *Text* – это свойства, значения которых могут быть только строками. Что же такое строка и как с ней работать?

Строка - это последовательность символов, заключённая в апострофы. При описании переменной для работы со строками используется тип ***String***.

```
Var
    s: String;
```

Такая запись означает, что в программе будет использоваться строковая переменная s с неограниченной длиной строки.

Строки можно складывать. Эта операция называется конкатенация, или сложение, или слияние строк и обозначается знаком плюс. Например:

```
Var
    s, st: String;
Begin
    s:='Мы изучаем'; //В первую строку кладём значение
                        «Мы изучаем»
    st:=' Delphi;  //Во вторую строку кладём значение
                     «Delphi»
    s:=s+st;  //Складываем эти две строки
End;
```

В результате выполнения данной программы мы получим в переменной ***S*** строку: *Мы изучаем Delphi*.

Строки можно сравнивать.

Сравнение осуществляется посимвольно слева направо в соответствии с тем порядком, в котором символы хранятся в компьютере. Латинские буквы хранятся в алфавитном порядке, цифры – от 0 до 9 в порядке возрастания.

Пример:

```
'AB' > 'AA'
'A' < 'AB'
'DC' > 'ABCDE'
'ABCE' > 'ABCD'
```

Для работы со строками существует несколько стандартных функция и процедур. Рассмотрим их.

Таблица стандартных процедур и функций работы со строками

Процедуры работы со строками		
Имя	Тип параметров	Назначение

`Delete(St,Poz,N)`	`St: string;` `Poz, N: integer;`	Удаление N символов строки St, начиная с позиции Poz
`Insert(St1,St2, Poz)`	`St1,St2: string;` `Poz: integer;`	Вставка строки St1 в строку St2, начиная с позиции Poz
Функции работы со строками		
Имя	Типы результата и параметров	Назначение

`Copy(St,Poz,N)`	`Результат: string;`
	`St: string;`
	`Poz, N: integer;`
		Копирует из строки St подстроку длиной N символов, начиная с позиции Poz.

`Length(St)`	`Результат: integer;` `St: string;`	Вычисляет длину строки St, то есть число символов, из которых состоит строка.
`Pos(St1,St2)`	`Тип: integer;` `St1,St2: string;`	Обнаруживает первое появление в строке St2 подстроки St1. Значением функции является номер той позиции, где находится первый символ подстроки St1. Если в St2 подстроки St1 не найдено, результат равен 0.

Примеры:

Процедура **Delete**

Значение St	Выражение	Значение St, после выполнения процедуры
`'abcdef'`	`Delete(St,4,2)`	`'abcf'`
`'Turbo-Pascal'`	`Delete(St,1,6)`	`'Pascal'`

Процедура **Insert**

Значение St1	Значение St2	Выражение	Значение St2, после выполнения процедуры
`'Turbo'`	`'-Pascal'`	`Insert(St1,St2,1)`	`'Turbo-Pascal'`
`'-Pascal'`	`'Turbo'`	`Insert(St1,St2,6)`	`'Turbo-Pascal'`

Функция **Copy**

Значение St	Выражение	Значение Str, после выполнения процедуры
'abcdefg'	Str:=Copy(St,2,3);	'bcd'
'abcdefg'	Str:=Copy(St,4,4);	'defg'

Функция **Length**

Значение St	Выражение	Значение N, после выполнения процедуры
'abcdefg'	N:=Length(St);	7
'Turbo-Pascal'	N:=Length(St);	12

Функция **Pos**

Значение St2	Выражение	Значение N, после выполнения процедуры
'abcdef'	N:=Pos('de',St2);	4
'abcdef'	N:=Pos('r',St2);	0

Рассмотрим пример

В процедуру передаётся в качестве параметра строка, содержащая два слова, разделённых одним пробелом.

Процедура должна поменять слова местами.

```
Procedure Change(var s : String);
var
   s1:string;
begin
   s1:= copy(s,1,pos('␣',s)-1); // Выделить первое слово S1
   //(словом будет всё, что находится до пробела)
   delete(s,1,pos('␣',s)); //Удалить первое слово и пробел
   // (в переменной S останется второе слово)
   s:=s+' '+s1; //Поставить пробел и первое слово в конец
end;
```

Примечание. Значком ′␣′ здесь обозначен пробел

ЗАДАНИЯ

Задание 1.

В поле ввода вводится строка, содержащая три слова, разделённые пробелом. Напишите программу, которая при нажатии кнопки меняет местами второе и третье слово.

Задание 2.

В поле ввода вводится строка. Заменить в ней все пробелы на восклицательные знаки.

Задание 3.

В поле ввода вводится строка. Подсчитать сколько раз в ней встречается точка.

Задание 4.

В поле ввода вводится строка. Подсчитать сколько раз в неё входит подстрока abc.

Задание 5.

В поле ввода вводится строка, в которой есть одна открывающая и одна закрывающая круглые скобки. Вывести в поле вывода символы, заключённые между этими скобками.

Задание 6.

В поле ввода вводится строка. Подсчитать в ней количество слов.

Задание 7.

В поле ввода вводится строка. Заменить в ней слово dog на слово cat..

Задание 8.

В поле ввода вводится слово. Получить новое слово, образуемое путём прочтения исходного слова, начиная с его конца.

Задание 9*.

В поле ввода вводится строка, содержащая несколько слов. Подсчитать в ней количество слов. Поменять в ней местами второе и предпоследнее слово.

Занятие 13. Строки. Работа с численной информацией

Процедура ***Val (st, X, code)*** преобразует строку символов ***st*** во внутреннее представление целой или вещественной переменной ***X***, которое определяется типом этой переменной.

Параметр ***code*** (тип integer) содержит ноль, если преобразование прошло успешно, и тогда в X помещается результат преобразования, в противном случае он содержит номер позиции в строке ***st*** , где обнаружен ошибочный символ, и в этом случае содержимое ***X*** не меняется.

В строке ***st*** могут быть ведущие и/или ведомые пробелы. Если содержит символьное представление вещественного числа, разделителем целой и дробной части должна быть точка.

Пример:

Для проверки ввода чисел ***X*** и ***Y*** с клавиатуры используем процедуру ***Val.***
Если в переменные ***sx, sy*** введены числа, то переменные ***cx*** и ***cy*** примут значение 0, если нет, то отличные от 0.

```
Val (sx, x, cx);
Val (sy, y, cy);
If (cx=0) and (cy=0) Then
   begin
      <операторы>
   end
Else
   begin
      <операторы>
   end
```

Процедура ***Str (X [:Width [: Decimal]], st:)*** преобразует число ***X*** любого вещественного или целого типов в строку символов ***st*;** параметры ***Width*** и ***Decimal*,** если они присутствуют, задают формат преобразования.

Width определяет общую ширину поля, выделенного под соответствующее символьное представление числа ***X***, а ***Decimal*** – количество символов в дробной части (этот параметр имеет смысл только в том случае, когда *X* – вещественное число).

Пример:

```
var
  x:integer;
  y:real;
  s:string;
begin
  y:=3.5;
  Str(y,s);
```

```
        ……..;
        ……..
    end;
```

В переменной ***s*** будет записано **3.500000000000000E+0000**

Однако, если записать преобразование следующим образом ***str(y:6:2, s);*** то получим **ںں3.50**

отсутствующие позиции целой части заменяются пробелами (знак ں мы использовали для обозначения пробела)

Рассмотрим другой пример:

Str(12345.6:5:2, s).

В этом случае целая часть числа выводится целиком, т.е. ограничение игнорируется.

Получим **12345.6**

Рассмотрим ещё один пример:

Str(12.3456:5:2, s).

В этом случае дробная часть числа округляется до указанного знака после запятой.

Получим **12.35**

ЗАДАНИЯ

Задание 1

В поле ввода вводится строка. Удалить из неё все встретившиеся цифры. Результат вывести в соответствующую надпись.

Задание 2

В поле ввода вводится строка. Подсчитать сумму всех встретившихся в ней цифр.

Задание 3

В поле ввода вводится строка, в которой есть слова и целые числа, отделённые друг от друга одним пробелом. Подсчитать сумму всех чисел.

Задание 4

В поле ввода вводится строка вида **число1 знак_плюс число2 знак_равно** без пробелов. Вписать после знака равно результат.

Пример

Исходная строка 12.35+7.123=

Результирующая строка 12.35+7.123=19.473

Задание 5

В поле ввода вводится строка, в которой есть одна открывающая и одна закрывающая круглые скобки. Между скобками записано несколько цифр без пробелов. Вписать вместо всех цифр их среднее арифметическое.

Задание 6

В поля ввода вводятся две строки вида **ФамилияЧеловека Пробел Число**, где Число - это рост Человека в сантиметрах. Вывести фамилию более высокого человека. Если рост одинаков, то вывести обе фамилии.

Задание 7

В поле ввода вводится целое число. Записать его в двоичной системе счисления.

Задание 8*

В поле ввода вводится целое число. Записать его в шестнадцатеричной системе счисления.

Занятие 14. Компонент Memo

Компонент ***Memo*** – многострочное редактируемое текстовое поле.

Компонент Memo предназначен для ввода, редактирования и отображения достаточно длинного многострочного текста.

С помощью различных свойств и методов можно динамически формировать содержимое компонента Memo (или поля Memo).

Свойства ***Font*** **и** ***Readonly*** поля Memo аналогичны этим же свойствам поля ввода. Свойство *Text* содержит весь текст поля Memo, однако это свойство доступно только по время выполнения.

При выводе в область просмотра текста на русском языке необходимо учитывать, что не все шрифты оснащены кириллицей. Кроме того, некириллической может оказаться текущая кодовая таблица. Если вместо русского текста на экране появилась абракадабра, го нужно изменить значение свойства ***Charset*** **(**Набор символов) объекта ***Font*** (Шрифт). Для большинства шрифтов удовлетворительными значениями свойства *Charset* являются ***DEFAULT_CHARSET*** или ***RUSSIAN_CHARSET***.

Свойства

Свойство Lines

Текст компонента Memo хранится в свойстве ***Lines*** и представляет собой пронумерованный набор строк (нумерация начинается с нуля) – первая строка имеет индекс 0, вторая - индекс 1 и т.д.

Пример:

```
Var
   s : string;
Begin
   s:= memEx.Lines[2];
   //переменной s присваивается содержимое 3-ей строки
   memEx.Lines[3]:='Привет'; //строке с номером 4 присвоить
   //строковую константу 'Привет'
   …..
end;
```

Обращаться указанным способом к строкам поля Memo из программы можно только, если строки уже ранее созданы либо через инспектор объектов, либо введены с клавиатуры после запуска программы, либо созданы соответствующим методом. Нельзя обратиться к несуществующей строке.

С помощью свойства ***Lines*** можно заполнить поле Memo.

Для этого в инспекторе объектов надо нажать на кнопку с многоточием в поле свойства ***Lines;*** откроется окно редактора строк. В окне редактора строк можно вводить строки, заканчивая набор каждой строки нажатием клавиши ***<Enter>.***

После окончательного ввода строк, нажмите кнопку ***<OK>.***

Свойство Count

Общее количество строк можно узнать с помощью свойства ***Count***.

```
k:=memEx.Lines.Count;
```

Свойство ***Count*** имеет статус ***Read/Only***, т.е. изменять (редактировать) его нельзя.

Свойство WordWarp

Определяет, будут ли переноситься строки, выходящие за пределы области просмотра или они останутся невидимыми.

Свойство MaxLength

Определяет максимальное количество символов во всем поле Memo. Если значение свойства *MaxLength* равно ***0***, то количество символов ничем не ограничено.

Свойство ScrollBars

Это свойство определяет наличие или отсутствие полос прокрутки. При указании параметров - обе полосы или вертикальная, свойство MaxLength игнорируется

Свойство Aligment(выравнивание)

Это свойство позволяет расположить текст в окне просмотра так, так вам необходимо.

Методы

Рассмотрим некоторые методы работы с полем Memo.

Метод Delete.

Метод удаляет строку с указанным индексом, происходит перенумерация строк (остальные строки автоматически сдвигаются).

```
memEx.Lines.Delete(0); //удаляет самую первую строку, т.е.
                          строку с индексом 0.
```

Пример 1.

Оставьте в поле Memo только нечетные строки.

```
k:=memEx.Lines.Count;
For i:=k-1 Downto 1 Do
   if (i mod 2) <>0 then
         memEx.Lines.Delete(i);
```

Метод Exchange

Данный метод позволяет поменять местами строки с указанными индексами.

```
memEx.Lines.Exchange(0,1);
// меняет местами строки с индексом 0 и 1
```

Метод Move

Данный метод позволяет передвинуть строку с одного места на другое.

Например

```
memEx.Lines.Move(1,5);
```

При выполнении этого метода строка с номером 1 (т.е. вторая в поле, т.к. первая имеет номер 0) будет изъята из текста, все строки, которые следуют за ней, будут подняты на одну строку вверх, и на место пятой строки будет поставлена первая.

ЗАДАНИЯ

Задание 1.

Заполнить поле Memo.

Написать программу, которая позволяет при нажатии на клавишу оставить в поле

- только первые 4 строки, если строк больше 4-х
- 1 строку, если строк не больше 4-х.

Задание 2.

Заполнить поле Memo и окно ввода.

Написать программу, которая позволяет при нажатии на клавишу вывести в заголовок надписи

a) номер строки из поля Memo, которая соответствует строке в окне ввода, или надпись, что аналогичных строк нет;
b) номера строк, которые содержат введённое в поле ввода сочетание символов, или надпись, что строк с данным сочетанием символов нет.

Задание 3.

Заполнить поле Memo текстом, состоящим из нескольких строк, таким образом, чтобы в одной из строк было слово «cat», ещё в одной – «dog».

Написать программу, которая позволяет поменять эти строки местами.

Задание 4.

Заполнить поле Memo.

Написать программу, которая позволяет найти наиболее длинную строку и поставить её в начало.

Задание 5.

Заполнить поле Memo числами (по одному в каждой строке).

Написать программу, которая позволяет все чётные числа увеличить в два раза и перенести во второе поле Memo, а нечётные уменьшить в два раза и перенести в третье поле Memo.

Занятие 15. Компонент Memo (продолжение)

Рассмотрим ещё несколько свойств и методов, которые позволяют работать с полем Memo.

Свойство Clear

Позволяет очистить полностью содержимое поля Memo.

memEx.Lines.Clear

Методы Append и Add

Если поле Memo пустое или было применено свойство ***Clear***, то для заполнения поля Memo можно использовать метод ***Append,*** который добавляет новую строку в конец текста поля Memo.

Пример.

Заполнить поле Memo числами от 1 до 10 по одному на каждой строке.

```
...
memEx.Lines.Clear;
For i:=1 to 10 do
  begin
       memEx.Lines.Append (IntToStr(i));
  end;
```

Аналогично работает и другой метод ***Add***.

```
memEx.Lines.Add ('This example uses A string List.');
```

Как видно из примеров, в качестве параметров у этих методов используются строки.

С помощью метода ***Add*** можно не только добавлять строки в поле просмотра, но и узнать (и вывести, например) номер строки, в которую произошло добавление строки с помощью метода ***Add.***

```
b:=memEx1.Lines.Add(edtEx1.Text);
lblEx1.Caption:=IntToStr(b);
```

В данном примере, текст, который был введен в поле edtEx1, был добавлен в поле memEx1, и в надпись lblEx1 был выведен номер строки, в которую был добавлен текст.

Переносить, сохранять, восстанавливать фрагменты текста поля Memo можно и с помощью Буфера обмена (**Clipboard)**.

Метод Insert

Метод вставляет строку с указанным индексом, происходит перенумерация строк (остальные строки автоматически сдвигаются).

```
memEx.Lines.Insert(2,''); // добавляет пустую строку на место
   строки с индексом 2, которая  сдвигается (не пропадает),
```

Сортировка строк в поле Memo

Сортировка строк осуществляется по следующему алгоритму:

- Находится место минимальной строки (напоминаем, что строки сравниваются посимвольно A<B<C ..<Z и т.д.).
- Меняются местами первая строка и строка, стоящая на месте минимальной.
- Эти операции производятся (N-1) раз, где N – количество строк в поле Memo.

Напишем эту программу.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
begin
   //Для каждой i- ой строки из поля просмотра
   //находим место минимальной строки, начиная с i-ого номера
   // меняем ее с текущей.
end.
```

Описываем необходимые нам переменные, вписываем нужные операторы языка Delphi, описываем используемую функцию нахождения места минимальной строки в поле Memo.

В результате получим следующую программу.

```
function PlMin(numStr:integer):integer;
var
  k,m:integer;
begin
  m:=NumStr;//Пусть строка, стоящая на месте num_str, минимальна
  for k:=m+1 to frmEx1.memEx1.Lines.Count do//Для всех строк,
  //начиная с номера num_str
     if  frmEx1.memEx1.Lines[k]< frmEx1.memEx1.Lines[m] then
     //Если текущая строка меньше минимальной, то
           m:=k;//Переопределяем место минимальной строки
  PlMin:=m;//Значению функции присваиваем место мин-ной строки
  end  ;
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
    i,j:integer;
begin
   for i:=0 to memEx1.Lines.Count do
   //Для каждой строки из поля
     begin
     //просмотра
       j:=plmin(i);
       //находим место мин. строки, начиная с i-ого номера
       memEx1.Lines.Exchange(i,j);//меняем ее с текущей.
     end;
end;
end.
```

ЗАДАНИЯ

Задание 1.

Заполнить первое поле Memo любыми числами (по одному в каждой строке).

Написать программу, которая позволяет при нажатии на клавишу все положительные числа переписать во второе поле Memo, а отрицательные – в третье.

Задание 2.

Заполнить поле просмотра, введя в него список класса. Написать программу, которая позволяет при нажатии на клавишу отсортировать список.

Задание 3.

Заполнить два поля Memo.

Написать программу, которая позволяет при нажатии на клавишу вывести в третье поле Memo те строки, которые есть и в первом и во втором полях просмотра, в противном случае вывести сообщение, что одинаковых строк нет.

Задание 4.

Заполнить поле Memo числами (по одному в каждой строке).

Написать программу, которая позволяет при нажатии на клавишу отсортировать числа.

Занятие 16. Генератор случайных чисел. Константы. Описание собственных типов. Массивы и правила работы с ними

Функция RANDOM

Для того чтобы получить в программе случайное число необходимо использовать функцию ***Random***, которая имеет два формата:

- функция ***Random*** без параметров возвращает **вещественное** случайное число в диапазоне от 0 до 1 (0 – включается, 1 – не включается);
- функция ***Random(N)*** возвращает **целое** случайное число в диапазоне от 0 до N-1, где N — целое.

Если необходимо задать случайное целое число в диапазоне от a до b включительно, то надо воспользоваться формулой:

```
Random(b-a+1)+a.
```

Если необходимо задать случайное вещественное число в диапазоне от a до b (a – включено, b – не включено), то надо воспользоваться формулой:

```
(b-a) * Random+a.
```

В общем случае, при каждом вызове программы получается один и тот же набор случайных чисел.

Для того, чтобы каждый раз при запуске программы получался разный набор случайных чисел, необходимо использовать процедуру ***Randomize.*** Вызов процедуры ***Randomize*** должен быть оформлен таким образом, чтобы она выполнялась только **один раз** в программе.

Именованные константы

Константами называются величины, которые не меняются в процессе выполнения программы. Константы, так же как и переменные, имеют свой тип. Константы бывают неименованные и именованные.

Примеры неименованных констант:

100	**—**	**Integer**
2.7	**—**	**Real**
7.12457e-03	**—**	**Real**
TRUE	**—**	**Boolean**
14.0	**—**	**Real**
-37	**—**	**Integer**
‘asderc’	**—**	**String**

В некоторых случаях удобно использовать именованные константы. Для этого в блоке объявлений программы задаётся имя константы и её значение.

Формат описания именованных констант:

```
Const
   <имя константы> = <значение>;
```

Имя константы задаётся по тем же правилам что и имя переменной. Тип именованной константы определяется автоматически в соответствии с типом её значения.

Именованные константы обычно используют для наглядности и для настройки программы.

Собственные типы

В языке Delphi имеется возможность описывать новые типы.
Пример описания собственных типов:

```
type
   kg = integer;
   cm = integer;
```

```
var
   Weight : kg;
   Size   : cm:
```

Что даёт нам использование собственных типов? Во-первых, лучшее понимание программы, наглядность. Кроме того, если в дальнейшем окажется, что мы неудачно выбрали тип для описания переменных то достаточно поменять тип только в одном месте, а не во всех местах, где мы описывали переменные.

ЗАДАНИЯ

Задание 1.

Поиграем с компьютером в игру «Угадай число».

Пусть компьютер выбирает случайное число из диапазона, вводимого в два окна ввода. В третьем окне ввода мы будем отгадывать это число.

Написать программу, которая позволяет при нажатии на клавишу вывести в заголовок надписи сообщение о том, угадано нами число, или нет. Если нет, то какое оно в сравнении с загаданным (больше или меньше).

И пусть будет ещё кнопка, при нажатии на которую, можно подсмотреть загаданное число.

Занятие 17. Одномерные массивы

Массив – это совокупность данных одного типа, имеющая одно общее имя, но разные индексы и значения. Например, список учащихся класса. Имя каждого элемента данного массива – это «Ученик», в журнале каждому ученику присвоен номер – это индекс элементов массива, а вот кто этот ученик – это значение элемента массива.

Формат описания массива:

```
<имя массива> : Array [<нач.зн>...< кон.знач.индекса>] Of
   <тип эл-в> ;
```

Описание массива включает в себя:

- **имя массива**;
- ключевое слово ***Array***;
- **начальное** и **конечное значение индекса**, которые определяют границы изменения индексов, например: 1..40, -2..2, 0..10. Нижняя граница показывает наименьшее возможное значение индекса, верхняя - наибольшее. Очевидно, что нижняя граница не может превосходить верхнюю. Нижняя граница отделяется от верхней двумя точками. Индекс должен иметь порядковый тип (это такой тип, для элементов которого определён предыдущий и последующий элементы). Из них нам пока известны: все целочисленные типы, логический тип. В качестве индекса могут выступать только константы, то есть размер массива задаётся при написании программы и не может быть изменён в ходе работы программы;

- **тип элементов** массива.

Пример описания массива

```
A: array[1..10]of real;
```

Это массив **A**, состоящий из **10** элементов, каждый из которых является вещественным числом.

Важно, что к элементу массива можно обращаться по индексу. При этом индекс может быть выражением соответствующего типа.

Например

```
X:=5;
A[x+2]:=4.5
```

Здесь элементу массива со значение индекса равным 7 (5+2), присваивается число 4.5.

Использование обращения по индексу позволяет эффективней решать задачи. Например, чтобы присвоить всем элементам массива ноль достаточно написать следующий цикл

```
For i:=1 To 10 do
   A[i]:=0;
```

Для того, чтобы массив можно было передать в качестве параметра в процедуру, он должен быть обязательно описан собственным типом, например

```
Type
    Mas = array[1..30] of integer;
.....
.....
Var
   A : mas;
```

Теперь рассмотрим задачу заполнения массива, содержащего N элементов, значения которых находятся в указанном диапазоне, вывода массива в поле Memo и нахождение максимального элемента.

Запишем алгоритм решение задачи в комментариях.

```
Begin
  //Вводим диапазон chB и chE изменения значений элементов
    массива
  //Заполняем массив A числами из указанного диапазона
  //Выводим массив в поле Memo
  //Находим максимальный элемент массива Amax.
  //Выводим его
End.
```

Сопоставим каждой фразе алгоритма операторы языка Delphi.

```
Begin
 Vvod(chB,chE); //Вводим диапазон chB и chE изменения значений
                  элементов  массива
 Zapoln(a,chB,chE); //Заполняем массив A случ-ми числами в
                      диапазоне от chB до chE
 Vyvod(a,n); //Выводим массив A, состоящий из n элементов.
 Amax:=max(a); //Находим максимальный элемент массива Amax
 lblOtv.Caption:=IntToStr(Amax); //Выводим его
End.
```

Прежде, чем описать необходимые нам для решения задачи процедуры и функции, отметим, что для того, чтобы передавать массив как параметр процедуры, необходимо описать собственный тип массив. Решение задачи будет выглядеть следующим образом.

```
Const
  n=20;
Type
  massiv=array[1..n] of integer;
procedure Vvod(var ch1,ch2:integer);
begin
  ch1:=StrToInt(frmMas.edtCh1.Text);
  ch2:=StrToInt(frmMas.edtCh2.Text);
end;
procedure Zapoln(var a:massiv; chB,chE:integer);
var
  i:integer;
begin
  randomize;
  For i:=1 to n do
     a[i]:=random(chE-chB)+chB;
end;
procedure vyvod(a:massiv; n:integer);
var
  i:integer;
begin
  frmMas.mem_ish.Lines.Clear;
  for i:=0 to n-1 do
     frmMas.memIsh.Lines.Append(IntToStr(a[i+1]));
end;
function max(a:massiv):integer;
var
  i,m:integer;
begin
  m:=a[1];
  for i:=1 to n do
      if a[i]>m then
          m:=a[i];
  max:=m;
end;
procedure TfrmMas.btnClick(Sender: TObject);
```

```
var
   chB,chE:integer;
   Amax : integer;
   A : massiv;
  begin
   Vvod(chB,chE); //Вводим диапазон chB и chE изменения значений
                    элементов массива
   Zapoln(a,chB,chE); //Заполняем массив A случ. числами в
                        диапазоне от chB до chE
   Vyvod(a,n); //Выводим массив A, состоящий из n элементов.
   Amax:=max(a); //Находим максимальный элемент массива Amax.
   lblOtv.Caption:=IntToStr(Amax); //Выводим его
  end;
end.
```

ЗАДАНИЯ

Задание 1.

Ввести с клавиатуры границы диапазона, заполнить массив случайными целыми числами в диапазоне, начальное и конечное значение которого введены в поля ввода.

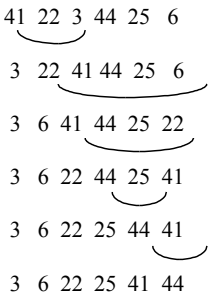
- Вывести его в поле Memo.
- Подсчитать и вывести в соответствующие надписи
  - сумму элементов массива,
  - их среднее значение,
  - количество положительных и отрицательных,
  - найти максимальный и минимальный элементы.
- Разбить массив на два – с положительными и отрицательными элементами – и затем вывести их в два других поля Memo.

Занятие 18. Сортировка массивов. Сортировка простым выбором

Сортировка – это упорядочивание чего-либо по какому-нибудь признаку. Для сортировки элементов массива существует очень много алгоритмов. Мы разберем один из самых простых и понятных методов (алгоритмов) сортировки – сортировка простым выбором.

Метод основан на следующем принципе: если массив надо отсортировать по возрастанию, то выбирается наименьший элемент, он меняется местами с первым элементом. Затем, среди оставшихся n-1 элементов выбирается наименьший элемент, он меняется местами со вторым элементом. Эта операция затем повторяется с оставшимися n-2 элементами, затем с n-3 элементами, пока не останется только один элемент — наибольший.

Пример сортировки простым выбором может быть представлен в следующем виде:



Рассмотрим сортировку массива по возрастанию. Сначала пишем алгоритм.

```
//Вводим диапазон chB и chE изменения значений элементов массива
//Заполняем массив A случайными числами в диапазоне от chB до chE
//Выводим массив A, состоящий из n элементов, в первое поле.
//Сортируем массив A по возрастанию.
//Выводим массив A, состоящий из n элементов, во второе поле.
```

Напишем рядом с каждой строкой алгоритма процедуру с фактическими параметрами.

```
procedure TfrmMas.btnGoClick(Sender: TObject);
var
  chB,chE:integer;
  A : massiv;
begin
  Vvod(chB,chE); //Вводим диапазон chB и chE изменения
                   значений элементов массива
 Zapoln(a,chB,chE); //Заполняем массив A случайными числами в
                      диапазоне от chB до chE
  Vyvod1(a,n); //Выводим массив A, состоящий из n элементов, в
                 первое поле Memo.
  SortMas(a); //Сортируем массив A по возрастанию.
  Vyvod2(a,n); //Выводим массив A, состоящий из n элементов,
                 во второе поле Memo.
end;
```

Первые три процедуры подробно разобраны в предыдущем занятии. Напишем только процедуру сортировки элементов массива по возрастанию (она почти такая же, как и для сортировки поля Memo).

```
procedure SortMas(var a:massiv);
var
  i:integer;
begin
  for i:=1 to n-1 do //Для каждого элемента массива
```

```
        change(a[i],a[NumMin(a,i)]); //Меняем местами текущий
    //элемент и элемент, стоящий на месте минимального,
    //начиная с текущего номера.
end;
```

И теперь перед этой процедурой опишем процедуру обмена местами двух величин целого типа и функцию нахождения места минимального элемента массива, начиная с текущего номера элемента.

```
function NumMin(a:massiv; start:integer):integer;
var
  i,m:integer;
begin
  m:=start;
  for i:=m+1 to n do
     if a[i]<a[m] then
       m:=i;
  NumMin:=m;
end;

procedure change(var one, two:integer);
var
  temp:integer;
begin
  temp:=one;
  one:=two;
  two:=temp;
end;
```

ЗАДАНИЯ

Задание 1.

Ввести с клавиатуры границы диапазона, заполнить массив случайными целыми числами в диапазоне, начальное и конечное значение которого введены в поля ввода.

Вывести его в поле Memo.

Отсортировать массив по убыванию, вывести во второе поле Memo.

Задание 2.

Ввести с клавиатуры границы диапазона, заполнить массив размером N случайными целыми числами в диапазоне, начальное и конечное значение которого введены в поля ввода.

Вывести исходный массив в первое поле Memo.

Все положительные элементы поместить в начало массива, а отрицательные и нулевые в конец (**сортировку не использовать**). Вывести полученный массив во второе поле Memo.

Задание 3.

Ввести с клавиатуры границы диапазона, заполнить массив размеров 2N случайными вещественными числами в диапазоне, начальное и конечное значение которого введены в поля ввода.

Вывести исходный массив в первое поле Memo.

Поменять местами первую половину массива со второй. Вывести полученный массив во второе поле Memo.

Занятие 19. Текстовая таблица (StringGrid)

Для создания таблиц, в ячейках которых располагаются произвольные текстовые строки, предназначен компонент ***StringGrid***.

Компонент ***StringGrid*** находится на вкладке ***Additional*** (Дополнительные). Префикс, который мы будем использовать для имени компонента - ***sgd***. Для примеров возьмём имя компонента ***sgdMy***.

Таблица делится на две части — фиксированную и рабочую.

Фиксированная часть служит для показа заголовков колонок и рядов, а также для ручного управления их размерами. Обычно фиксированная часть занимает одну левую колонку и один верхний ряд таблицы. Однако, с помощью свойств ***FixedCols*** и ***FixedRows*** можно задать другое количество фиксированных колонок и рядов (если эти свойства имеют значение 0, таблица не содержит фиксированной зоны).

Обычно

```
sgdMy.FixedCols:=1; //Количество фиксированных столбцов = 1
sgdMy.FixedRows:=1; //Количество фиксированных строк = 1
```

Рабочая часть — это остальная часть таблицы. Она может содержать произвольное количество колонок и рядов (эти величины могут изменяться программно). Рабочая часть может не умещаться целиком в пределах окна компонента, в этом случае в него автоматически помещаются нужные полосы прокрутки. При прокрутке рабочей области фиксированная область не исчезает, но меняется её содержимое — заголовки колонок и рядов.

Основным свойством компонента является свойство ***Cells*** — набор ячеек, каждая из которых может содержать произвольный текст. Конкретная ячейка определяется парой чисел — номером колонки и номером ряда, на пересечении которых она находится (нумерация начинается с нуля).

Свойство ***Cells*** имеет тип ***String***.

Заполнять ячейки таблицы можно вручную, после запуска программы на выполнение. Для этого необходимо в окне инспектора объектов открыть свойство ***Options***, нажав на ***"+"***, и свойству ***goEditing*** присвоить значение ***True***.

Можно задавать значения ячеек программно с помощью оператора присваивания, обращаясь к каждой ячейке по ее индексам, при этом необходимо помнить, что ***первый индекс – это номер столбца, а второй – номер строки***.

Например:

```
sgdMy.Cells [1,1] := 'Левая верхняя ячейка рабочей зоны';
```

sgdMy.Cells [0,0] := 'Nomera : ';

Строка 'Nomera : ' будет записана в самую (вообще) первую ячейку таблицы, т.е. в первую ячейку фиксированной зоны.

Количество ячеек по каждому измерению определяет пара свойств ***ColCount*** (количество колонок) и ***RowCount*** (количество рядов). Значения этих свойств и, следовательно, размеры таблицы могут меняться как на этапе разработки программы, так и в ходе ее работы, однако их значения должны быть как минимум на единицу больше соответственно значений в свойствах ***FixedCols*** и ***FixedRows***, определяющих размеры фиксированной зоны.

В таблице sgdMy установим количество колонок =3 и количество рядов =5.

```
sgdMy. ColCount:=3;
sgdMy. RowCount:=5;
```

Для задания цвета ячеек фиксированной области используется свойство ***FixedColor***, а для задания цвета ячеек рабочей области используется свойство ***Color.***

В качестве примера рассмотрим задачу, в которой по нажатию кнопки создаётся текстовая таблица, количество столбцов, строк и фиксированных столбцов и строк вводятся в четыре поля ввода. По нажатию второй кнопки фиксированная зона закрашивается в зелёный цвет, а по нажатию третьей кнопки рабочая область закрашивается в красный цвет.

```
Procedure VVod(var n1, n2, n3, n4: integer);
Begin
   n1:=StrToInt(frmTab.edtLine.Text);
   n2:=StrToInt(frmTab.edtStolb.Text);
   n3:=StrToInt(frmTab.edtFline.Text);
   n4:=StrToInt(frmTab.edtFstolb.Text);
End;
Procedure MakeTab(n1, n2, n3, n4: integer);
Begin
   frmTab.sgdMy.RowCount:=n2;
   frmTab.sgdMy.ColCount:=n1;
   frmTab.sgdMy.FixedCols:=n4;
   frmTab.sgdMy.FixedRows:=n3;
end;
procedure TfrmTab.btnTablClick(Sender: TObject);
```

```
var
  nl, ns, nfl, nfs:integer;
begin
  Vvod(nl, ns, nfl, nfs); //Ввести количество строк nl,
  //столбцов ns, фикс. строк nfl и фикс. столбцов  nfs
  MakeTab(nl, ns, nfl, nfs); //Создать таблицу с заданным
  //количеством строк и столбцов
end;
procedure TfrmTab.btnCelRedClick(Sender: TObject);
begin
  frmTab.sgdMy.Color:=clRed;
end;
procedure TfrmTab.btnFCGreenClick(Sender: TObject);
begin
  frmTab.sgdMy.FixedColor:=clGreen;
end;
```

Некоторые Полезные Свойства компонента StringGrid

Свойство	Описание
`BorderStyle: TBprderStyle;`	Определяет рамку компонента: bsNone — нет рамки; bsSingle — рамка толщиной 1 пиксел
`ColCount: Longlnt;`	Содержит количество колонок таблицы
`DefaultColWidth: Integer;`	Содержит значение ширины колонки, заданное по умолчанию
`RowCount:: Longlnt;`	Содержит количество рядов таблицы
`DefaultRowHeight: Integer;`	Содержит значение высоты рядов, заданное по умолчанию
`Color: TColor;`	Определяет цвет рабочей зоны
`FixedCols: Integer;`	Определяет количество колонок фиксированной зоны
`FixedRows: Integer;`	Определяет количество рядов фиксированной зоны
`FixedColor: TColor;`	Определяет цвет фиксированной зоны
`GridHeight: Integer;`	Содержит значение высоты таблицы
`GridWidth: Integer;`	Содержит значение ширины таблицы
`GridLineWidth:`	Определяет толщину линий, расчерчивающих таблицу

ЗАДАНИЯ

Задание 1.

Создать таблицу StringGrid, для этого:

В полях ввода задать:

- Количество строк
- Количество столбцов
- Количество фиксированных строк
- Количество фиксированных столбцов

Для изменения цвета фиксированной зоны создать 3 командных кнопки (для разных цветов).

Для изменения цвета рабочей зоны создать 3 командных кнопки (для разных цветов)

Создать поле ввода и 2 командные кнопки.
1 команда кнопка выводит строку с номером, заданным в поле ввода.
2 команда кнопка выводит столбец с номером, заданным в поле ввода.

Вывод строки или столбца осуществляется в поле Memo.

Задание 2.

Создать таблицу StringGrid.

При проведении экспериментальной работы по физике задаются: количество измерений, цена деления прибора, показания прибора в делениях.

В полях ввода задать:

- Количество измерений (от 5 до 10).
- Цена деления.

Показания прибора в делениях заносятся в таблицу с клавиатуры.

Вычислить и занести в таблицу результаты всех измерений, а также минимальное и максимальное значения.

Например, количество измерений = 5, цена деления = 10, тогда таблица имеет вид:

Номер измерения	**Цена деления**	**Измерения**	**Результат**	**Максимальное/ минимальное значения**
1	10	83	830	Максимальное
2	10	51	510	
3	10	67	670	
4	10	49	490	Минимальное
5	10	75	750	

Занятие 20. Особенности решения задач с компонентом StringGrid.

Нахождение количества элементов с данным свойством.

Задачи на нахождение количества элементов с данным свойством в компоненте ***StringGrid*** решаются практически так же, как и для поля Мемо и для одномерных массивов. Необходимо лишь добавить второй цикл ко второму индексу.

Задача 1. Найти количество отрицательных элементов в каждой строке заданной целочисленной матрицы.

Решение.

Заполним ячейки компонента ***StringGrid*** с помощью генератора случайных чисел. Учтём, что нумерация строк и столбцов начинается с нуля. Для вывода результата будем использовать поле Мемо.

Напишем программу на языке Delphi.

На форме разместим два поля Edit для задания количества строк и столбцов, компонент ***StringGrid***, поле ***Memo*** для вывода результата и две кнопки:

1 - создание и заполнение компонента ***StringGrid***,
2 – подсчёт количества отрицательных элементов в строках.

Опишем соответственно две процедуры обработки события нажатия на кнопку.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,j,n,m : integer;
begin
  n:=strtoint(edit1.Text);
  m:=strtoint(edit2.Text);
  stringgrid1.RowCount:=n;
  stringgrid1.ColCount:=m;
  for i:=0 to n-1 do
    for j:=0 to m-1 do
      stringgrid1.Cells[j,i]:=inttostr(random(100)-50);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  i,j,n,m,k : integer;

begin
  n:=stringgrid1.RowCount;
  m:=stringgrid1.ColCount;
  for i:=0 to n-1 do
    begin
      k:=0;
      for j:=0 to m-1 do
        if strtoint(stringgrid1.Cells[j,i])<0 then
```

```
          k:=k+1;
        memo1.Lines.Append(inttostr(k));
      end
end;
```

Задача 2. Заменить все отрицательные элементы матрицы на противоположные.

Варианты решения:

- необходимо умножить значение отрицательного элемента на (-1);
- надо взять значения элементов по абсолютной величине;
- можно просто поставить знак "минус" перед элементом матрицы, если он - отрицательный.

Программа 1.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i,j,n,m : integer;
begin
  n:=stringgrid1.RowCount;
  m:=stringgrid1.ColCount;
  for i:=0 to n-1 do
    begin
      for j:=0 to m-1 do
        if strtoint(stringgrid1.Cells[j,i])<0 then
          stringgrid1.Cells[j,i]:=inttostr((-
1)*strtoint(stringgrid1.Cells[j,i]));
    end
end;
```

Программа 2.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i,j,n,m : integer;

begin
  n:=stringgrid1.RowCount;
  m:=stringgrid1.ColCount;
  for i:=0 to n-1 do
    begin
      for j:=0 to m-1 do

stringgrid1.Cells[j,i]:=inttostr(abs(strtoint(stringgrid1.Ce
lls[j,i])));
    end
end;
```

Программа 3.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i,j,n,m : integer;
begin
  n:=stringgrid1.RowCount;
```

```
  m:=stringgrid1.ColCount;
  for i:=0 to n-1 do
    begin
      for j:=0 to m-1 do
        if strtoint(stringgrid1.Cells[j,i])<0 then
          stringgrid1.Cells[j,i]:=inttostr(-
strtoint(stringgrid1.Cells[j,i]));
    end
end;
```

Задача 3. Определить, является ли данная квадратная матрица симметричной относительно своей главной диагонали.

Решение.

Используем следующее правило: если для всех ***i = 1, 2, ..., n и j = 1, 2, ..., n,*** причём ***i>j,*** выполняется равенство ***StringGrid1.Cells[i,jj= StringGrid1.Cells [j, i],*** то матрица является симметричной. Поэтому можно составить следующую программу:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,j,n : integer;
begin
  n:=strtoint(edit1.Text);
  stringgrid1.RowCount:=n;
  stringgrid1.ColCount:=n;
  for i:=0 to n-1 do
    for j:=0 to n-1 do
      stringgrid1.Cells[j,i]:=inttostr(random(2));
  label2.Caption:='';
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  i,j,n,m : integer;
  simm : boolean;
begin
  n:=stringgrid1.RowCount;
  simm:= True;
  {Предположим, что матрица симметрична}
  i:=2;
  While simm and (i < n) Do
   Begin
    j:=1;
    While (j < i) and (stringgrid1.Cells[j,i] =
stringgrid1.Cells[i,j]) Do
      j:=j+1;
    simm:=(j=i);
    i:=i+1
```

```
    End;
    if simm then
      label2.Caption:='матрица симметрична'
    else
      label2.Caption:= 'матрица не симметрична'
end;
```

Задача 4. Заполнить матрицу размером n x m следующим образом (по "змейке").

1 2 3 4 5 ... n
... n+2 n+1

Решение.

Для того чтобы заполнить матрицу указанным способом, надо вывести правило заполнения. В данном случае правило будет таким: если ряд нечётный (то есть, когда номер строки - чётное число), то ***stringgrid1.Cells[j,i]:= inttostr(i*m + j+1)***, иначе (то есть когда строка чётная) ***stringgrid1.Cells[j,i]:= inttostr((i+1)*m-j).***

В соответствии с этим правилом составляем программу заполнения матрицы:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i,j,n,m : integer;
begin
  n:=strtoint(edit1.Text);
  m:=strtoint(edit2.Text);
  stringgrid1.RowCount:=n;
  stringgrid1.ColCount:=m;
  for i:=0 to n-1 do
    for j:=0 to m-1 do
      if  i mod 2 =0 then
        stringgrid1.Cells[j,i]:=  inttostr(i*m + j+1)
      else
        stringgrid1.Cells[j,i]:=  inttostr((i+1)*m-j)
end;
```

ЗАДАНИЯ

Задание 1.

Заполнить матрицу размером n x n 0 и 1 по следующей схеме (пример, матрица размером 5 x 5):

1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0

Задание 2

Заполнить матрицу размером n x n 0 и 1 по следующей схеме (пример матрица размером 5 x 5):

0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 1 1 1 0
1 1 1 1 1

Задание 3

Заполнить матрицу размером m x n случайными числами в диапазоне, начальное и конечное значение которого вводятся с клавиатуры.

Задание 4

Заполнить матрицу размером m x n случайными числами в диапазоне, начальное и конечное значение которого вводятся с клавиатуры.

Найти в каждом столбце минимальный элемент и поменять его местами с элементом, стоящим на главной диагонали.

Задание 5

Заполнить матрицу размером m x n случайными числами в диапазоне, начальное и конечное значение которого вводятся с клавиатуры.

Для каждого столбца матрицы найти и вывести произведение элементов с нечётными индексами строки.

Задание 6

Заполнить матрицу размером m x n случайными числами в диапазоне, начальное и конечное значение которого вводятся с клавиатуры.

Найти и вывести произведение положительных элементов на нечётных столбцах матрицы.

Занятие 21. Двухмерные массивы

Как мы уже знаем, ***массив*** – это упорядоченная совокупность элементов одного типа, занимающая непрерывную область в памяти компьютера, имеющих одно имя и разные индексы. Все элементы любых массивов следуют (занимают ячейки памяти) друг за другом.

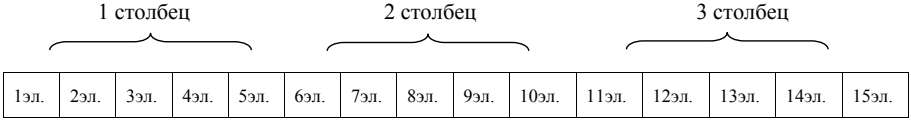
1 элемент	2 элемент	3 элемент	…	N элемент

При работе просто с последовательностью данных (числами, строками или другими типами данных) мы используем одномерный массив.

(По сути все массивы можно рассматривать как одномерные. Другое представление массива, такое как двухмерные массивы и т.д. необходимо только для удобства представления данных, для удобства решения задач). Элементы двухмерных массивов имеют 2 индекса, N-мерных – N индексов.

При решении задач, требующих табличной организации данных, удобнее использовать двухмерный массив.

Например, исходные данные необходимо представить (а также выполнить различные действия с ними) в виде 3-х столбцов по 5 строк в каждом.



все элементы

Как видно мы получили массив, состоящий из трёх столбцов, каждый из которых в свою очередь состоит из 5 строк. То есть, можно сказать, что мы получили массив массивов.

Очевидно, что в этом случае, массив столбцов будет занесён в компонент ***StringGrid*** следующим образом:

	1 столбец	2 столбец	3 столбец
1 строка	1 эл.	6 эл.	11 эл.
2 строка	2 эл.	7 эл.	12 эл.
3 строка	3 эл.	8 эл.	13 эл.
4 строка	4 эл.	9 эл.	14 эл.
5 строка	5 эл.	10 эл.	15 эл.

Опишем массив

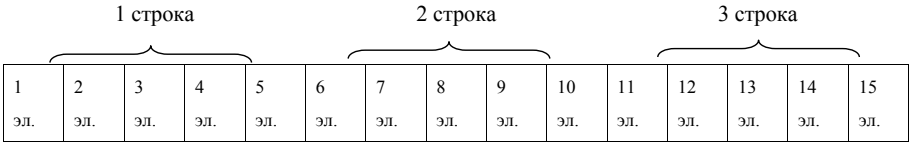
Сначала опишем массив столбцов, который нам нужен

```
Const
  m=3;
Type
  mas_stolb=array[1..m] of mas_strok;
```

Добавим описание массива строк

```
Const
  m=3;
  n=5;
Type
  mas_strok=array[1..n] of integer;
  mas_stolb=array[1..m] of mas_strok;
```

Разберём ещё один пример, в котором исходные данные необходимо представить (а также выполнить различные действия с ними) в виде 3-х строк по 5 столбцов в каждой.



все элементы

Как видно мы получили массив, состоящий из трёх строк, каждый из которых в свою очередь состоит из пяти столбцов. То есть, можно сказать, что мы получили массив массивов.

Очевидно, что в этом случае, массив столбцов будет занесён в компонент ***StringGrid*** следующим образом:

	1 столбец	2 столбец	3 столбец	4 столбец	5 столбец
1 строка	1 эл.	2 эл.	3 эл.	4 эл.	5 эл.
2 строка	6 эл.	7 эл.	8 эл.	9 эл.	10 эл.
3 строка	11 эл.	12 эл.	13 эл.	14 эл.	15 эл.

Опишем массив

Сначала опишем массив строк, который нам нужен

```
Const
  n=3;
Type
  mas_strok=array[1..n] of mas_stolb;
```

Добавим описание массива столбцов

```
Const
   n=3;
   m=5;
Type
   mas_stolb =array[1..m] of integer;
   mas_strok =array[1..n] of mas_stolb;
```

Как видно из этих примеров, двухмерный (в общем случае N-мерный) массив можно представить как массив массивов. Причём двухмерный массив – это массив строк (строка – одномерный массив) или массив столбцов (столбец – одномерный массив).

Обращаться к элементам массива (например, с именем A) будем следующим образом

A[1][4] или A[3][5] или A[I][j] ,

где I – это текущий номер элемента двухмерного массива, а j – текущий номер элемента одномерного, составляющего двухмерный.

В качестве примера рассмотрим следующую задачу.

Заполнить двухмерный массив, как массив, состоящий из M столбцов по N элементов в каждом, случайными числами от -10 до 10. Вывести его в текстовую таблицу. Количество строк и столбцов фиксированной зоны задать равными 0.

Сначала опишем массив.

```
Const
   n=3;
   m=6;
Type
   mas_1=array[1..n] of integer;
   mas_2=array[1..m] of mas_1;
```

Теперь напишем алгоритм решения задачи.

```
Var
   C:mas_2;
Begin
   Zapol(c);
   //Заполнить массив C, состоящий из M столбцов и N строк
   Vyvod(c); //Выведем массив C в текстовую таблицу
end;
```

А теперь опишем необходимые нам процедуры. Опишем процедуру заполнения массива столбцов.

```
procedure Zapol(var mas:mas_2);
   var i:integer;
begin
   for i:=1 to m do //Для I принимающего значения  от 1 до M
        zap_st(mas[i]); //Заполни столбец i-ый массива
  end;
```

Опишем, как заполняется один столбец, состоящий из N элементов (строк) случайными числами в диапазоне от -10 до 10.

```
procedure zap_st(var ma_1:mas_1);
   var
        j:integer;
begin
   Randomize;
   for j:=1 to n do  //Для j принимающего значения от 1 до n
        ma_1[j]:=Random(21)-10; //Элементу массива с номером j
   //присвоим случайное значение из диапазона от -10 до +10
 end;
```

Опишем процедуру вывода полученного массива в текстовую таблицу.

```
procedure vyvod(mas:mas_2);
   var
        i,j:integer;
begin
   frm2mas.sgdMy.FixedCols:=0; //Количество фиксированных
   //столбцов
   frm2mas.sgdMy.FixedRows:=0; //Количество фиксированных
   строк
   frm2mas.sgdMy.ColCount:=m;  //Столбцы
   frm2mas.sgdMy.RowCount:=n;  //Строки
   for i:=1 to m do
       for j:=1 to n do
            frm2mas.sgdMy.Cells[i-1,j-1]:=IntToStr(mas[i][j]);
end;
```

Посмотрим, что изменится, если нам нужно вывести массив построчно, т.е. как бы перевернуть каждый столбец на 90 градусов. Тогда у нас будет 3 строки по 6 элементов в каждой. При выводе необходимо помнить, что у ячейки текстовой таблицы сначала идёт индекс столбца, а потом строки.

Процедура вывода будет выглядеть следующим образом.

```
procedure vyvod(mas:mas_2);
   var
        i,j: integer;
begin
   frm2mas.SgdMy.ColCount:=n; //n – количество строк массива
   frm2mas.SgdMy.RowCount:=m; //m – количество столбцов
   массива
   for i:=1 to n do
       for j:=1 to m do
          frm2mas.SgdMy.Cells[i-1,j-1]:=IntToStr(mas[j][i]);
end;
```

ЗАДАНИЯ

Задание 1.

Заполнить случайными целыми числами двухмерный массив, состоящий из M столбцов по N элементов в каждом.

Диапазон изменения случайных чисел задавать в полях ввода ***Edit***.

Вывести массив в созданную текстовую таблицу ***StringGrid*** (число строк и столбцов фиксированной зоны установить равными 0).

В поле вывода ***Memo*** вывести:

- сумму всех элементов массива;
- максимальный и минимальный элементы массива;
- сумму элементов каждого столбца.

Всё вывести с поясняющим текстом.

Например:

Сумма всех элементов=234
Максимальный элемент = 68
Минимальный элемент = 5
Сумма элементов 1 столбца = 94
Сумма элементов 2 столбца = 43
и т.д.

Задание 2.

Заполнить случайными вещественными числами двухмерный массив, размерности M на N элементов.

Диапазон изменения случайных чисел задавать в полях ввода ***Edit***.

Создать текстовую таблицу ***StringGrid***, пронумеровать в ней фиксированную область.

Вывести массив в созданную текстовую таблицу ***StringGrid*** .

Посчитать сумму элементов каждой строки и вывести в дополнительную ячейку в каждой строке.

Занятие 22. Дата и время

Для того, чтобы удобно производить различные действия с датами и временем в Lasarus существует специальный тип данных, который так и называется TDateTime (тип дата-время).

Описываются переменные следующим образом:

```
Var
   a, b: TDateTime;
```

Для представления переменных данного типа в компьютере используется одно вещественное число с фиксированной точкой, которое занимает 8 байт. Целая часть такого числа показывает, сколько дней прошло с 30 декабря 1899 года. Т.е. если целая часть числа равна 1, то это 31 декабря 1899 года, если 2, то это 1 января 1900 года и т.д. Если число отрицательное, то даты отсчитывается в обратном порядке. Например, -1 означает 29 декабря 1899 года, -2 – это 28 декабря 1899 года и т.д.

Дробная часть числа показывает время, а именно, какая часть суток прошла от 0 часов текущей даты. Например, 1,25 – это 31 декабря 1899 года 6 часов утра (0,25 – это четвертая часть от 24 часов).

Работая с датами, мы записываем их как строку. В Lasarus стандартные средства дают перевод числа в дату только в нашей эре, т.е. от 1 января 0 года от Рождества Христова.

Процедуры работы с датами.

Рассмотрим процедуры работы с датами.

DecodeDate(Date:TDateTime; var Year, Month, Day:Word); - декодирует число в дату, т.е. извлекает из числа год, месяц и день в качестве целых чисел. Тип Word – это положительные целые числа.

Например, если в переменной A содержится число равное дате 1 марта 2008 года, то применение процедуры Decode(A, y,m,d) даёт y=2008, M=3, d=1.

DecodeTime(Time:TDateTime; var Hour, Min, Sec, MSec:Word); возвращает часы, минуты, секунды и миллисекунды (т.е. тысячные доли секунды).

Функции работы с датами

Рассмотрим функции работы с датами.

Пусть у нас имеются три переменных типа TDateTime.

```
Var
   a,b,c:TDateTime;
Begin
   a:=Date; //Возвращает текущую дату
              (в виде вещественного числа)
   b:=Time; //Возвращает время от 0 часов 0 минут 0 секунд
   c:=Now;  //Возвращает и дату и время (текущие)
End;
```

После выполнения данной программы в переменной ***a*** будет находиться текущая дата в виде вещественного числа, в переменной ***b*** – текущее время, а в переменной ***c*** – и текущая дата и текущее время.

Естественно, что эти функции берут дату и время из часов в компьютере, т.е. если установки даты и времени в компьютере не правильные, то эти функции возвратят нам эти неправильные дату и время.

Существуют функции, которые переводят дату и время в строку.

TimeToStr(t:TDateTime): String; -переводит время в строку, т.е. из переменной t берется время в виде ‘hh:mm:ss’, в качестве разделителя используются настройки Windows (у нас это двоеточие).

DateToStr(t:TDateTime): String; -возвращает дату как строку в виде ***‘dd:mm:yyyy’***, например ***'03:11:2008'*** при этом разделитель и порядок (дата-месяц или месяц-дата) зависит от настроек Windows. Год – четырёхзначное число.

DateTimeToStr(t:TDateTime): String; -возвращает дату и время в виде ***‘dd.mm.yyyy hh:mm:ss’***, в качестве разделителя используются настройки Windows

Также существуют функции для перевода строки в дату, время или и в дату и во время.

```
StrToTime(S:String):TDateTime;
StrToDate(S:String):TDateTime;
StrToDateTime(S:String):TDateTime;
```

При работе с этими функциями существуют следующие особенности строк на входе.

- При вводе времени можно не указывать секунды. Например '9:10' будет по умолчанию воспринято как '9:10:00'.
- Данные должны быть корректные.
- Если год указан двумя цифрами, то он будет взят из промежутка 1930-2030.
- Можно не указывать год, тогда год будет взят текущий.
- Нельзя не указывать день и месяц.

Рассмотрим ещё несколько функций для работы с датами и временем.

`DayOfWeek(Date : TDateTime): Word` – возвращает номер дня недели от 1 до 7 (1 – воскресенье и т.д.);

`DecodeDateFully(Date : TDateTime; var Year, Month, Day, DOW : Word): Boolean` – декодирует дату (DOW – номер дня недели), возвращает **True** для високосного года;

`EncodeDate( var Year, Month, Day: Word): TDateTime` – кодирует дату;

`EncodeTime(Hour, Min, Sec, Msec: Word) : TDateTime` – кодирует время

Действия с датами

- Две даты можно вычитать, при этом целая часть разности – это количество дней между датами, например 05.02.2008 – 01.02.2008 = 4 (дня)
- Складывать две даты бессмысленно, но можно к дате прибавить или вычесть целое число, в этом случае мы получим новую дату, отстоящую на соответствующее количество дней вперёд или назад.
- Складывать время можно следующим образом:
  Например, узнать, сколько времени будет через полтора часа можно двумя способами

Time+1,5/24
или
Time+StrToTime('1:30')

ЗАДАНИЯ

Задание 1.

В текстовую таблицу StringGrid заносятся фамилия, дата рождения и дата получения премии Нобелевских лауреатов. Определить фамилию самого молодого лауреата.

Задание 2.

Предположим, что в некоторой школе все перемены длятся одинаковое время, и все уроки длятся одинаковое время. Необходимо, по заданным в полях ввода времени начала первого урока, длительности урока, длительности перемены и количеству уроков составить расписание звонков в этой школе в виде:

	Время начала	**Время конца**
Урок 1	9:00	9:40
Перемена	9:40	9:50
Урок 2	9:50	10:30
Перемена	10:30	10:40
. . .		

Задание 3.

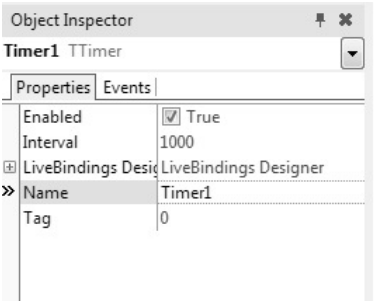
В поле ввода вводится дата. Посчитать сколько полных лет, месяцев и дней отделяют введённую дату от текущей. Результат вывести в надписи.

Занятие 23. Таймер.

Иногда необходимо, чтобы некоторые действия в программе происходили через определённый промежуток времени. Рассмотрим следующую задачу. Пусть через некоторое время (например, через 5 секунд) после запуска программы форма должна окраситься в красный цвет.

Для решения этой задачи необходимо воспользоваться компонентом ***Timer***, который расположен на вкладке ***System***.

Разместим его на форме в любом месте (при запуске программы его не будет видно). Посмотрим на инспектор объектов и увидим, какие свойства есть у этого компонента



Кроме имени компонента нас будут интересовать ещё 2 свойства - ***Enabled*** и ***Interval***. Первое из них отвечает за состояние таймера: если ***true***, то таймер запущен, ***false*** – выключен. Свойство ***Interval*** указывает в миллисекундах время срабатывания таймера. По умолчанию там стоит ***1000***, то есть ***1 секунда***.

В нашем случае установим значение 5000 (5 секунд).

Щёлкнем по таймеру на форме – создастся процедура обработки события срабатывания таймера. В ней напишем оператор изменения цвета формы. Получим

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=clRed;
end;
```

После запуска программы увидим, что через некоторое время форма окрасится в красный цвет.

Подправим программу. Разместим на форме кнопку. В процедуру обработки нажатия кнопки запишем следующий оператор

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  timer1.Enabled:=true;
end;
```

Свойство таймера ***Enabled*** установим в состояние ***false***. Теперь после запуска программы ничего не происходит. Однако, стоит нам нажать на кнопку и через 5 секунд форма окрасится в красный цвет. То есть, когда при нажатии кнопки свойство ***Enabled*** перешло в состояние ***true***, то включился таймер и, через отведённые ему 5 секунд, запустилась процедура обработки события срабатывания таймера.

Исправим процедуру обработки события таймер следующим образом.

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=256*256*random(256)+256*random(256)+random(256);
end;
```

Здесь в правой части оператора присваивания задаётся случайный цвет.

Дополним и процедуру обработки нажатия кнопки

```
  procedure TForm1.Button1Click(Sender: TObject);
  begin
    timer1.Enabled:=true;
    timer1.Interval:=1500;
  end;
```

Здесь мы установили интервал срабатывания таймера в 1,5 секунды.

Запустим программу и кликнем по кнопке. Что мы видим? Каждые 1,5 секунды цвет формы меняется произвольным образом. То есть запущенный таймер, вызывает процедуру обработки события срабатывания таймера постоянно через заданный интервал. Чтобы остановить таймер, необходимо установить свойство ***Enabled*** в состояние ***false***.

Добавим на форму ещё одну кнопку и в процедуру обработки напишем

```
  procedure TForm1.Button2Click(Sender: TObject);
  begin
    timer1.Enabled:=false;
  end;
```

Теперь, нажимая первую кнопку, мы будем запускать переключение цвета, а нажимая вторую – останавливать.

Если мы хотим, чтобы таймер сработал только один раз, то необходимо в процедуру обработки срабатывания таймера вписать оператор, переводящий свойство ***Enabled*** в состояние ***false***. То есть, если мы хотим, чтобы через 1,5 секунды после нажатия кнопки цвет формы изменился однократно, то процедура обработки должна быть такой

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.Color:=256*256*random(256)+256*random(256)+random(256);
timer1.Enabled:=false;
end;
```

Для того, чтобы таймер выключился через заданный промежуток времени, необходимо добавить второй таймер. Например, чтобы переключение цветов закончилось

через 20 секунд, добавим второй таймер (не забудем при этом установить изначально свойство ***Enabled*** в состояние ***false***), в процедуру обработки которого запишем

```
procedure TForm1.Timer2Timer(Sender: TObject);
begin
  timer1.Enabled:=false;
  timer2.Enabled:=false;
end;
```

Исправим процедуру обработки нажатия первой кнопки и процедуру обработки первого таймера следующим образом

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  timer1.Enabled:=true;
  timer1.Interval:=1500;
  timer2.Enabled:=true;
  timer2.Interval:=20000;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
   form1.Color:=256*256*random(256)+256*random(256)+random(256);
end;
```

Запустим программу. После нажатия первой кнопки цвет формы начинает меняться с интервалом 1,5 секунды и через 20 секунд изменение цвета прекращается.

С помощью таймеров можно создавать и движущиеся объекты. Рассмотрим в заключении ещё один пример.

Разместим на форме в левой её части какой-нибудь компонент, например, поле ввода. Исправим программу обработки первого таймера следующим образом

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   edit1.Left:=edit1.Left+15;
end;
```

Запустим программу и нажмём первую кнопку. Мы видим, что поле ввода движется по форме слева направо.

Таким образом, использование таймеров позволяет отмерять различные промежутки времени при работе программы, а кроме того, динамически изменять различные параметры и создавать движущиеся объекты.

ЗАДАНИЯ

Задание 1.

Используя компонент таймер, напишите программу, которая «рисует» звёздное небо. Причём звезды зажигаются и через некоторое время гаснут.

Задание 2.

Напишите программу, которая создаст следующий «штрафной» таймер.

В компонент ***Edit*** вводится время. При нажатии на кнопку начинается обратный отсчёт времени (посекундно). Как только время дошло до 0, цвет фона компонента ***Edit*** становится красным, и начинается отсчёт штрафного времени (увеличение).

Задание 3.

Напишите программу, которая создаёт часы, показывающие текущее время в цифровом формате, а ниже указывают текущую дату и день недели.

Задание 4.

Написать программу, позволяющую имитировать стрелочные часы. Секундная стрелка должна менять положение каждую секунду, минутная - каждую минуту, часовая - каждый час (можно сделать, чтобы часовая стрелка перемещалась плавно, пропорционально пройдённой части часа).

.

Занятие 24. Файлы.

Текстовые файлы.

Запись данных в текстовый файл.

В современном мире любая программа тем или иным способом связана с внешними файлами.

Файл – это именованная область на диске (носителе информации), в которой записана какая-то информация.

Это могут быть как специальные файлы (например, файлы стандартных библиотек), так и файлы данных, получающихся в результате работы программы. Все файлы имеют свой специфический формат. Эта специфика зависит от типа кодировки информации в файле.

Широкое распространение получили ***текстовые файлы***.

Текстовые файлы состоят из строк (одной или нескольких) произвольной длины в которых могут быть записаны любые символы. Каждая строка текста завершается специальным символом, означающим конец строки, а файл заканчивается символом конца

файла. Одной из особенностей текстовых файлов является то, что их можно просматривать и редактировать с помощью различных текстовых редакторов, например редактора ***Notepad*** (***Блокнот***).

Текстовый файл представляет собой совокупность связанных данных, рассматриваемых как единое целое и имеющих общее имя. У текстовых файлов принято ставить расширения ***.txt***, однако наличие данного расширения не гарантирует того, что файл текстовый, равно как и не обязательно то, что текстовый файл будет иметь расширение ***txt***. Приведём примеры стандартных расширений файлов, которые по типу являются текстовыми:.***ini, .log, .inf, .dat***, ***.bat***. Мы будем использовать расширение ***.txt***.

Ещё одна особенность текстовых файлов заключается в способе доступа к данным, которые в них содержатся. Текстовый файл является файлом последовательного доступа.

В файлах последовательного доступа данные располагаются в том порядке, в каком они поступили в файл – последовательно (друг за другом). Для поиска требуемый данных необходимо последовательно просматривать весь файл - от начала до нужных данных. Следовательно, время доступа к данным находится в прямой зависимости от местоположения в файле.

Такие файлы целесообразно использовать в том случае, если при обращении к файлу обрабатываются почти все данные, а их содержимое меняется достаточно редко. Основной недостаток файлов с последовательным доступом заключается в трудности обновления существующих данных, замены данных на новые, вставки новых данных.

Рассмотрим работу с текстовыми файлами в среде ***Delphi***.

Любая программа не может непосредственно обратиться к диску, обращение происходит только через переменные, которые хранятся в оперативной памяти. Поэтому для работы с файлами нам нужна файловая переменная, которая и будет передаточным звеном между программой и диском. Описывается файловая переменная следующим образом:

```
Var
    f:TextFile;
```

В программе необходимо поставить в соответствие файловую переменную и файл, который будет создан или уже существует на диске. Это делается при помощи процедуры ***AssignFile,*** которая имеет два параметра. В качестве первого параметра указывается файловая переменная, а второго – строка, прописывающая локальный или полный путь к файлу.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
begin
   AssignFile(f,'my.txt');
end;
```

В данном примере мы поставили в соответствие файловую переменную ***f*** и файл с именем ***my.txt***, который находится (или будет находиться) в текущей директории (в директории, из которой запущена программа). Это не полное имя файла. Если мы полностью указываем путь к файлу (если, например, файл находится не в текущей директории), то это будет полное имя файла.

Внимание: связывание файловой переменной с именем файла с помощью оператора ***AssignFile*** не проверяет, существует такой файл или нет, и не открывает существующий файл, и не создаёт новый файл

В текстовый файл информацию можно записывать или считывать из файла. Причём можно делать только что-то одно. Нельзя одновременно писать и читать.

Чтобы производить какие-либо операции с файлом, его необходимо открыть. Соответственно открыть его можно либо для записи в файл, либо для чтения из файла.

Рассмотрим сначала, как можно писать в файл.

Чтобы открыть файл для записи, необходимо воспользоваться оператором ***Rewrite(f)***, где ***f*** - файловая переменная.

При этом, на диске создаётся пустой файл с именем, указанным в операторе ***AssignFile.*** Если файл с таким именем уже был, то его содержимое уничтожается.

Теперь в файл можно записывать информацию. Для записи данных в файл используются операторы ***write*** и ***writeln***. Эти операторы работают почти так же, как и при выводе на экран, только в начале необходимо указывать файловую переменную, чтобы указать, что вывод будет не на экран, а в соответствующий файл.

Ниже приведён пример процедуры, которая записывает различную информацию в текстовый файл с именем '***пример.txt***'

```
Procedure ZapFile;
var
    f:TextFile;
    x,y:integer;
    Ok:boolean;
 begin
   AssignFile(f, 'пример.txt');
       ReWrite(f);
       x:=10;
       y:=7;
       Writeln(f,x); //В файл в первую строку запишется 10,
       //курсор перейдёт на вторую строку.
       Write(f,x+2); ); //В файл во вторую строку запишется
                    12 (10+2), и курсор останется в конце
                    второй строки
       Write(f,'Привет'); ); //В файл в конец второй строки
                     (там, где стоит курсор) запишется слово
                     «Привет», курсор останется в конце
                     второй строки.
       Write(f,x,y); //В файл в конец второй строки (там, где
                  стоит курсор) запишется 10 и 7 без пробела,
                  курсор останется в конце второй строки.
       Writeln(f,x,y); ); //В файл в конец второй строки
                    (там, где стоит курсор) запишется 10 и 7
                    без пробела, курсор переместится на третью
                    строку.
       Ok:=5>7;
```

```
10
12Привет107107
FALSE
10  7
x=10
```

```
        Writeln (f,Ok); //В файл в третью строку запишется
                        значение переменной Ok - False, курсор
                        переместится на следующую (четвертую)
                        строку.
    Writeln(f,x,'  ',y); //В файл в четвёртую строку
                запишется значение переменных x - 10 и y - 7
                через два пробела, курсор переместится на
                пятую строку.
        Writeln(f,'x=',x); //В файл в четвёртую строку
                           запишется x=10.
        CloseFile(f);
end;
```

Так же можно заметить, что после всех операций вывода в файл стоит процедура ***CloseFile(f).*** Эту процедуру **необходимо всегда использовать** после окончания работы с файлом, Она закрывает файл. Если убрать эту процедуру, то часть информации будет потеряна.

Как было сказано выше, оператор ***Rewrite*** всегда создаёт новый файл. Однако бывает необходимо дописать информацию к уже имеющемуся файлу. В этом случае файл необходимо открыть для добавления в него информации с помощью процедуры ***Append(f)*** (вместо процедуры ***Rewrite***). При использовании процедуры ***Append(f)*** не создаётся новый файл, а открывается существующий, и курсор устанавливается в конец файла. Соответственно все операторы ***write*** и ***writeln*** будут выводить информацию в файл, начиная с того места, где стоит курсор и далее.

Открывать файл с помощью процедуры ***Append(f)*** можно, только если такой файл существует, в противном случае произойдёт ошибка и выполнение программы прервётся. После окончания работы с файлом необходимо тоже закрыть его с помощью процедуры ***CloseFile(f).***

Если файл передаётся в процедуру как параметр, то данный параметр ***всегда*** должен быть описан как ***параметр-переменная***, т.е. с помощью слова **var**.

Рассмотрим пример программы, которая создаёт два файла '***проба1.txt***' и '***проба2.txt***' и записывает в них ноли следующим образом – в первом файле 3 строки по 5 нолей, во втором 4 строки по 3 ноля. Ноли отделены друг от друга двумя пробелами.

```
procedure ZapZero(var fl:TextFile;n,m:integer);
 var
 i,j:integer;
 begin
   Rewrite(fl);
   for i:=1 to n do
    begin
      for j:=1 to m do
       write(fl,0:3);
      writeln(fl)
    end;
   CloseFile(fl)
```

```
 end;

procedure TForm1.Button1Click(Sender: TObject);
 var
   f,g:TextFile;
begin
  AssignFile(f, 'проба1.txt');
  AssignFile(g, 'проба2.txt');
  ZapZero(f,3,5);
  ZapZero(g,4,3);
end;
```

Чтение данных из текстового файла.

Рассмотрим чтение из файла. Открывается файл для чтения с помощью процедуры

Reset(f)

Обратите внимание, что при открытии файла для чтения из него, файл должен существовать на диске, если его нет, то появится сообщение об ошибке и выполнение программы прервётся.

Если открытие прошло успешно, то курсор (указатель) помещается в начало открытого файла, т.е. на первую строку, которую мы можем прочитать в строковую переменную ***s*** при помощи процедуры

ReadlLn(f, s)

При выполнении данного оператора в переменную ***s*** записывается первая строка, курсор помещается на начало второй строки. Если снова выполнить оператор ***ReadlLn*** , то в переменную считается вторая строка, а курсор переместится в начало следующей строки и т.д. После того, как с файлом поработали, его необходимо закрыть (так же, как и при записи в файл) с помощью процедуры ***CloseFile(f)***.

Рассмотрим пример процедуры, которая записывает первую строку файла ***'my.txt'*** в поле Memo с именем ***memEx1.***

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                           файловую переменную
   Reset(f);  //Открываем файл для чтения
   ReadLn(f, s); //Читаем из файла строку
   memEx1.Lines.Append(s); //Записываем строку в поле вывода
   CloseFile(f); //Закрываем файл
end;
```

При запуске этой программы на выполнение в поле вывода всегда будет записываться только одна строка – первая. Чтобы считать несколько строк из файла, необходимо воспользоваться циклом. Например, следующая процедура считает и поместит в поле Мемо 5 строк из файла ***'my.txt'.***

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
   i:integer;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                              файловую переменную
   Reset(f);  //Открываем файл для чтения
   For i:=1 to 5 do //выполняем 5 раз
    begin
       ReadLn(f, s); //Читаем из файла строку
       memEx1.Lines.Append(s); //Записываем строку в поле
                                  вывода
   end;
   CloseFile(f); //Закрываем файл
end;
```

Как уже было сказано ранее, текстовые файлы – это файлы последовательного доступа, т.е. нельзя сразу обратиться к конкретной строке, а необходимо предварительно считать все предыдущие. Например, если необходимо вывести в поле вывода только пятую строку, то надо предварительно считать 4 строки. Для этого можно воспользоваться оператором ***ReadLn*** без указания переменной, в которую считываем, т.е. воспользоваться следующей формой оператора ***ReadLn(f)***

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
   i:integer;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                              файловую переменную
   Reset(f);  //Открываем файл для чтения
   For i:=1 to 4 do //выполняем 4 раза
       ReadLn(f); //пропускаем строку
   Readln(f, s); //Читаем из файла строку (пятую)
   memEx1.Lines.Append(s); //Записываем строку в поле вывода
   CloseFile(f); //Закрываем файл
end;
```

В этом примере оператор ***ReadLn(f)*** просто переносит указатель на начало следующей строки, не считывая никаких данных.

Мы не всегда можем знать, сколько именно строк в файле. Как же считать все строки из файла, не зная сколько их? Решить эту задачу можно с помощью булевской функции ***EOF(f)*** (от английского ***E**nd **O**f **F**ile* – конец файла). Данная функция принимает значение «***истина***», если в момент её вызова курсор находится в конце файла и «***ложь***» - в противном случае. Приведём пример процедуры, которая все строки файла перепишет в поле Memo.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                                файловую переменную
   Reset(f);  //Открываем файл для чтения
   While not EOF(f) do //пока не конец файла
    begin
       ReadLn(f, s); //Читаем из файла строку
       memEx1.Lines.Append(s); //Записываем строку в поле
                                  вывода
   end;
   CloseFile(f); //Закрываем файл
end;
```

Обратите внимание, что во всех задачах мы использовали оператор ***ReadLn***, а не ***Read***. Так, например, если в последней задаче использовать оператор ***Read*** вместо ***ReadLn***, то программа никогда не закончит свою работу. Дело в том, что оператор ***Read*** считает данные в строку, но не переведёт курсор в начало следующей строки, а оставит его в конце первой строки. Следующий оператор ***Read*** считает в переменную пустую строку, так как в конце строки больше нет данных, и снова курсор останется на месте и т.д. Таким образом, курсор никогда не попадёт в конец файла и соответственно программа не выйдет из цикла. Следовательно, если данные из файла считываются в строку, то надо использовать оператор ***ReadLn.***

Кстати, а можно ли выяснить, что курсор находится в конце строки? Да, для этого служит другая булевская функция ***EOLN(f)*** (от английского ***E**nd **O**f **L**i**N**e* – конец строки). Не трудно догадаться, что данная функция принимает значение «***истина***», если в момент её вызова курсор находится в конце строки и «***ложь***» - в противном случае.

Мы узнали, как работать с текстовым файлом, если нам необходимо считывать из файла данные, которые интерпретируются как строки. Однако если в файле содержатся данные, которые представляют из себя числа (как целые, так и вещественные), которые отделены друг от друга одним или несколькими пробелами, то можно организовать работу иным способом. В этом случае данные можно считывать сразу в переменные численного типа (целые или вещественные). Посмотрим на примере, как это можно сделать.

Пусть имеется некоторый текстовый файл, содержащий строки, в которых через пробелы (пробелов может быть любое количество) записаны числа, как, например, на рисунке справа.

```
10  7    25
14   1  8  17
24
27  17
11
12  34  45
```

Если сразу после открытия файла для чтения, мы воспользуемся оператором ***Read(f,x)***,

где ***f*** – файловая переменная,

x - переменная целого типа,

то в переменную ***x*** будет записано число ***10***, а курсор остановится сразу после ***0*** перед разделителем (в данном случае пробелом). При выполнении оператора ***Read(f,x)*** ещё раз произойдёт следующее – сначала будут пропущены все разделители (пробелы), и курсор остановится перед числом 7, затем число 7 запишется в переменную ***x***, а курсор остановится после 7 перед очередным разделителем (пробелом). При выполнении оператора ***Read(f,x)*** третий раз произойдёт следующее – сначала будут пропущены все разделители (пробелы), и курсор остановится перед числом 25, затем число 25 запишется в переменную ***x***, а курсор остановится после 5 перед очередным разделителем (в данном случае концом строки). Таким образом, оператор ***Read(f,x)*** (если ***x*** является численной переменной) работает следующим образом: пропускаются все разделители (сколько бы их не было) до достижения данных, считываются данные до следующего разделителя, данные интерпретируются как число и записываются в численную переменную, а курсор останавливается перед разделителем.

Рассмотрим пример

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
   x, i, k:integer;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                              файловую переменную
   Reset(f);  //Открываем файл для чтения
   k:=0;
   For i:=1 to 5 do //выполнить пять раз
    begin
       Read(f, x); //Читаем очередное число
       k:=k+x; //прибавляем его к сумме
   end;
   CloseFile(f); //Закрываем файл
end;
```

После выполнения этой процедуры в переменной ***k*** будет ***57*** - сумма первых пяти чисел ***10, 7, 25, 14 и 1.*** Как можно заметить, оператор ***Read(f, x)*** считывал численные данные в численную переменную, пропуская любое количество разделителей, стоящих между данными. В качестве разделителей могут выступать как пробелы, так и концы строк.

Что произойдёт, если в приведённом примере заменить оператор ***Read(f, x)*** на ***ReadLn(f, x)***? Оператор ***ReadLn(f, x)*** работает почти так же, как и оператор ***Read(f, x)***. Единственное его отличие заключается в том, что после считывания данных в переменную, курсор не остаётся после данных, а перемещается на начало следующей строки.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
```

```
    f:TextFile;
    s:string;
    x, i, k:integer;
begin
    AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                            файловую переменную
    Reset(f);  //Открываем файл для чтения
    k:=0;
    For i:=1 to 5 do //выполнить пять раз
     begin
        ReadLn(f, x); //Читаем очередное число
        k:=k+x; //прибавляем его к сумме
    end;
    CloseFile(f); //Закрываем файл
end;
```

После выполнения этой процедуры в переменной ***k*** будет ***86*** - сумма чисел, стоящих в начале первых пяти строк ***10, 14, 24, 27 и 11.***

Часто возникает задача получить числа из какой-то одной строки, например, посчитать сумму чисел в первой строке, а количество чисел в строке нам не известно. С учётом сказанного выше кажется логичным использовать здесь функцию ***EOLN(f)***. И в данном случае процедура, решающая нашу задачу могла бы выглядеть так

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
    f:TextFile;
    s:string;
    x, k:integer;
begin
    AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                            файловую переменную
    Reset(f);  //Открываем файл для чтения
    k:=0;
    While not EOLN(f) do //пока не конец строки
     begin
        Read(f, x); //Читаем очередное число
        k:=k+x; //прибавляем его к сумме
    end;
    CloseFile(f); //Закрываем файл
end;
```

После выполнения этой процедуры в переменной ***k*** должно быть ***42*** - сумма чисел из первой строки ***10, 7, 25.*** Однако такой результат будет не всегда. Почему же может быть иной результат. Ответ зависит от того, есть ли после последнего числа в строке пробелы или нет. Если пробелов нет, то результат будет соответствовать ожиданию.

Давайте рассмотрим, что произойдёт, если после числа ***25*** в нашем примере будет стоять пробел. В этом случае, после того, как тело цикла выполнится в третий раз и считается число ***25***, курсор остановится после цифры ***5*** перед пробелом. Произойдёт вызов функции ***EOLN(f)***. Функция выдаст результат «***ложь***», так как курсор стоит не в конце строки (вспомним, после курсора стоит пробел), а, следовательно, выход из цикла не состоится и снова выполнится оператор ***Read(f, x)***, который пропустит все разделители (в

нашем случае пробел и конец строки) и считает число из следующей строки. То есть цикл продолжит выполняться и в переменной ***k*** окажется не сумма чисел из первой строки, а сумма чисел из нескольких строк (из скольких конкретно зависит от того, когда после последнего числа в строке не будет пробелов).

Для того, чтобы избежать подобных проблем необходимо ***при чтении чисел*** из файла использовать вместо функции ***EOLN(f)*** функцию ***SeekEOLN(f)***. Функция ***SeekEOLN(f)*** сначала пропускает все пробелы, а потом проверяет, стоит ли курсор в конце строки. Если мы перепишем вышеприведённый пример, заменив функцию ***EOLN(f) на*** функцию ***SeekEOLN(f)***, то программа будет работать верно, всегда, вне зависимости от того стоят ли после последнего числа в строке пробелы или нет.

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
   x, k:integer;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                           файловую переменную
   Reset(f);  //Открываем файл для чтения
   k:=0;
   While not SeekEOLN(f) do //пока не конец строки
    begin
       Read(f, x); //Читаем очередное число
       k:=k+x; //прибавляем его к сумме
   end;
   CloseFile(f); //Закрываем файл
end;
```

Аналогично, при работе с числами вместо функции ***EOF(f)*** надо применять функцию ***SeekEOF(f)***. Данная функция пропускает не только пробелы, но и концы строк.

Например, если необходимо написать процедуру, которая посчитает сумму всех чисел в файле, то она может выглядеть так

```
procedure TfrmEx1.btnEx1Click(Sender: TObject);
  var
   f:TextFile;
   s:string;
   x, k:integer;
begin
   AssignFile(f,'my.txt'); //Ставим в соответствие файл и
                           файловую переменную
   Reset(f);  //Открываем файл для чтения
   k:=0;
   While not SeekEOF(f) do //пока не конец файла
    begin
       Read(f, x); //Читаем очередное число
       k:=k+x; //прибавляем его к сумме
   end;
   CloseFile(f); //Закрываем файл
end;
```

При чтении данных из файла необходимо руководствоваться следующим правилом:

Если данные из файла читаются как ***строки***, то следует использовать функции ***EOLN(f)*** и ***EOF(f)***, если данные из файла читаются как ***числа***, то следует использовать функции ***SeekEOLN(f)*** и ***SeekEOF(f).***

Если при считывании в численную переменную в файле окажутся не численные данные, то возникнет ошибка и работа программы прервётся. Так же произойдёт ошибка, если в целую переменную попытаться считать вещественное число, или при считывании в вещественную переменную, в числе в файле разделитель целой и дробной части будет не точка (а, например, запятая).

ЗАДАНИЯ

Задание 1.

Написать программу, которая найдёт самую короткую строку в текстовом файле и выведет её содержимое в надпись (label). Если таких строк несколько, то вывести последнюю из них. Имя файла вводится в поле ввода.

Для проверки работоспособности программы создайте текстовый файл с помощью текстового редактора (например, Блокнота).

Задание 2.

Написать программу, которая перепишет в поле Memo те строки из текстового файла, которые являются палиндромами (т.е. читаются с двух сторон одинаково). Имя файла вводится в поле ввода (можно использовать диалог).

Для проверки работоспособности программы создайте текстовый файл с помощью текстового редактора (например, Блокнота).

Задание 3.

В текстовом файле записано несколько строк. Каждая строка содержит несколько целых чисел, отделённых друг от друга одним или несколькими пробелами. Написать программу, которая перепишет в другой файл те строки, сумма чисел в которых чётна. Имена файлов вводятся в поля ввода (можно использовать диалоги).

Для проверки работоспособности программы создайте текстовый файл с помощью текстового редактора (например, Блокнота).

Задание 4.

В текстовом файле записаны сведения о номерах школ учащихся, участвовавших в олимпиаде. В первой строке сообщается количество учащихся N, каждая из следующих N строк имеет формат:

```
<Фамилия> <Инициалы> <номер школы>
```

где <Фамилия> – строка, состоящая не более чем из 20 символов, <Инициалы> – строка, состоящая из 4-х символов (буква, точка, буква, точка), <номер школы> – не более чем

двузначный номер. <Фамилия> и <Инициалы>, а также <Инициалы> и <номер школы> разделены одним пробелом. Пример входной строки:

```
Иванов П.С. 57
```

Напишите программу, которая выведет в надпись (label) номер школы из которой было меньше всего участников. Если таких школ несколько, то следует перечислить их всех через запятую. При этом необходимо вывести информацию только по школам, пославшим хотя бы одного участника. Следует учитывать, что N>=1000.

Занятие 25. Стандартные диалоги для работы с файлами

На прошлом занятии мы рассмотрели стандартную (классическую) работу с файлами. Но в системе ***Delphi*** есть возможность организовать работу с файлами в диалоговом режиме. То есть выбирать файлы для чтения и для записи из стандартных окон открытия и сохранения файла.

Компоненты стандартных диалогов работы с файлами расположены на вкладке ***Dialogs*** (Диалоги) палитры компонентов. Компоненты стандартных диалогов не видимы при работе программы.

Мы будем работать с компонентами ***OpenDialog*** (иконка – открытая папка) для чтения информации из файла (при этом файл обязательно должен существовать) и ***SaveDialog*** (иконка – диск) для записи в существующий файл или в новый файл, имя которого вводиться с клавиатуры в стандартном окне для сохранения файлов.

Данные компоненты имеют свойство ***FileName*** (имя файла) и метод ***Execute*** (выполнять, осуществлять, делать), который является функцией логического типа.

Алгоритм решения организации диалога при работе с файлами следующий:

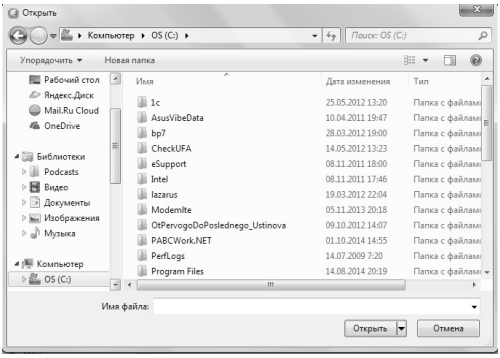
```
Var
    Okf1,Okf2:Boolean;
    StF1, StF2:string;
Begin
   //Определяем выбран ли файл в режиме диалога для чтения.
   //Если в режиме диалога выбран файл для чтения информации,
то
    Begin
        //Определяем имя файла из режима диалога
        //Ставим в соответствие файл и файловую переменную
        //Открываем файл для чтения
          <работаем с информацией из файла: считываем,
обрабатываем и т.д,>
        //Закрываем файл
   End;

//Определяем выбран ли файл в режиме диалога для записи.
```

```
//Если в режиме диалога выбран файл для записи, то
   Begin
         //Определяем имя файла из режима диалога
         //Ставим в соответствие файл и файловую переменную
         //Открываем файл для записи
          <записываем то, что нужно в файл>
         //Закрываем файл
    End;
end;
```

Рассмотрим первую строку программы. Она будет выглядеть следующим образом

```
Okf1:= OpenDialogEx1.Execute;
```

При запуске программы на выполнение перед нами появиться следующее стандартное окно открытия файла, в котором мы выберем интересующий нас файл.



Нажав кнопку «Открыть» мы откроем выбранный файл для чтения информации из него. При этом метод ***Execute*** примет значение ***True***, а имя выбранного файла будет присвоено значению свойства ***FileName*** компонента ***OpenDialogEx1.***

Дальше пишем:

```
If Okf1 then
//Если в режиме диалога выбран файл для чтения информации, то
     Begin
        Stf1:= OpenDialogEx2.FileName;
        //Берём имя файла из режима  диалога
        AssignFile(f, Stf1);
        //Ставим в соответствие файл и файловую переменную
        Reset(f); //Открываем файл для чтения
        ……; //Делаем что-то с файлом
        Closefile(f);  // Закрываем файл
    End;
```

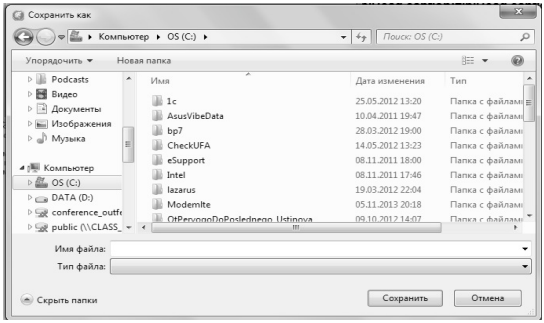
В данной программе после открытия файла строка, состоящая из точек. На месте точек будут стоять операторы, которые нам необходимы для решения поставленной задачи (работаем с файлом так, как было разобрано в предыдущих занятиях, т.е. читаем

информацию из файла так, как требуется для решения задачи, обрабатываем информацию и т.д.)

Для записи информации в файл надо написать так:

```
Okf2:=SaveDialogEx2.Execute;
//Определяем выбран ли файл в режиме диалога для записи.
```

При обработке данной команды перед нами появиться стандартное окно сохранения файла, в котором мы можем выбрать уже существующий файл.



Или в окне «Имя файла» Мы можем написать новое имя файла. После того, как мы нажмём кнопку «Сохранить» Имя выбранного файла (путь к нему) будет присвоено свойству ***SaveDialogEx1.FileName.***

Если нажать отмена, то ничего не произойдёт.
Весь блок записи информации в файл будет выглядеть следующим образом:

```
Okf2:=SaveDialogEx2.Execute;
//Определяем выбран ли файл в режиме диалога для записи.
   If Okf2 then
   //Если в режиме диалога выбран файл для записи
     информации, то
    Begin
       Stf2:= SaveDialogEx2.FileName;
       //Берём имя файла из режима диалога
       AssignFile(g, Stf2);
       //Ставим в соответствие файл и файловую переменную
       Rewrite(g); //Открываем файл для записи
       ……..; //Работаем с файлом
       Closefile(g);   //Закрываем файл
   End;
End;
```

Мы разобрали алгоритм работы с любыми текстовыми файлами в диалоговом режиме.

ЗАДАНИЯ

Задание 1.

Создать в программе блокнот текстовый файл. Написать программу, которая, работая с файлами в режиме диалога, переписывает все строки чётной длины во второй файл, а нечётной длины в третий файл.

Задание 2.

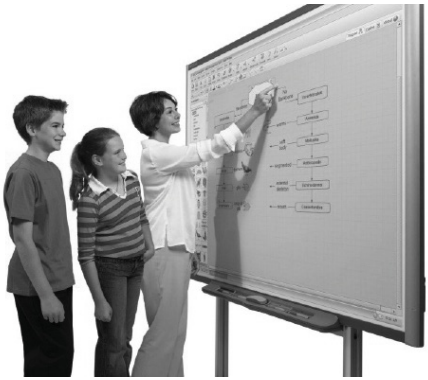
Создать в программе блокнот текстовый файл, каждая строка которого содержит целые числа, отделённые друг от друга одним или несколькими пробелами. Написать программу, которая, работая с файлами в режиме диалога, считает сумму всех чисел и записывает её в другой файл.

Задание 3.

Создать в программе блокнот текстовый файл, каждая строка которого содержит как целые числа, так и любые другие слова, отделённые друг от друга одним или несколькими пробелами. Написать программу, которая, работая с файлами в режиме диалога, считает сумму всех встретившихся в файле чисел и записывает её в другой файл.

softline®

школам



Поставки программного обеспечения. Поставка «софта» во все школы страны.

Поставки компьютерного и учебного оборудования. Ноутбуки, компьютеры и оборудование для мобильных классов, цифровые лаборатории, интерактивное оборудование.

Реализация инфраструктурных проектов. Комплексная автоматизация, порталы, системы документооборота, системы управления учебными материалами и дистанционного обучения, электронные библиотеки.

www.shkola.softline.ru

129343 Москва, Россия, проезд Серебрякова 6
Тел.: +7 (495) 708-43-93
russia.info@embarcadero.com
www.embarcadero.com

При поддержке