МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт сервиса, туризма и дизайна (филиал) СКФУ в г. Пятигорске
Колледж ИСТиД (филиал) СКФУ в г. Пятигорске

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

ОП.09 ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Специальности СПО

09.02.01 Компьютерные системы и комплексы

Квалификация техник по компьютерным системам

Пятигорск, 2020

Методические указания для практических занятий по дисциплине ОП.09 Основы алгоритмизации и программирования составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.01 Компьютерные системы и комплексы.

Рассмотрено на заседании ПЦК ИСТиД (филиал) СКФУ в г. Пятигорске.

Протокол №_8_от _12.03___2020 г.

Составитель В.В. Кондратенко
Директор З.А. Михалина

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа Основы алгоритмизации и программирования предусматривает изучение языков программирования.

При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате освоения учебной дисциплины обучающийся должен **уметь:**

– формализовать поставленную задачу; применять полученные знания к различным предметным областям;

– составлять и оформлять программы на языках программирования;

– тестировать и отлаживать программы;

В результате освоения учебной дисциплины обучающийся должен **знать:**

– общие принципы построения и использования языков программирования, их классификацию;

– современные интегрированные среды разработки программ;

– процесс создания программ; стандарты языков программирования;

– общую характеристику языков ассемблера: назначение, принципы построения и использования;

В результате изучения Основы алгоритмизации и программирования студенты *должны* овладеть общими компетенциями, включающими в себя способность:

ОК1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, определять методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОКЗ. Решать проблемы, оценивать риски и принимать решения в нестандартных ситуациях.

ОК 4. Осуществлять поиск, анализ и оценку информации, необходимой для постановки и решения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии для совершенствования профессиональной деятельности.

ОК 6. Работать в коллективе и команде, обеспечивать ее сплочение, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Ставить цели, мотивировать деятельность подчиненных, организовывать и контролировать их работу с принятием на себя ответственности за результат выполнения заданий.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Быть готовым к смене технологий в профессиональной деятельности.

Профессиональными компетенциями:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование, определение параметров и отладку микропроцессорных систем.
ПК 2.3. Осуществлять установку и конфигурирование персональных компьютеров, и подключение периферийных устройств.

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

По Основы алгоритмизации и программирования практические работы содержат задачи и теоретические вопросы. Варианты для каждого обучающегося – индивидуальные.

Задачи и ответы на вопросы, выполненные не по своему варианту, не засчитываются.

Практическая работа выполняется в отдельной тетради. Условия задачи и формулировки вопросов переписываются полностью. Формулы, расчеты, ответы на вопросы пишутся ручкой, а чертежи, схемы и рисунки выполняются карандашом, на графиках и диаграммах указывается масштаб. Вначале задача решается в общем виде, затем делаются расчёты по условию задания. Решение задач обязательно ведется в Международной системе единиц (СИ).

При выполнении практической работы необходимо следовать методическим указаниям: повторить краткое содержание теории, запомнить основные формулы и законы, проанализировать пример выполнения аналогичного задания, затем преступить непосредственно к решению задачи. К зачету допускаются студенты, получившие положительные оценки по всем практическим работам.

Правила выполнения практических работ.

1. Студент должен прийти на практическое занятие подготовленным к выполнению практической работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и.т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).
6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.
7. Если студент не выполнит практическую работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.
8. Оценку по практической работе студент получает с учетом срока выполнения работы, если;

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Лабораторная работа 1 Элементы интерфейса
Запуск Delphi

Способов запустить среду существует множество (как и любой другой программы впрочем). Ярлык на рабочем столе, иконка на панели быстрого запуска, пункт в главном меню (Пуск - Программы - Borland Delphi n - Delphi n, где n - номер версии). Также есть удобный способ запустить Delphi через окно Пуск - Выполнить - ввести в этом окне delphi32. Более новые версии (2007, 2009, 2010) выпускаются уже не Borland, а CodeGear, поэтому в главном меню группа называется CodeGear (Delphi входит в состав RAD Studio, поэтому может быть и название CodeGear RAD Studio). Из командной строки запуск осуществляется командой bds.

Delphi IDE

Вот и первое, возможно новое, для Вас слово. IDE (Integrated Development Environment) - интегрированная среда разработки программного обеспечения. После запуска Delphi перед Вами предстаёт эта самая среда. Состоит она из нескольких окон. Сейчас мы разберём, что это за окна и каково назначение каждого из них. В разных версиях Delphi эти окна могут выглядить немного по-разному, а некоторые и вообще могут отсутствовать. В данной статье будут приведены иллюстрации окон Delphi 7.

Итак, после запуска, наверное, Вы сразу обратите внимание, что среда в целом практически не отличается от других Windows-приложений. Все элементы стандартные. Главным окном можно считать то, которое содержит строку меню и панели инструментов. Вот строка меню:

Многие из этих пунктов стандартны. Если Вы установили русскую версию Delphi, то пункты будут называться примерно так: Файл, Правка, Поиск, Вид, Проект, Запуск, Компонент, База данных, Инструменты, Окно, Справка.

Как и во многих приложениях здесь есть панели инструментов. Они небольшие, кнопок на них немного, но всё самое основное как раз здесь и собрано. Панели инструментов выглядят примерно так:

Теперь рассмотрим те элементы, которых в обычных приложениях нет.

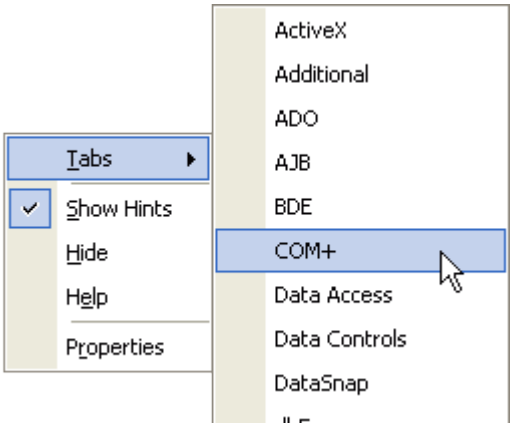
Палитра компонент (Component palette)

Палитра компонент - это набор вкладок, на каждой из которых расположены элементы. Именно с помощью этих элементов создаются интерфейсы

программ. Все эти элементы принято называть *компонентами*. Среди компонент бывают как визуальные, так и невизуальные, но об этом мы поговорим позже. Вот как выглядит палитра компонент:

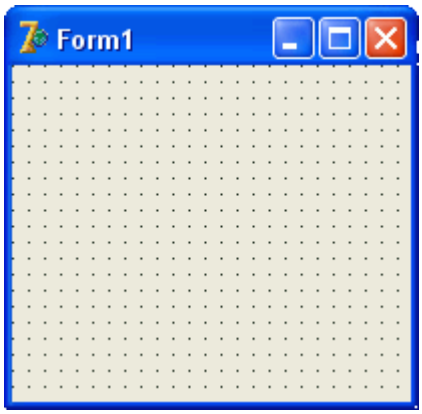
Её внешний вид практически одинаков во всех версиях Delphi. Да что там Delphi, такие же вкладки есть в любой среде объектно-ориентированного программирования (ООП), ибо это самый удобный способ предоставить выбор из сотен (а иногда даже тысяч) различных элементов.

Переключение между вкладками осуществляется стандартным способом - щелчком по названию одной из вкладок. Сразу после установки в Delphi Вы можете видеть огромное количество вкладок. Они даже не помещаются на экране - для прокрутки созданы горизонтальные кнопки со стрелками. Также есть ещё один удобный способ перемещаться по вкладкам - можно щёлкнуть правой кнопкой мыши по палитре компонент и в появившемся меню выбрать пункт Tabs - в результате откроется меню, где будут названия всех существующих вкладок в алфавитном порядке:



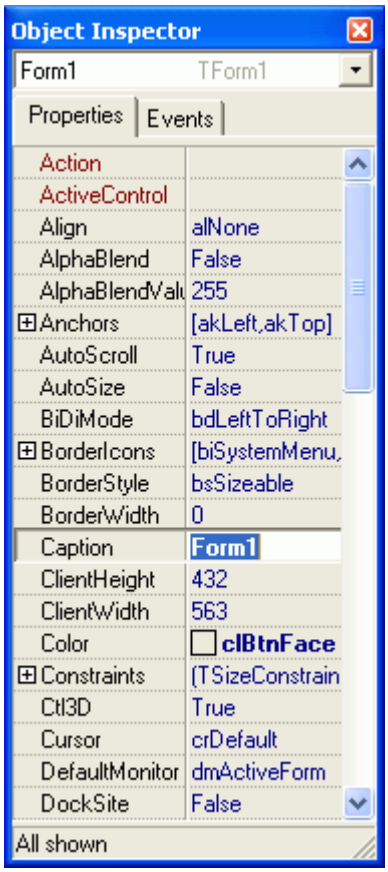
Дизайнер форм (Form Designer)

Это самое большое окно всей среды, которое изначально пустое. Именно это - заготовка окна Вашей программы. Здесь и будут размещаться все компоненты. Удобной составляющей дизайнера форм является сетка (множество точек). С помощью этой сетки компоненты удобно размещать на одном уровне, делать их одинаковых размеров и т.д. Это сделано для того, чтобы приложения соответствовали стандартам, установленным Microsoft. На этом мы ещё остановимся в одной из статей. Сетка является настраиваемой - можно изменить расстояние между точками, а можно её и вовсе отключить.



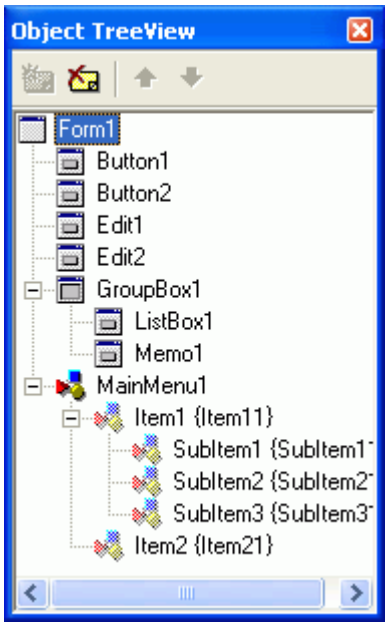
Инспектор объектов (Object Inspector)

Это окошко с двумя вкладками, каждая из которых состоит из двух колонок. В этом окне можно настроить параметры выбранного элемента и задействовать установленные события. Вкладки - *Properties* и *Events* (*Свойства* и *События* соответственно). Что это за свойства и что же такое события? По этому вопросу можно сказать очень много, это тема для отдельной статьи. А вкратце вот о чём речь. Допустим, у нас есть кнопка. Обыкновенная, какая используется в большинстве приложений. Примерами свойств этой кнопки могут быть её размеры (ширина, высота), текст, расположенный на ней и т.д. События - это предопределённые моменты реакции кнопки на какие-либо действия пользователя (либо действия со стороны операционной системы, внешних устройств и т.п.). Самый простой пример - щелчок по кнопке (так называемый "*клик*" - от слова *Click)*. Очевидно, что это событие произойдёт тогда, когда пользователь щёлкнет по кнопке, т.е. нажмёт её. У большинства компонент предусмотрены стандартные события. Как правило, среди них есть все необходимые, которые могут понадобиться при создании программы. Однако можно создать и свои события как реакции на что-либо.



Дерево объектов (Object TreeView)

Это окошко появилось только в Delphi 6, в более ранних версиях его не было. В этом окне отображаются все элементы, какие есть на данной форме. Это сделано с целью упростить выбор компонентов для изменения их свойств в Object Inspector (далее - OI). Помимо того, что отображаются названия компонентов, рядом находятся маленькие графические значки, по которым можно определить, что это за объект. Элементы на форме не всегда автономны, т.е. самостоятельны, поэтому образуются иерархические связи - "деревья". Из-за этого окно и называется деревом объектов. В качестве простейшего примера иерархии объектов можно привести меню. Меню - это самостоятельный компонент, а вот его пункты - это "подчинённые" объекты. Пункт меню не может "висеть в воздухе" - он создан в конкретном меню. Примечание: при динамическом создании пунктов меню они всё же могут просто находиться в памяти и не быть привязанными к какому-либо меню; данный пример приведён лишь для общего представления о связях между объектами.

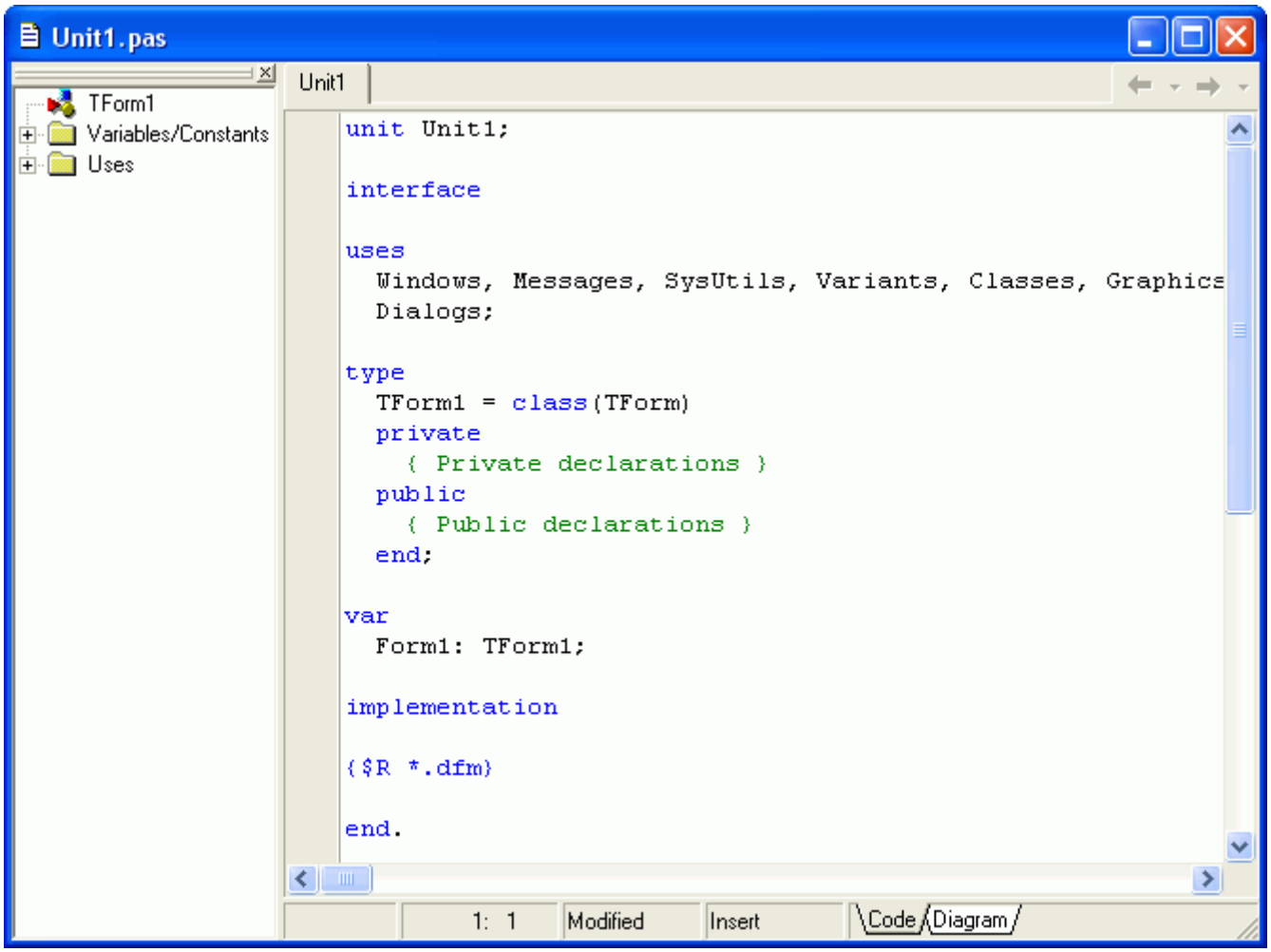


Редактор кода

Редактор кода представляет собой окно, похожее на обычный текстовый редактор за исключением некоторых дополнительных элементов. Основная область этого окна - поле редактирования. Именно здесь пишется текст программы. В отличие от языков программирования, работающих в текстовом режиме (Pascal, QBasic и т.п.) код программы здесь не пишется "с нуля". Как только Вы запустите Delphi и создадите новый проект, то, открыв редактор кода, увидите там уже часть написанной программы. Эти строки удалять ни в коем случае нельзя!

Окно редактора кода может содержать сразу несколько открытых файлов - переключение между ними осуществляется по закладкам вверху окна (на рисунке открыт только один файл - Unit1, поэтому закладка одна-единственная). В левой части окна расположено поле, называемое Code Explorer (*Обозреватель кода*). Здесь в виде дерева отображаются все типы, классы, свойства, методы, глобальные переменные и другие блоки, находящиеся в данном файле (модуле). Дело в том, что содержимое модуля состоит из отдельных участков. Назначение каждого из них мы рассмотрим чуть позже.

В нижней части окна расположена строка состояния, содержащая полезную информацию. В ней представлена текущая позиция курсора в тексте (номер строки : номер символа), текущий режим режима замены (Insert/Overwrite), информация о том, были ли внесены изменения в модуль с момента последнего сохранения и т.п.



Когда Вы попытаетесь запустить программу, то внизу появится информационное поле, в котором будут показаны сообщения об ошибках в тексте программы. Также в этом окне показываются полезные советы по оптимизации кода. В одной из статей мы разберём все эти сообщения более подробно. Если программа написана "идеально", т.е. ошибок нет и оптимизировать нечего, то окошко даже и не появится на экране.

Заключение

Итак, мы рассмотрели все основные элементы оболочки Delphi, которые используются в процессе работы. Конечно же, в Delphi существует множество других окон, но их назначение и способы вызова на экран мы будем рассматривать в процессе работы. Интерфейс в более новых версиях, чем Delphi 7, отличается, но все основные элементы те же самые, просто расположены они несколько иначе. При желании можно настроить оболочку по своему вкусу.

Лабораторная работа №2, ПЕРВЫЙ ПРОЕКТ

Постановка задачи

Создать программу, выполняющую следующие действия.

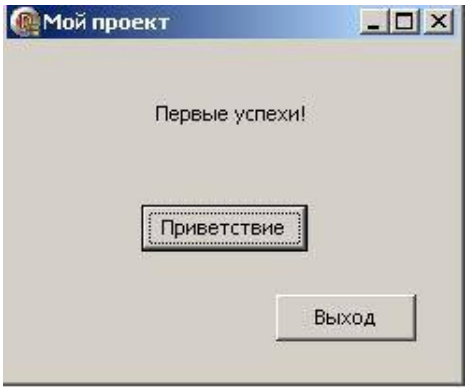


Рис.4

После запуска программы по щелчку мышью на кнопке «**Приветствие»** появляется сообщение «**Первые успехи!**» (рис.4). Для выхода из программы необходимо щелкнуть мышью на кнопке «**Выход»**.

Новым в этой работе является:

- использование компонентов **Label** (метка) и **Button** (кнопка) палитры компонентов **Standard,**
- обработка события **OnClick** – нажатие кнопки.

План разработки программы

1. Откройте новый проект.
2. Разместите в форме экземпляры компонентов: метку **Label** и две кнопки **Button** (см. рис.5).

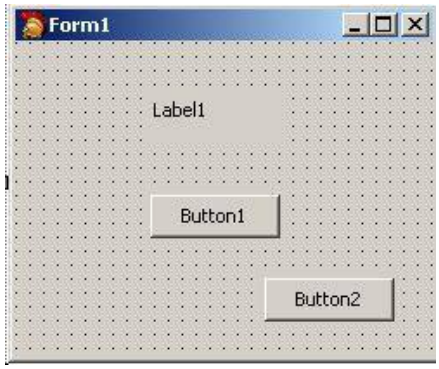


Рис.5

3. Выделите кнопку **Button2**, перейдите в **Object Inspector** на вкладку **Properties**, найдите свойство **Caption** (заголовок) и измените **Button2** на заголовок «**Выход**».

Перейдите на страницу **Events** окна **Object Inspector**, найдите событие **OnClick**, справа от него дважды щелкните мышью. Оказавшись в коде программы, точнее, в

заготовке процедуры **TForm1.Button2Click**, напишите лишь одну команду **Close;** (обязательно поставьте точку с запятой после «Close»).

```
procedure TForm1.Button2Click(Sender:
TObject); begin

  Close;


end;
```

---

4. Сохраните код программы и проект под именами, например, **Unit2.pas** и **Pr2.dpr**.

5. Запустите программу, затем закрыть окно проекта, щелкнув на кнопке «Выход».

6. Выделите форму, в свойстве **Caption** окна **Object Inspector** замените слово **Form1** на «Мой проект». Это и будет заголовком основного окна программы.

7. Выделите кнопку **Button1**, в свойстве **Caption** окна **Object Inspector** замените слово **Button1** на название копки «Приветствие». При необходимости увеличьте длину кнопки.

8. Не снимая выделения с кнопки **Button1**, перейдите на страницу **Events** окна **Object Inspector** и найдите событие **OnClick**, справа от него дважды щелкните мышью. Введите следующий код:

```
Label1.Caption:= 'Первые успехи!';
```

9. Сохраните проект окончательно, запустите и протестируйте его.

Краткое описание плана разработки программы

В этом разделе показано, как можно кратко описать план разработки программы. Для краткости в дальнейшем будем использовать этот способ записи.

1. Откройте новый проект.

2. Разместите в форме экземпляры компонентов компоненты: метку **Label** и две кнопки **Button** (см. рис.5).

3. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие

Button2	Properties	Caption	Выход
	Events	OnClick	`Close;`

4. Сохраните код программы и проект под именами, например, **Unit2.pas** и **Pr2.dpr**.

5. Запустите программу, затем закройте окно проекта кнопкой «Выход».
6. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Form1	Properties	Caption	Мой проект
Button1	Properties	Caption	Приветствие
	Events	OnClick	`Label1.Caption:= 'Первые успехи!';`

7. Сохраните проект, запустите и протестируйте его.

Задание для самостоятельного выполнения

1. Сделайте шрифт выводимой реплики «Первые успехи!» отличным от стандартного по виду, цвету и размеру.

Подсказка. В **Object Inspector** дважды щелкните на кнопку справа от названия свойства **Font** (шрифт), откроется окно выбора шрифта, его цвета и стиля.

2. Замените вид кнопки «Выход» на более привлекательный.

Подсказка. Для замены кнопки надо удалить существующую, а другую кнопку найдите в палитре компонентов на вкладке **Additional**. Она называется **BitBtn**. Затем измените ее вид с помощью свойства **Kind**.

3. Сделайте так, чтобы после нажатия кнопки «Приветствие» на экране появлялось сообщение «Первые и не последние!».

Подсказка. Измените значение свойства **Caption** метки **Label1** при реакции кнопки

Button1 на событие **OnClick**.

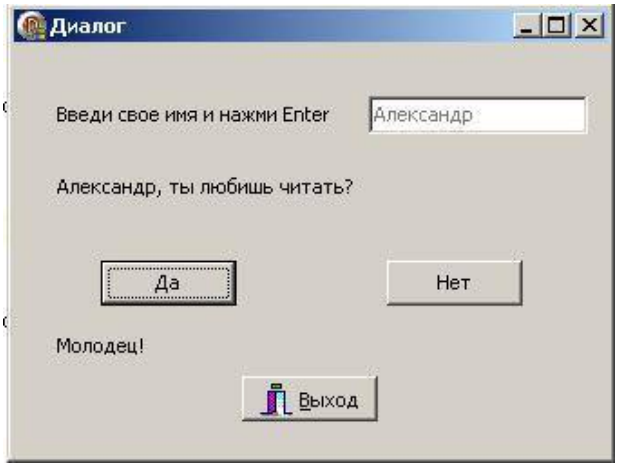
4. Запустите исполняемый файл Pr2.exe не в среде Delphi, а в Windows.

Подсказка. Выйдите из Delphi в Windows. Используйте диспетчер программ или проводник Windows.

Лабораторная работа №3, Проект ДИАЛОГ

Постановка задачи

Создать программу, выполняющую следующие действия.



После запуска программы пользователь вводит свое имя, например, Александр, в прямоугольник с мигающим текстовым курсором и нажимает клавишу **Enter** (см.

рис.6).

Появляется вопрос: «Александр, ты любишь читать?». Если пользователь щелкает на кнопке «Да», то появляется реплика «Молодец!», если на кнопке «Нет», то реплика «Почему же? Надо читать». Для выхода из программы необходимо щелкнуть мышью на кнопке

«Выход».

Рис.6

Новым в этой программе является:

- использование строки ввода **Edit** вкладки палитры компонентов **Standard**,
- обработка события **OnKeyPress** – нажатия клавиши.

План разработки программы

3. Откройте новый проект.

4. Разместите на форме экземпляры компонентов в соответствии с рис.7.

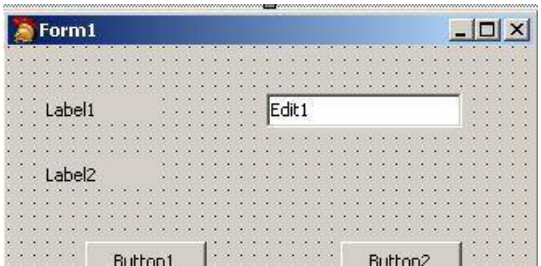


Рис.7

3. Сохраните код программы и проект под именами, например, **Unit3.pas** и **Pr3.dpr**.

Комментарий

В последующем сохраняйте проект и проверяйте его работоспособность после каждого изменения.

5. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Form1	Properties	Caption	Диалог
BitBtn1	Properties	Caption	&Выход
		Kind	bkClose
Label1	Properties	Caption	Введи свое имя и нажми Enter
Edit1	Properties	Caption	Удалить название объекта

	Events	OnKeyPress	`If key=#13 then`
			`  Label2.Caption:=Edit1.Text +`
			`', ты любишь читать? ';`
			Комментарий
			Каждая клавиша имеет свой код. Так клавиша **Enter** имеет код #13, из

			множества управляющих символов ASCII (American Standard Code For Information Interchange – американский стандартный код для обмена информацией).
Button1	Properties	Caption	Да
	Events	OnClick	`Label3.Caption:= 'Молодец! ';`
Button2	Properties	Caption	Нет
	Events	OnClick	`Label3.Caption:= 'Почему же? Надо читать.';`
Label2	Properties	Caption	Удалить название объекта
Label3	Properties	Caption	Удалить название объекта

5. Сохраните проект окончательно, запустите и протестируйте его.

Задание для самостоятельного выполнения

1. Сделайте кнопки «Да» и «Нет» доступными только после ввода имени и нажатия клавиши **Enter**.

6. *Подсказка*. Изменение свойства доступности компонента для пользователя – **Enabled**. Если свойство имеет значение **True**, то компонент доступен, а если значение **False**, то – не доступен. Значение свойства **Enabled** кнопок «Да» и «Нет» установите равными **False**, а в процедуру **Edit1KeyPressed** включите, код *Button1.Enabled := true;*

Button2.Enabled := true;

3. Сделайте, чтобы строка ввода стала не доступной после нажатия кнопки «Да» или «Нет».

4. Введите дополнительную кнопку «Повторить», которая позволяет повторно выполнить задание.

Подсказка. Разместите на форме еще одну кнопку «Повторить ». По нажатию кнопки «Повторить » программно очистите содержимое компонентов **Edit1**, **Label2**, **Label3** для повторного диалога:

```
Label2.Caption := '';

Label3.Caption := '';

Edit1.Text := '';
```

4. Сделайте так, чтобы при повторении диалога строка ввода была бы снова активной.

Подсказка. `Form1.ActiveControl := Edit1.`

Лабораторная работа №4, СПРАВОЧНИК

Постановка задачи

Создать программу, выполняющую следующие действия.

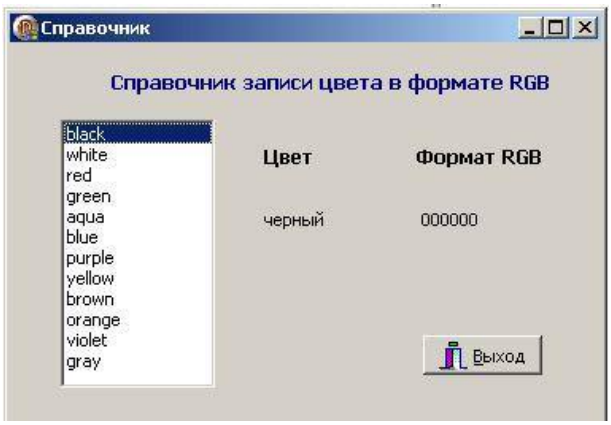


Рис.8

После запуска программы пользователь выбирает с помощью стрелок на клавиатуре название цвета и нажимает клавишу **Enter**. На экране появляется название цвета на русском языке и код в формате **RGB** (рис.8). Программа заканчивает свою работу по нажатию клавиши «Выход».

Новыми в этой работе являются:

- использование компоненты **ListBox** (список) вкладки палитры компонентов **Standard**,
- использование встроенного редактора **String List Editor** для ввода информации,
- алгоритм выбора (оператор **Case**).

План разработки программы

7. Откройте новый проект.
8. Разместите на форме экземпляры компонентов в соответствии с рисунком.

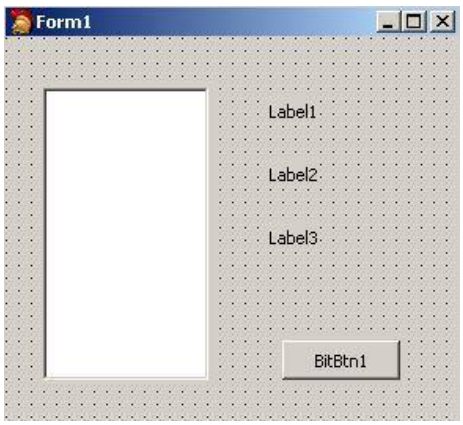


Рис.9

4. Сохраните код программы и проект под именами, например, **Unit4.pas** и **Pr4.dpr**.

5. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Form1	Properties	Caption	Справочник
BitBtn1	Properties	Caption	&Выход
		Kind	bkClose
Label1	Properties	Caption	Справочник записи цвета в формате RGB
Label2	Properties	Caption	Цвет в формате RGB
Label3	Properties	Caption	Удалить название объекта

6.
Выделите объект **ListBox1**, найдите свойство **Items**, щелкните на кнопке с тремя точками, расположенной справа от него. В появившемся окне встроенного редактора **String List Editor** введите названия цветов, каждый на новой строке.

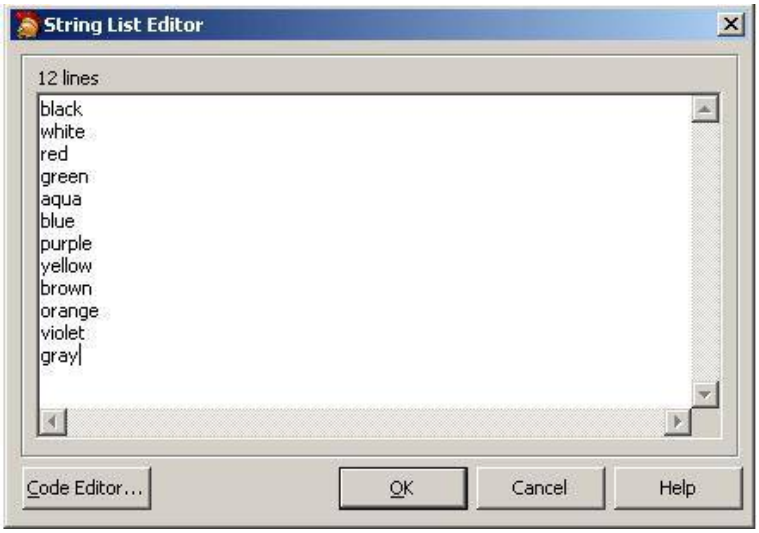


Рис.10

7.
Комментарий
8.
а) Свойство **Items** содержит элементы списка.
9.

б) Список может быть создан при создании формы или во время работы программы.
в) Свойство **ItemIndex** определяет номер элемента , выбранного из списка. Первый элемент имеет номер 0. Если не выбран ни один из элементов, то значение свойства **ItemIndex** равно – «-1».

6. Сохраните набранный текст в файле под именем Color.txt. Для этого нажмите правую кнопку мыши и выберите режим **Save**. Для выхода из встроенного редактора щелкните на кнопке **OK**.

10.

Комментарий

Просмотреть содержимое созданного текстового файла Color.txt, можно с помощью любого текстового редактора, а также внести изменения в тестовый файл, не используя встроенный редактор Delphi.

7. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие

ListBox1	Events	OnKeyPress	`if key=#13 then`
			`Case Listbox1.ItemIndex of`
			`0: Label3.Caption:='черный 000000';`
			`1: Label3.Caption:='белый FFFFFF';`
			`2: Label3.Caption:='красный FF0000';`
			`3: Label3.Caption:='зеленый 00FF00';`
			`4: Label3.Caption:='бирюзовый 00FFFF';`
			`5: Label3.Caption:='синий 0000FF';`
			`6: Label3.Caption:='фиолетовый FF00FF';`
			`7: Label3.Caption:='желтый FFFF00';`
			`8: Label3.Caption:='коричневый 996633';`
			`9: Label3.Caption:='оранжевый FF8000';`
			`10: Label3.Caption:='лиловый FF0008';`
			`11: Label3.Caption:='серый 999999';`

```
end;
```

8. Сохраните проект окончательно, запустите и протестируйте его.

Задание для самостоятельного выполнения

1. Измените шрифт, цвет экрана и букв.

Подсказка. Возможно, придется в коде программы подкорректировать количество пробелов между названием цвета и его кодом.

2. Сделайте так, чтобы при установке курсора мыши в поле **ListBox1**, появлялась подсказка о том, что надо сделать.

Подсказка. Воспользуйтесь свойствами **Hint** (текст сообщения), **Showhint** (показывать ли сообщение) объекта **ListBox1**.

3. Внести изменения в программу, чтобы для надписей «Цвет» и «Формат RGB» использовались два отдельных объекта **Label**.

4. Сделайте так, чтобы выбор цвета в окне **ListBox1** осуществлялся ни только по нажатию клавиши **Enter**, но и при щелчке мыши.

Подсказка. Для компоненты **ListBox1** в обработчике события **OnClick** вставить те же действия, которые описаны в п.7 Плана разработки программы.

5. Сделайте так, чтобы цвет текста, выводимого на **Label3**, соответствовал названию цвета.

Немного теории

В компьютерной графике цвет представляется тремя составляющими: красным, зеленым, голубым (RGB – Red, Green, Blue). В разных пропорциях из этих трех базовых цветов можно получить любой другой. Каждый из цветов представлен в виде одного байта, поэтому для хранения трех цветов достаточно 3 байтов. Только сразу стоит сказать, что на самом деле в Delphi для кодирования цвета отводится не три байта, а четыре. Первый байт используется для обозначения прозрачности, а следующие байты для обозначения цвета.

Один байт может принимать значения от 0 до 255 (в десятичной системе счисления) или от 0 до FF (в шестнадцатеричной системе счисления). В шестнадцатеричной системе счисления коричневый цвет будет выглядеть $00336699, где 00 – байт прозрачности, 33 – байт для голубого цвета, 66 – байт для зеленого цвета, 99 – байт для красного цвета. Отсюда видно,

что на самом деле в памяти цвет хранится как BGR (в обратном порядке). Абсолютно красный цвет – $000000FF, абсолютно зеленый цвет – $0000FF00, абсолютно синий цвет – $00FF0000.

Подсказка. Шрифт, который используется для вывода текста, определяется значением свойства **Font** соответствующего объекта **Label**.Свойство **Font** представляет собой объект типа **TFont**, который имеет свои свойства. Изменить цвет, выводимый на объект **Label** можно с помощью программы, изменив свойство **Color**:

```
Label3.Font.Color:=$FFFFFF; // устанавливается белый цвет
```

Лабораторная работа №5, ВАШ ВЕС

Постановка задачи

Пусть оптимальный вес человека определяется как рост минус 100см. Если фактический вес человека меньше оптимального, то будем считать его худым, если больше, то полным.

Создать программу, выполняющую следующие действия.

Введя рост и фактический вес и нажав кнопку «Расчет», учащийся может определить, худой он или полный и на сколько килограмм надо поправиться или похудеть (рис.11)

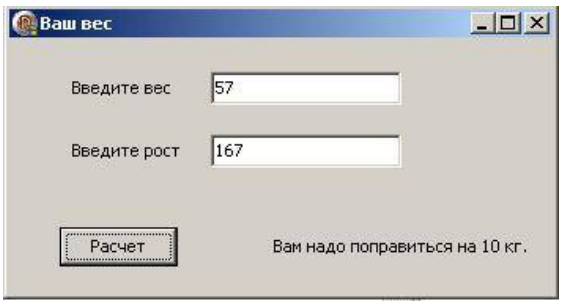


Рис.11

Новым в этой работе являются:

- использование типов переменных - целочисленного и действительного (Integer и Real);
- преобразование строковых данных в числовой тип и числовые в строковые с помощью функций **StrToInt**, **StrToFloat**, **IntToStr FloatToStr**;
- обработка исключительных ситуаций с помощью оператора

  Try – except – end;

- использование процедуры **ShowMessage** для вывода сообщения в отдельном окне.

План разработки программы

9. Откройте новый проект.

10. Разместите в форме экземпляры компонентов в соответствии с рис.12. В поле **Edit1** будем вводить вес в килограммах, а в **Edit2** – рост в сантиметрах.

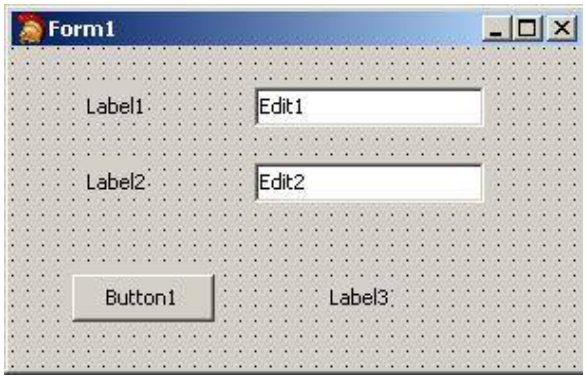


Рис.12

11. Сохраните код программы и проект под именами, например, **Unit5.pas** и **Pr5.dpr**.

Для сохранения результатов расчета введем переменные

faktW – фактический вес, optW – оптимальный вес,

Rost – рост,

Delta – разница между оптимальным весом и фактическим.

4. начале будем считать, что все данные у нас целые числа.

5. блоке реализации после слова **implementation** разместите описание переменных:

```
VAR

factW, optW, Rost, Delta : Integer;
```

5. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие

Label1	Properties	Caption	Введите вес
Label2	Properties	Caption	Введите рост
Label3	Properties	Caption	Удалить название объекта
Edit1	Properties	Text	Удалить название объекта
	Events		
Edit2	Properties	Text	Удалить название объекта
	Events		

Button1	Events	OnClick	factW := StrToInt(Edit1.text);
			Rost := StrToInt(Edit2.Text);
			OptW :=Rost - 100;
			Delta := abs(factW - OptW);
			if OptW = factW then
			Label3.caption := 'Ваш вес идеален! '
			else
			if OptW > factW then
			Label3.caption:= 'Вам надо поправиться на
			'
			+IntToStr(Delta)+ ' кг. '
			else
			Label3.caption:= 'Вам надо похудеть на '
			+IntToStr(Delta)+ ' кг. '

Комментарий

а) Компонента **Edit** содержит информацию строкового типа, поэтому нам необходимо для выполнения вычислений перевести ее в числовой вид.

б) После выполнения арифметических действий результат вычислений нужно будет разместить на форме в компоненте **Label**, которая так же может содержать только информацию строкового типа.

в) Функция **StrToInt** преобразует строку символов в целое число, функция **IntToStr** выполняет обратное действие – целое число преобразует в строку символов.

4. Сохраните проект, запустите и протестируйте его.
5. Усовершенствуйте программу так, чтобы можно было вводить любые десятичные величины. Для этого необходимо использовать вещественный тип переменных **Real**:

```
VAR

factW, optW, Rost, Delta : Real;
```

Комментарий

Преобразование действительных чисел в строковый тип и строковый тип в действительное число выполняется с помощью функций: **FloatToStr** и **StrToFloat**.

Внесите соответствующие изменения в обработку события **OnClick** компонента

Button1.

8. Сохраните проект окончательно, запустите и протестируйте его.

Немного теории

В случае преобразования строкового типа в числовой тип может возникнуть ситуация появления ошибки, если введены недопустимые символы. Если функции **StrToInt** или **StrToFloat** обнаружат ошибку в записи числа, они инициируют так называемую исключительную ситуацию, которая обычно приводит к аварийному завершению работы программы. Но существует возможность не допустить аварийное завершение программы. Для этого используется *обработчик исключительных ситуаций*:

```
Try

<защищенный блок
операторов> except

<обработка исключительной ситуации>
end;
```

Если при выполнении операторов из защищенного блока возникнет исключительная ситуация, управление будет передано в блок операторов, располагающийся между ключевыми словами **except** и **end**. Но если обработка пройдет без ошибок, блок

обработки исключительной ситуации игнорируется и управление передается оператору, следующему за ключевым словом **end**.

Пример использования обработки исключительной ситуации для процедуры **Edit1KeyPressed** может выглядеть так:

```
try
FactW:=StrToInt(Edit1.Tex
t); except

ShowMessage('Ошибочная запись числа: ' +
Edit1.Text); Edit1.SetFocus;

Exit;

end;
```

В результате выполнения оператора

```
FactW:=StrToInt(Edit1.Text);
```

если возникнет исключительная ситуация, то процедура **ShowMessage** выведет на экран простое диалоговое окно, содержащее строку с текстом и кнопку «OK». Эта процедура используется для сообщения пользователю какой-либо информации и не требует принимать каких-либо решений. После появления окна работа программы приостановится в ожидании реакции пользователя.

При вызове стандартной процедуры **Exit** снова активизируется окно редактора (компонента **Edit1**), в котором обнаружен ошибочный текст.

9. Внесите необходимые изменения для обработки исключительных ситуаций, возникающих при вводе чисел.

Задание для самостоятельного выполнения

7. Усовершенствуйте проект:

сделайте к программе заголовок,

сделайте шрифт выводимой реплики отличным от стандартного по виду, цвету и размеру,

вставьте кнопку выхода из программы,
предусмотрите возможность повторного запуска программы (см. Проект «Диалог»).

8. Сделайте так, чтобы в начале программы и после повторного запуска объект **Button1** был не доступен и только после того, как будет введен вес, появлялась возможность нажать на кнопку «Расчет».

Подсказка. Изменение свойства доступности компонента для пользователя – **Enabled**. Если свойство имеет значение **True**, то компонент доступен, а если значение **False**, то – не доступен (см. проект «Диалог»).

3. Сделайте так, чтобы в начале программы и после повторного запуска объекты **Label2** и **Edit2** были не видны и появлялись на экране только после того, как будет введен вес.

Подсказка. Изменение свойства видимости компонента – **Visible**. Если свойство имеет значение **True**, то компонент виден, а если значение **False**, то – не виден.

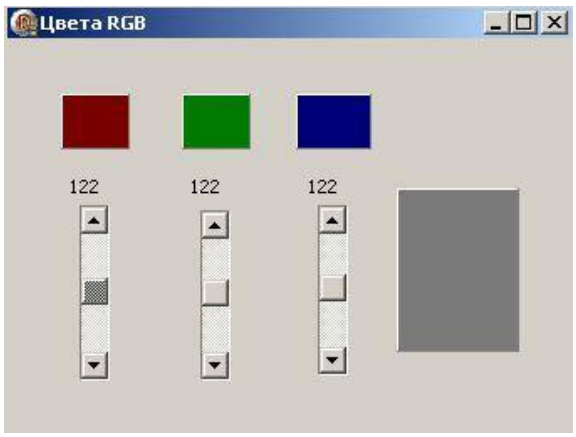
4. Предусмотрите невозможность ввода отрицательных значений веса и роста.
5. Измените алгоритм расчета с учетом *Индекса массы тела*.
Вес – X, Рост – Y.

Индекс массы тела – A, где $A = X / Y^2$
(кг/м^2) Результат определяется по таблице:

№ п/п	Значение индекса	Результат (сообщение, которое надо вывести)
1.	A < 18	Большой недовес
2.	18 <= A < 20	Маловато и небезопасно, можно получить истощение
3.	20 <= A <= 25	Идеально
4.	26 <=A <=30	Легкий недобор
5.	30 < A	Срочно нужно худеть

Лабораторная работа №6, ЦВЕТА В ФОРМАТЕ RGB

Постановка задачи



Создайте программу , с помощью которой пользователь может увидеть в зависимости от значений насыщенности красного, зеленого и синего результирующий цвет

(рис.13).

Рис.13

Новым в этой работе являются:

- использование для ввода данных полосы прокрутки **ScrollBar** вкладки палитры компонентов **Standard**,

- компонента панель **Panel** вкладки палитры компонентов **Standard**,

- функция преобразования значений цветовых составляющих – **TColorRef**.

План разработки программы

Откройте новый проект.

Разместите в форме экземпляры компонентов в соответствии с рис.14.

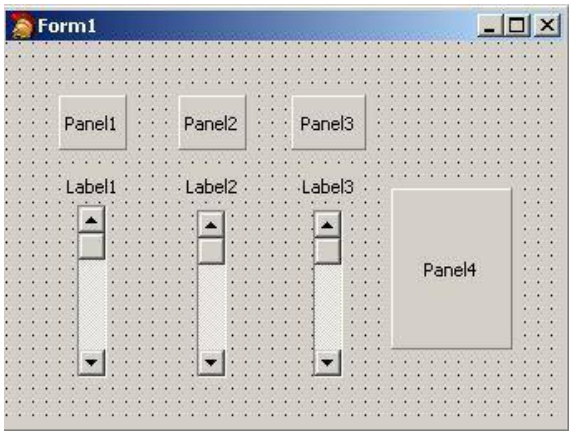


Рис.14

Комментарий

Полоса прокрутки **ScrollBar** может быть горизонтальной (по умолчанию) или вертикальной. Это определяется свойством **Kind**. В нашем случае используется вертикальная полоса прокрутки.

Сохраните код программы и проект под именами, например, **Unit6.pas** и **Pr6.dpr**. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Panel1	Properties	Name	RedPanel **Комментарий** Установка имени панели **RedPanel**, под которым компонент будет известен программе.
		Caption	Удалить название объекта
Label1	Properties	Name	RedLabel

			Комментарий Установка имени метки **RedLabel**, под которым компонент будет известен программе.
		Caption	Удалить название объекта
ScrollBar1	Properties	Name	RedBar **Комментарий** Установка имени полосы прокрутки **RedBar**, под которым компонент будет известен программе.
		Max	255 **Комментарий** Максимальный диапазон целых значений – количество градаций компонента RGB.
		Position	122 **Комментарий** Начальная позиция ползунка – начальное значение.

Аналогично задайте значения для **ScrollBar2**, **Panel2**, **Label2**, присвоив им имена

GreenBar, **GreenPanel**, **GreenLabel** и **ScrollBar3**, **Panel3**, **Label3**, присвоив им имена **BlueBar**, **BluePanel**, **BlueLabel**.

5. Когда на форме много компонентов, ручное выравнивание становится весьма утомительным занятием. Для этого случая в среде Delphi предусмотрены средства автоматизированного выравнивания компонентов.

Выделите компоненты, которые собираетесь выровнять, в нашем случае это **RedLabel** (**Label1**), **RedPanel** (**Panel1**), **RedBar** (**ScrollBar1**). Во всех четырех углах каждого выделенного компонента появятся небольшие квадратики- маркеры. А затем вызовите команду главного меню **Edit**/**Align**, в результате откроется окно Alignment (рис.15).

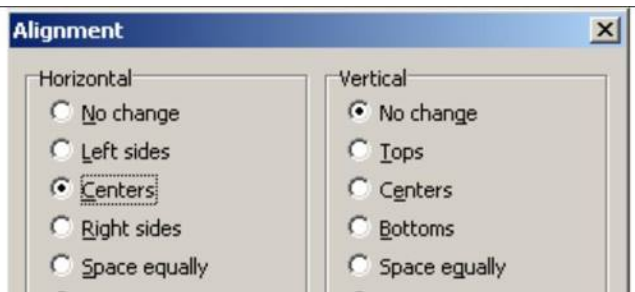


Рис 15

Выберите в списке нужный режим выравнивания и нажмите клавишу «OK».
Повторите эту же операцию для других групп компонент (**GreenBar**, **GreenPanel**,

GreenLabel и **BlueBar**, **BluePanel**, **BlueLabel**).

6. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
RedBar (см. п.4 - ScrollBar1)	Events	OnChange	`RedPanel.Color:= TColorRef(RGB(RedBar.Position, 0, 0)); RedLabel.Caption:=IntToStr(RedBar.Position ); Panel4.Color:= TcolorRef(RGB (RedBar.Position, GreenBar.Position, BlueBar.Position));` **Комментарий** В зависимости от передвижения ползунка **RedBar**, будет меняться цвет панели **RedPanel**, выводиться числовое значение кода на месте **RedLabel** и меняться цвет панели **Panel4**. Функция RGB(R,G,B) превращает три составляющие цвета из трех отдельных значений в одно целое значение цвета. У этой функции три параметра R – значение красного цвета, G – значение зеленого цвета, B – значение синего цвета. В нашем случае в качестве параметров используются значения соответствующих полос прокрутки **ScrollBar**.

TColorRef – это 32-битовое значение, сооветствующее цвету, которое получается с помощью функции RGB.

Аналогично задайте значения для **GreenBar** и **BlueBar**, проследите за правильностью записи параметров в функции **RGB**.

7. Сохраните проект окончательно, запустите и протестируйте его

Задание для самостоятельного выполнения

Усовершенствуйте проект:
сделайте к программе заголовок,
вставьте кнопку выхода из программы.
Предусмотрите, чтобы при запуске программы были установлены начальные цвета панелей в зависимости от исходных значений ползунков.
Подсказка. Поместите в событие при создании формы (**OnCreate** для Form1) обработку значений позиции **RedBar**, **GreenBar** и **BlueBar**.

Внесите изменения так, чтобы выводились на экран значение кода цвета ни только в десятичной системе счисления, но и в шестнадцатеричной системе счисления.

Подсказка. Добавьте объект **Label4**. Поместите в событие по изменению полосы прокрутки **ScrollBar** (RedBar) дополнительную строку, изменяющую свойство нового компонента **Label4**:

```
Label4.Caption:=Format(' %x', [RedBar.Position]);
```

Для перевода значений в шестнадцатеричную систему счисления можно воспользоваться функцией **IntToHex**. Описание этой функции можно найти в **Help**

Лабораторная работа №7, ТЕСТ ПО ФИЗИКЕ

Постановка задачи

Создайте программу, выполняющую следующие действия.

После запуска программы появляется изображение аналогичное рис.16. Пользователь, перемещаясь с помощью клавиш-стрелок по списку «Физическая величина» выбирает любое слово, нажав клавишу «Enter». Затем он переходит в список «Название величины» и выбирает соответствующее название физической величины

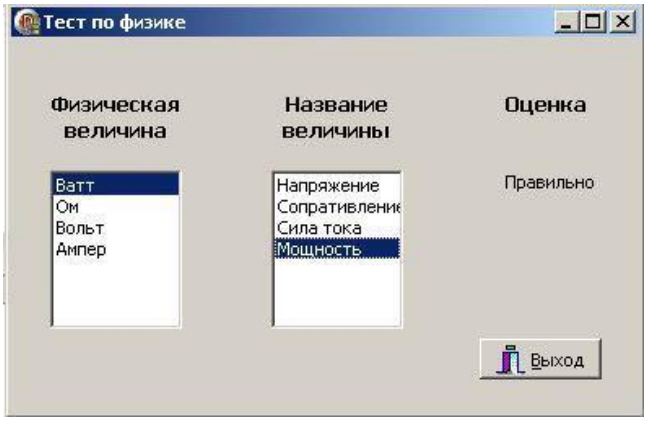


Рис.16

Если выбрано правильное название величины, то под словом «Оценка» появляется одобрительная реплика «Правильно», если выбрано неправильное слово, то – «Ошибка». Количество попыток ответа соответствует количеству записей в списке «Физическая величина».

Новым в этой работе является:

- обеспечение взаимодействия двух списков **ListBox** (вкладка палитры компонентов **Standard**) на основе свойств **Items** и **ItemIndex**,
- создание многострочных надписей в компоненте **Label**.

План разработки программы

Откройте новый проект.
Разместите в форме экземпляры компонент в соответствии с рис.17 и присвойте заголовки меткам. Обратите внимание, что заголовки меток «Физическая величина» и «Название величины» состоят из двух строк и отцентрированы.

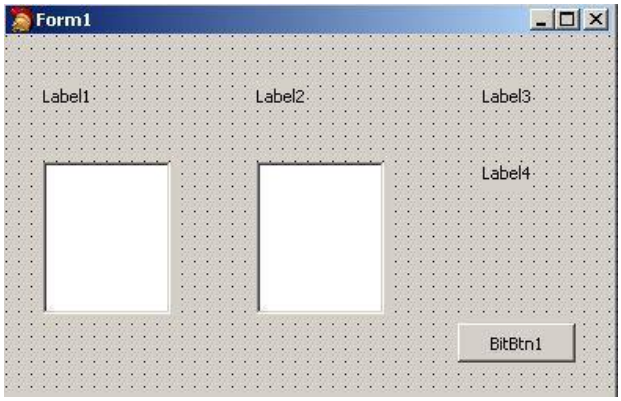


Рис.17

Для вывода многострочных надписей в **Label** задайте:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Label1	Properties	**AutoSize** (изменение размера в зависимости от текста в Caption)	False
		WordWrap (разрыв строки)	True
		Height Width	Установить подходящие размеры

		Alignment (выравнивание текста)	taCenter

12. Сохраните код программы и проект под именами, например, **Unit7.pas** и **Pr7.dpr**.

13. Выделите **ListBox1**, справа от свойства **Items** щелкните на кнопке с тремя точками. В появившемся окне встроенного редактора **String List Editor** введите физические величины, каждую на новой строке: «Ватт», «Ом», «Вольт», «Ампер».

Сохраните набранный текст под именем Fiz_1.txt. Для этого щелкните правой кнопкой мыши и выберите режим **Save**. Для выхода из встроенного редактора щелкните на кнопке «OK» (см. Проект «Справочник»).

5. Выделите **Listbox2** и проделайте с ним аналогичную работу, введя названия физических величин: «Напряжение», «Сопротивление», «Сила тока», «Мощность».

Сохраните набранный текст под именем Fiz_2.txt.

6. Вставьте в разделе реализации после ключевого слова **implementation** объявление переменных:

```
Var   Num1,      // номер выбранной записи в первом окне
      Num2,      // номер выбранной записи во втором окне
      CountR,    // количество правильных ответов
      CountC,    // общее количество ответов
      CountN     // общее количество вопросов теста
         Byte;
```

9. Создайте следующие процедуры обработки событий:

Выделенный объект	Имя события	Действие

Form1	OnCreate	`CountN:=4; // количество записей`
		`CountC:=0;`

ListBox1	OnKeyPress	`If key=#13 then Num1:=ListBox1.ItemIndex;`
		`ActiveControl:=Listbox2;`
		Комментарий
		Запоминает в переменной Num1 номер выбранной записи в первом окне.
		Делает активным объект **Listbox2**, т.е. после окончания

		ввода фокус перейдет в окно ввода **Listbox2**.

ListBox2
 	
1.
	OnKeyPress	begin
		
		case ListBox2.ItemIndex of
		
		Num2:= 2;
		2. Num2:= 1;
		3. Num2:= 3;
		4. Num2:=
		0; end;
		
		if Num1=Num2 then begin
		Label4.Caption:='Верно!';
		CountR:=CountR+1
		
		end
		
		else form1.Label4.Caption:='Ошибка!';
		CountC:=CountC+1;
		
		if CountC=CountN then ShowMessage('Тест окончен.
		
		Баллы : '+
		
		FloatToStr(CountR/CountN * 5)+' (правильных ответов: '+IntToStr(CountR)+')');
		Listbox2.Itemindex:=-1;
		ActiveControl:=Listbox1;
		
		end;

Комментарий

Если выбрана запись во втором окне, то ее номер сравнивается на соответствие с ранее выбранным номером из первого окна (оператор Case). В зависимости от результата сравнения выдается сообщение о правильности ответа , а затем проверяется на все ли вопросы получен ответ.

В конце изменяется значение свойства **Listbox2.Itemindex** для того, чтобы убрать выделение выбранной записи во втором окне и затем делает активным объект **Listbox1**, т.е. после окончания ввода фокус перейдет в окно ввода **Listbox1**.

8. Сохраните проект окончательно, запустите и протестируйте его.

Задание для самостоятельного выполнения

для того, чтобы не рассматривался выбор пустой строки.

Сделайте доступными списки **ListBox1** и **ListBox2** ни только после нажатия клавиши **Enter**, но и по щелчку мыши.

Расширьте количество физических величин до 10. Внести необходимые изменения в программу.

Введите дополнительную кнопку «Повторить», которая позволит повторно выполнить задание, восстановив списки **ListBox1**.

Подсказка.В процедуру обработки нажатия кнопки «Повторить» включить:

```
CountC:=0;

CountR:=0;

Num1:= -1;

Num2:= -1;
Listbox2.Itemindex:=-1;

ListBox1.Items.LoadFromFile('Fiz_1.txt'); // Повторная
загрузка файла Listbox1.SetFocus;
```

Списки **ListBox1** и **ListBox2** сделайте поочередно доступными после нажатия клавиши **Enter**.

Подсказка. Установить значение **False** свойству **Enabled** компонента **ListBox2**, а в процедуру **KeyPressed**, относящуюся к **ListBox1**, включить строки:

```
ListBox2.Enabled := True;

ListBox1.Enabled := False;
```

Внести изменения в программу, чтобы при правильном выборе названия физической величины слово в левом списке исчезало.

Подсказка.В процедуру **KeyPressed**, относящуюся к **ListBox2**, включить:

```
ListBox1.Items.Delete(Num1);

ListBox1.Items.Insert(Num1, '');
```

Здесь мы удаляем строку и вставляем на ее место пустую, чтобы сохранить соответствие между записями в двух окнах.

Лабораторная работа №8, ТЕСТ ПО ИНФОРМАТИКЕ

Постановка задачи

Создайте программу, выполняющую следующие действия.

После запуска программы появляется изображение аналогичное рис.18. Пользователь по своему усмотрению выбирает один из переключателей в группе. В зависимости от правильности ответов появляется одно из сообщений «Плохо», «Удовлетворительно», «Хорошо», «Отлично».

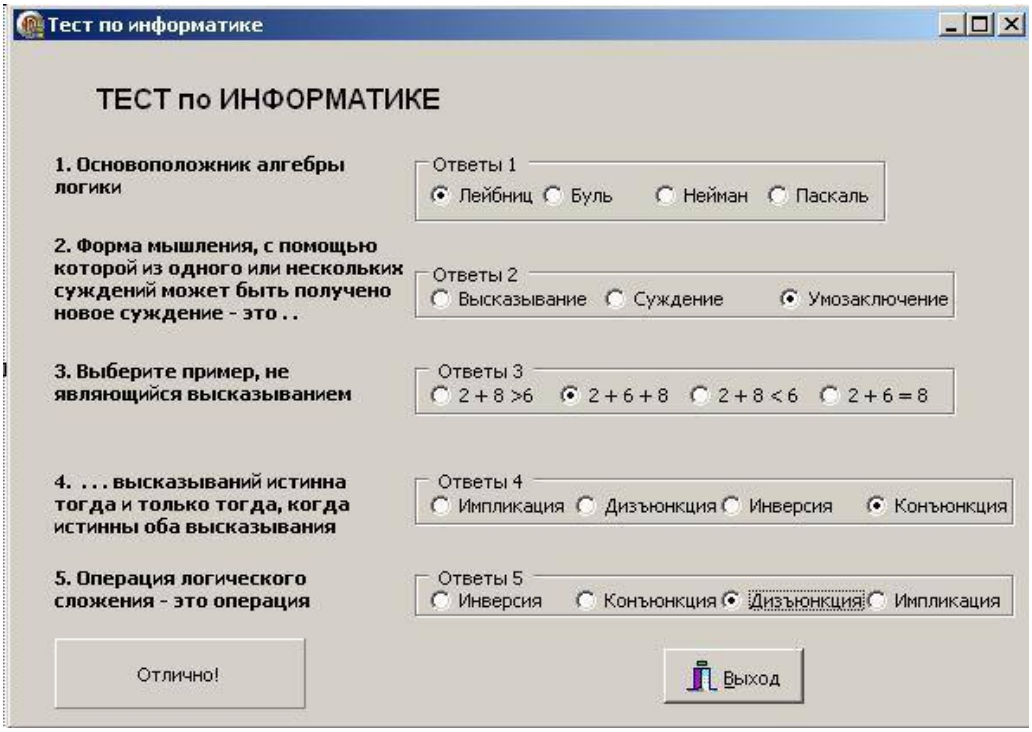


Рис.18

Новым в этой работе являются:

- группа переключателей **RadioGroup** вкладки палитры компонентов **Standard**.

План разработки программы

Откройте новый проект.

Разместите в форме объекты в соответствии с рис.19 и присвойте заголовки меткам и панелям

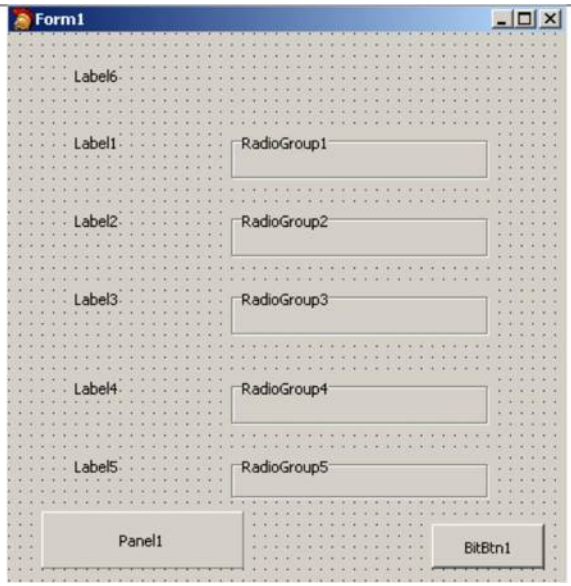


Рис.19

3. Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие
Label1	Properties	Caption	Основоположник алгебры логики
Label2	Properties	Caption	Форма мышления, с помощью которой из одного или нескольких суждений может быть получено новое суждение - это . . .
Label3	Properties	Caption	Выберите пример, не являющийся высказыванием
Label4	Properties	Caption	. . . высказываний истинна тогда и только тогда, когда истинны оба высказывания
Label5	Properties	Caption	Операция логического сложения - это операция
RadioGroup1	Properties	Caption	Ответы 1
		Columns	4
		Items	**Лейбниц** Буль Нейман Паскаль Введенный текст сохраните в файле T_1.txt (см. Проект «Справочник»).
RadioGroup2	Properties	Caption	Ответы 2

		Columns	3
		Items	Высказывание Суждение **Умозаключение** Введенный текст сохраните в файле T_2.txt**.**
RadioGroup3	Properties	Caption	Ответы 3
		Columns	4
		Items	2 + 8 >6 **2 + 6 + 8** 2 + 8 < 6 2 + 6 = 8 Введенный текст сохраните в файле T_3.txt.
RadioGroup4	Properties	Caption	Ответы 4
		Columns	4
		Items	Импликация Дизъюнкция Инверсия **Конъюнкция** Введенный текст сохраните в файле T_4.txt.
RadioGroup5	Properties	Caption	Ответы 5
		Columns	4
		Items	Инверсия Конъюнкция **Дизъюнкция** Импликация

Введенный текст сохраните в файле T_5.txt

Сохраните код программы и проект под именами, например, **Unit8.pas** и **Pr8.dpr**.

Вставьте в разделе реализации после ключевого слова **implementation** объявление переменной для подсчета правильных ответов:

```
Var SUM : Byte;
```

Для суммирования набираемых пользователем баллов, создайте следующую процедуру обработки события:

Выделенный объект	Имя события	Действие

RadioGroup1	OnClick	`SUM:=0;`
		`If RadioGroup1.ItemIndex=0 Then SUM:=SUM+1;`
		Комментарий
		Индекс первого переключателя равен 0. Правильный ответ содержит переключатель с меткой «Лейбниц», имеющий индекс 0.

Вставьте в обработчики событий **RadioGroup2.OnClick**, **RadioGroup3.OnClick**, **RadioGroup4.OnClick**, **RadioGroup5.OnClick** аналогичные коды, с учетом правильных ответов, но без обнуления переменной **SUM**, так как это необходимо лишь один раз перед началом суммирования.

Выведем на контрольную панель итоговое сообщение в зависимости от набранной суммы баллов и выведем сообщение об окончании тестирования.

Выделенный объект	Имя события	Действие

RadioGroup5	OnClick	`Case SUM of`
		`0..2: Panel1.Caption:='Плохо!';`
		`3: Panel1.Caption:='Удовлетворительно!';`
		`4: Panel1.Caption:='Хорошо!';`
		`5: Panel1.Caption:='Отлично!';`
		`end;`
		`ShowMessage('Конец теста');`

10. Сохраните проект окончательно, запустите и протестируйте его.

Задание для самостоятельного выполнения

Для контроля правильности работы программы выведите на панель количество правильных ответов пользователя.
Запустите программу и убедитесь, что верная сумма баллов получается лишь при последовательном выборе переключателей сначала из **RadioGroup1**, затем из **RadioGroup2** и т.д. Если порядок нарушен, то результат может быть неверным. Исправьте эту ошибку.
Введите дополнительную кнопку «Повторить», которая позволит повторно выполнить задание.
Для наглядности предусмотрите возможность вывода результата разным цветом.

Лабораторная работа № 9, МАТРИЦА

Постановка задачи

Создайте программу, которая в зависимости от величин N (количество строк) и M (количества столбцов) создает матрицу размером NxM. Программа предоставляет возможность заполнить матрицу с помощью случайных чисел или ввести значения вручную. Кроме этого можно подсчитать сумму элементов матрицы, определить максимальный и минимальный элементы матрицы.

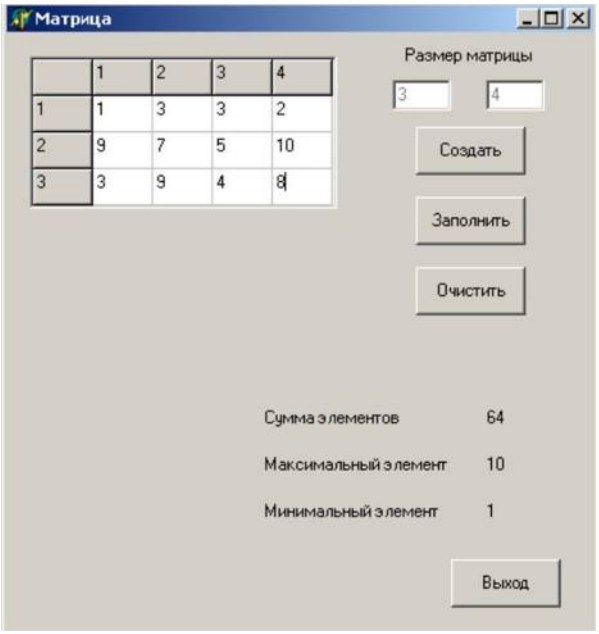


Рис.1

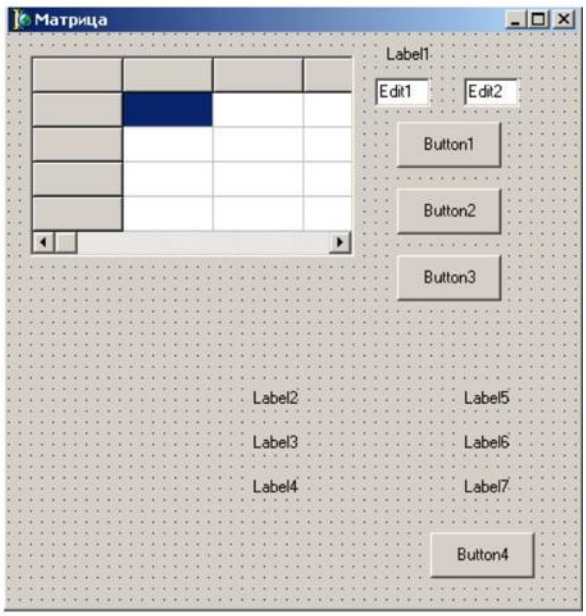


Рис.2

Новым в этой работе являются:

- компонент таблица строк **StringGrid** вкладки палитры компонентов **Additional**.

План разработки программы

Откройте новый проект.

Разместите в форме объекты в соответствии с рис.2.

Установите свойства компонент на вкладке **Object Inspector**:

Выделенный объект	Имя свойства	Значение
Label1	Caption	Размер матрицы
Edit1	Text	Удалить название объекта
Edit2	Text	Удалить название объекта
Button1	Caption	Создайте
Button2	Caption	Заполнить
Button3	Caption	Удалить название объекта
Label2	Caption	Сумма элементов
Label3	Caption	Максимальный элемент
Label4	Caption	Минимальный элемент
Label5	Caption	Удалить название объекта
Label6	Caption	Удалить название объекта
Label7	Caption	Удалить название объекта
Button4	Caption	Выход
StringGrid1	Name	MATR

4. Сохраните код программы и проект под именами, например, **Unit_M.pas** и **Pr_M.dpr**.

Немного теории

Компонент **StringGrid** (вкладка палитры компонентов **Additional**) предназначен для создания таблицы.

Свойству компонента **Cells** соответствует двухмерный массив ячеек, каждая из которых может содержать произвольный текст. Содержание ячейки массива, находящегося на пересечении столбца с номером **Col** и строки с номером **Row** определяется элементом **StringGrid 1.Cells [Col, Row]**. Это содержимое можно редактировать.

Двумерный массив состоит из двух частей: фиксированной и рабочей.

Фиксированная часть служит для показа заголовков столбцов (строк) и для ручного управления их размерами. Обычно фиксированная часть занимает крайний левый столбец и самую верхнюю строку таблицы. С помощью свойств **FixedCols** и **FixedRows** можно задать другое количество фиксированных столбцов и строк. Если свойства **FixedCols** и **FixedRows** имеют значение 0, то таблица не содержит фиксированной зоны.

Рабочая часть – это остальная часть таблицы. Она может содержать произвольное количество столбцов и строк, которое можно изменять в ходе выполнения программы. Если рабочая часть не умещается целиком в пределах окна компонента, то автоматически появляются полосы прокрутки. При прокрутке рабочей области фиксированная область не исчезает.

В таблице приведен перечень часто используемых свойств компонента **StringGrid**.

ColCount	Количество столбцов таблицы
RowCount	Количество строк таблицы
FixedCols	Количество столбцов фиксированной зоны
FixedRows	Количество строк фиксированной зоны
DefaultRowHeight	Высота строки таблицы
Height	Высота всей таблицы
DefaultColWidth	Ширина столбца таблицы
Width	Ширина всей таблицы
Options.goEditing	Признак редактирования содержимого ячеек таблицы. True – редактирование разрешено, False – запрещено
GridLineWidth	Ширина линий, ограничивающих ячейки таблицы
Font	Шрифт, используемый для отображения содержимого ячеек таблицы
Options .goEditing	Признак допустимости редактирования содержимого ячеек таблицы. True – редактирование разрешено, False – запрещено
Options .goTabs	Разрешает (True) или запрещает (False) использование клавиши «Tab» для перемещения курсора в следующую ячейку таблицы

Разместите в блоке реализации после слова **implementation** описание переменных

```
Var
    N, M: Integer;    //N-количество строк, M-количество столбцов
    MATR_MAX,         //Значение максимального элемента таблицы
    MATR_MIN,         //Значение минимального элемента таблицы
    MATR_SUM: Integer;//Значение суммы элементов таблицы
```

. Основная задача проекта – создать таблицу, размер которой будет определен после заполнения полей **Edit1** и **Edit2**.

5.

Создадим следующие процедуры обработки событий:

Выделенный объект	Имя событие	Текст процедуры

Button4
 	
«Выход»
	OnClick	`procedure TForm1.Button4Click(Sender: TObject);`
		`begin`
		`Close;`
		`end;`

Form1	OnCreat	`procedure TForm1.FormCreate(Sender: TObject);`
		`MATR.Visible:=False;`
		`Button2.Enabled:=False;`
		`Button3.Enabled:=False;`
		`end;`
		Комментарий
		При создании формы установим компонент **StringGrid** невидимым (новое имя этого компонента **MATR**), т.к. в начале неизвестно сколько же строк и столбцов в таблице. Кроме этого до определения размера таблицы установим недоступными кнопки «Заполнить» и «Очистить».

Button1
 	
«Создать
 	
таблицу»
	OnClick	`procedure TForm1.Button1Click(Sender: TObject);`
		`Var I, J:Byte;`
		`begin`
		`If (Edit1.Text<>'') and (Edit2.Text<>'') Then begin`
		`Edit1.Enabled:=False;`
		`Edit2.Enabled:=False;`

```
 Button1.Enabled:=False;
 Button2.Enabled:=True;
 Button3.Enabled:=True;
 N:=StrToInt(Edit1.Text);
 M:=StrToInt(Edit2.Text);
 MATR.DefaultColWidth:= 40;
 MATR.RowCount:=N+1;
 MATR.ColCount:=M+1;
 MATR.Height:=(MATR.DefaultRowHeight+2)*(N+1);
 MATR.Width:= (MATR.DefaultColWidth +2)*(M+1);
 MATR.Visible:=True;
 For I:=1 to N do
  MATR.Cells[0,I]:=IntToStr(I);
 For J:=1  to M do
  MATR.Cells[J,0]:=IntToStr(j);
```

```
end;
end;
```

Комментарий

Данная процедура будет выполняться только при условии, что введены размеры матрицы, т.е **Edit1.Text** и **Edit2.Text** не являются «пустым символом». В начале устанавливаются недоступными компоненты ввода, определяющие размер таблицы, и кнопка «Создать», а недоступные ранее кнопки «Заполнить» и «Очистить» становятся доступными. После этого переходим к формированию таблицы. Для этого устанавливаем значения переменных N и M, определяющих количество строк и столбцов таблицы – переводим из текстовой информации значения полей **Edit1** и **Edit2** в числовые значения. После этого программно задаем свойства компонента **StringGrid** (**MATR**) и устанавливаем его видимым. В конце заполняем фиксированную часть таблицы значениями номеров строк и столбцов.

Button2
 	
«Заполнить
 	
таблицу»
	OnClick	procedure TForm1.Button2Click(Sender: TObject);
		Var I, J:Byte;
		begin
		Randomize;
		For I:=1 to N do
		For J:=1 to M do
		MATR.Cells[J, I]:=IntToStr(Random(N*M));
		MATR_MAX:= StrToInt(MATR.Cells[1, 1]);
		MATR_MIN:= StrToInt(MATR.Cells[1, 1]);
		MATR_SUM:=0;
		For I:=1 to N do
		For J:=1 to M do
		begin
		MATR_SUM:=MATR_SUM+StrToInt(MATR.Cells[J, I]);
		IF StrToInt(MATR.Cells[J, I])>MATR_MAX Then
		MATR_MAX:=StrToInt(MATR.Cells[J, I]);
		IF StrToInt(MATR.Cells[J, I])<MATR_MIN Then
		MATR_MIN:=StrToInt(MATR.Cells[J, I]);
		end;
		Label5.Caption:=IntToStr(MATR_SUM);
		Label6.Caption:=IntToStr(MATR_MAX);
		Label7.Caption:=IntToStr(MATR_MIN);
		end;
		Комментарий
		В начале процедуры заполняются ячейки таблицы (MATR.Cells[J,I]) случайными числами в диапазоне от 0 до (N*M-1). Дальше определяются максимальное и минимальное числа и сумма элементов таблицы. В конце процедуры полученные значения выводятся на экран. Для этого используются компоненты **Label5**, **Label6**, **Label7**.
		Для заполнения ячеек таблицы как

		положительными, так и отрицательными
		числами, например, в диапазоне от -10 до 10 можно записать: `MATR.Cells[J, I]:=IntToStr(Random(11))-10;`

Button3
 	
«Очистить
 	
таблицу»
		`procedure TForm1.Button3Click(Sender: TObject);`
		`Var I, J:Byte;`
		`begin`
		`For I:=1 to N do`
		`For J:=1 to M do`
		`MATR.Cells[J, I]:= '';`
		`Label5.Caption:= '';`
		`Label6.Caption:= '';`
		`Label7.Caption:= '';`
		`Edit1.Text:= '';`
		`Edit2.Text:= '';`
		`Matr.Visible:=False;`
		`Button2.Enabled:=False;`
		`Button3.Enabled:=False;`
		`Button1.Enabled:=True;;`
		`Edit1.Enabled:=True;`
		`Edit2.Enabled:=True;`
		`end;`
		Комментарий
		Все ячейки таблицы заполняется пустой строкой, а затем очищаются компоненты **Label5**, **Label6**, **Label7**, **Edit1.Text**, **Edit2.Text**. Компонент **StringGrid** (**MATR**) устанавливается невидимым, кнопки «Заполнить» и «Очистить» становятся недоступными, кнопка «Создать» и компоненты **Edit1** и **Edit2** – доступными.

7. Программа готова. Но если в поля ввода «Размерность матрицы» мы по ошибке вместо числа введем букву или какой-либо другой символ, то

программа будет прервана. Для того чтобы это избежать создадим процедуры обработки события

KeyPress для компонента **Edit1**.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var
Key:Char); begin


case Key of
  '0'..'9':
  ; #8: ;


  #13:
  Edit2.SetFocus;
  else Key := Chr(0);

end;
```

Комментарий

В зависимости от нажатой клавиши программа выполнит следующие действия:

нажата любая цифровая клавиша или клавиша «Back Space» (код клавиши #8) – программа ничего не изменит,
нажата клавиша «Enter» (код клавиши #13) – курсор перейдет в окно редактора ввода **Edit2**,
во всех остальных случаях – введенный символ будет удален (присвоение значения пустого символа Chr(0)).

Для компонента **Edit2** самостоятельно создайте процедуру, обрабатывающую ввод. Она будет отличаться тем, что при нажатии клавиши «Enter», курсор должен перейти на кнопку «Создать таблицу» (**Button1**).

4. Созданная программа не позволяет редактировать элементы таблицы. Внесем изменения в свойства компонента **StringGrid** (**MATR**). Для этого на вкладке **Object Inspector** свойству **Options.goEditing** установим значение **True**. Процедура, обрабатывающая нажатие клавиши в поле компонента **StringGrid** (**MATR**) будет выглядеть так:

```
procedure TForm1.MATRKeyPress(Sender: TObject; var Key:
Char); Var I,J:Byte;
5. begin
case Key of
'0'..'9': ;
6. #8: ;
#13: if MATR.Col<MATR.ColCount-1 then
7. MATR.Col:=MATR.Col+1; else Key:=Chr(0);
end;
8. MATR_MAX:= StrToInt(MATR.Cells[1,1]);
MATR_MIN:= StrToInt(MATR.Cells[1,1]);
9. MATR_SUM:=0;
For I:=1 to N do
For J:=1 to M do
begin
MATR_SUM:=MATR_SUM+StrToInt(MATR.Cells[J,I]
); IF StrToInt(MATR.Cells[J,I])>MATR_MAX
10. Then
```

```
MATR_MAX:=StrToInt(MATR.Cells[J,I]);
IF StrToInt(MATR.Cells[J,I])<MATR_MIN
Then
MATR_MIN:=StrToInt(MATR.Cells[J,I]);
end;
Label5.Caption:=IntToStr(MATR_SU
M);
Label6.Caption:=IntToStr(MATR_MA
X);
Label7.Caption:=IntToStr(MATR_MI
N);
end;
```

Задание для самостоятельного выполнения

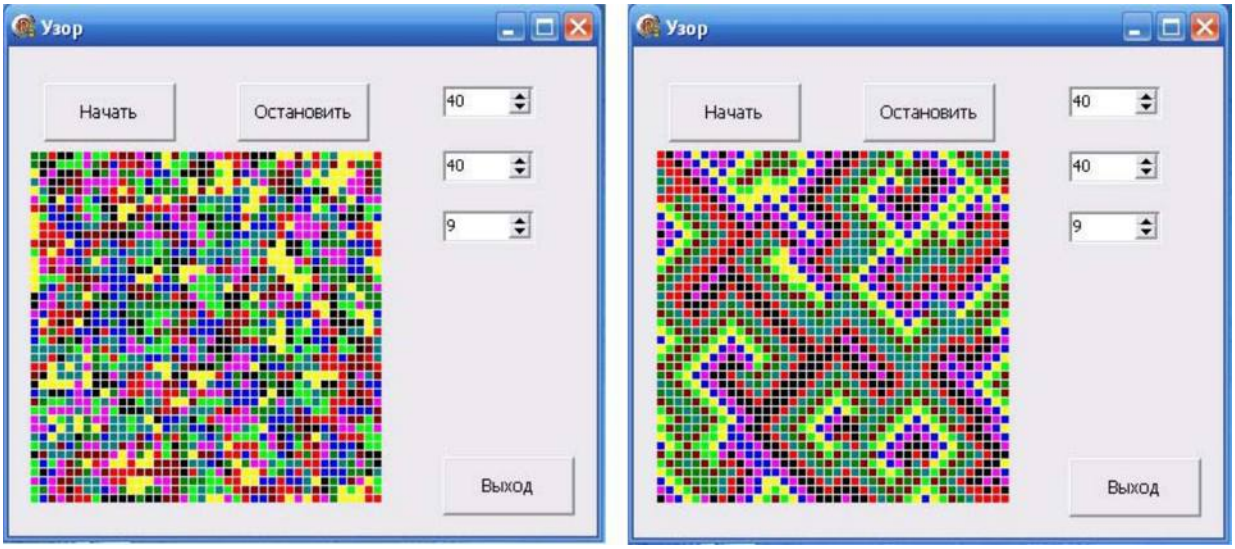
1. Дополните программу, вставив блок определения суммы по каждому столбцу матрицы.

Подсказка. Необходима еще одна таблица (компонент **StringGrid**), в которой будут находиться подсчитанные суммы по столбцам. Формировать эту таблицу нужно в момент формирования основной таблицы

Лабораторная работа №10, УЗОР

Постановка задачи

Создайте программу, которая в зависимости от величин N (количество строк) и M (количества столбцов) создает разноцветный узор размером N×M, первоначально заполненяя его случайным образом различными цветами палитры (рис.1 а). Количество цветов палитры определяется величиной K, которая задается в начале программы. Рисунок узора постоянно преобразуется по определенному правилу (рис. 1 б).



а) б) Рис.1

Правило преобразования разноцветного узора. Представим область, разбитую на клеточки. Каждая клеточка имеет свой цвет.

Цвета упорядочены по номерам. За красным цветом идет зеленый, за зеленым - желтый , за желтым - синий. Будем считать, что за самым последним цветом опять идет первый, то

есть за синим цветом следует красный. Кроме того, будем считать, что верхний и нижний, а также правый и левый края нашей области соединены друг с другом.

Проверим для каждой клеточки цвета соседей. Если среди соседей есть клетки со «следующим» цветом, то цвет нашей клетки становится таким же. Если таких соседей нет, цвет клетки остается без изменений.

Проверка и обновление всех клеток образуют шаг изменения узора. Таких шагов может быть сколько угодно.

Для упрощения обозначим каждый цвет числом. Например, красный цвет обозначим числом 1, зеленый - 2, желтый - 3, синий - 4. В этом случае Рис.2 можно представить в виде таблицы.

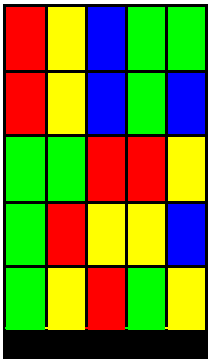
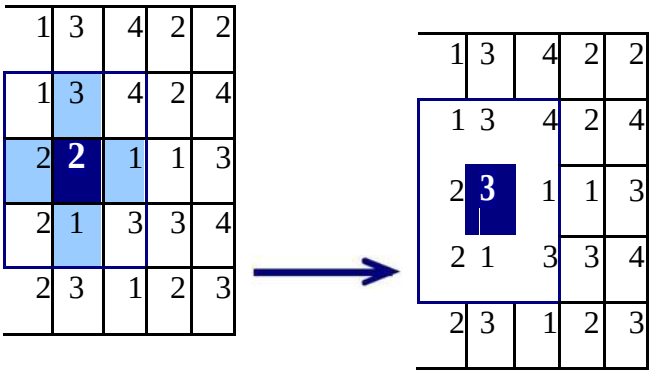


Рис.2

1	3	4	2	2
1	3	4	2	4
2	2	1	1	3
2	1	3	3	4
2	3	1	2	3

Посмотрим как изменится цвет клетки с координатами [3,2], если воспользоваться описанным ранее правилом. Наша клетка имеет цвет «2», вокруг нее расположены клетки с цветами «2», «3», «1», «1». Так как среди соседей клетки с координатами [3,2] есть клетки со «следующим» цветом («3»), то цвет нашей клетки становится «3».



В программе цветов может быть не четыре, а намного больше. На первый взгляд может показаться, что, каким бы ни был начальный узор, в конце концов получится одноцветная область. Но это мнение ошибочно. Что получитсч на самом деле, будет видно, когда программа заработает.

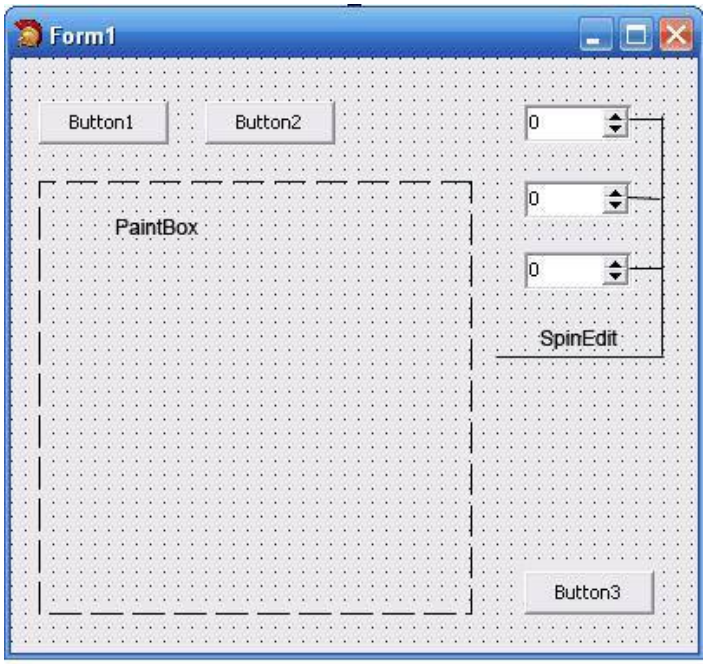


Рис.3

Нам понадобятся следующие компоненты:

№ п/п	Наименование компонента	Страница	Для чего предназначен
1	Button1	Standard	Кнопка начала построения узора
2	Button2	Standard	Кнопка остановки построения узора или продолжения построения узора
3	Button3	Standard	Кнопка выхода из программы
4	SpinEdit1	Samples	Количество строк в узоре
5	SpinEdit2	Samples	Количество столбцов в узоре
6	SpinEdit3	Samples	Количество цветов в узоре
7	PaintBox	System	Поле построения узора

Новым в этой работе являются:

- организация бесконечного цикла

Основные этапы программы

Задаются начальные значения количество строк (N_SIZE) и столбцов (M_SIZE) в рисунке, количество цветов (K_SIZE) в палитре.

Формируется исходный числовой массив (AR), состоящий из цветов заданной палитры. Для заполнения массива используется функция Random.

Расчитывается размер цветных прямоугольников, которыми будет заполняться поле PaintBox при формировании рисунка, в зависимости от велечин количества строк (N_SIZE) и столбцов (M_SIZE).

Лабораторная работа №11, ЗАПОЛНЕНИЕ УЗОРА

Заполняется поле PaintBox разноцветными прямоугольниками. Значение цвета прямоугольника определяется на основании числового массива AR.

После нажатия кнопки «Начать» программа приступает к преобразованию узора:

последовательно просматривает все значения массива AR и по заданному правилу формирует новый массив NEW_AR;

переписывает массив NEW_AR в массив AR;

формирует новое экранное изображение в зависимости от значений элементов массива AR.

П.5 повторяется до тех пор пока не будет нажата кнопка «Остановить», которая останавливает процесс формирования рисунка и меняет при этом название кнопки «Остановить» на «Продолжить».

Для продолжения формирования рисунка необходимо нажать кнопку «Продолжить», которая при этом изменяет свое название на «Остановить».

Можно изменить начальные значения количество строк (N_SIZE) и столбцов

(M_SIZE) в рисунке, количество цветов (K_SIZE) в палитре, для этого необходимо сначала нажать кнопку «Остановить», а затем поменять начальные значения и нажать кнопку «Начать».

Выйти из программы можно, если сначала остановить процесс формирования узора (нажать кнопку «Остановить»), а затем нажать кнопку «Выход».

Немного теории

Построение графических изображений осуществляется на поверхность объекта (формы или др.компонент). У ряда объектов из библиотеки визуальных компонент есть свойство **Canvas** . Для того чтобы вывести на поверхность объекта графический элемент (прямую линию, окружность, прямоугольник и т. д.), необходимо применить к свойству **Canvas** этого объекта соответствующий метод.

Canvas является в свою очередь объектом, объединяющим в себе поле для рисования, карандаш (**Pen**), кисть (**Brush**) и шрифт (**Font**). **Canvas** обладает также рядом графических методов: **LineTo**, **Ellipse**, **Rectangle**, **TextOut**, **Arc**, и др. Эти методы обеспечивают вывод графических примитивов (линий, окружностей, прямоугольников, текстов и т. д.), а свойства позволяют задать характеристики выводимых графических примитивов: цвет, толщину и стиль линий; цвет и вид заполнения областей; характеристики шрифта при

выводе текстовой информации.

Методы вывода графических примитивов рассматривают свойство **Canvas** как некоторый абстрактный холст, на котором они могут рисовать (*canvas* переводится как «поверхность», «холст для рисования»).

Холст состоит из отдельных точек – пикселей. Положение пикселя характеризуется его горизонтальной (X) и вертикальной (Y) координатами. Левый верхний пиксель имеет координаты (0, 0). Координаты возрастают сверху вниз и слева направо. Значения координат правой нижней точки холста зависят от размера холста.

Метод **Rectangle** рисует прямоугольник с одним углом в точке, указанной параметрами x1, y1, и противоположным диагональным углом в точке, заданной параметрами x2,y2. Рамка прямоугольника рисуется текущим пером (**Pen**) и заполняется текущим фоном

(**Brush**).

Объект.Canvas. Rectangle (x1,y1, x2,y2)

где: объект – имя объекта (компонента), на поверхности которого выполняется вычерчивание.

Свойство **Canvas** доступно только во время работы приложения.

Canvas.Brush – свойство фона.

Свойство **Canvas.Brush.Color** определяет необходимый цвет

Инструкция **Canvas.FillRect(ClientRect)** очищает всю рабочую область компонента, заливая ее установленным ранее цветом.

Свойство **Canvas.Brush.Bitmap** устанавливает рисунок в качестве фона – присвоить переменную (имя файла) с растровым рисунком.

Canvas.Pen – свойства пера.

Canvas.MoveTo(x,y) – устанавливает перо в заданную точку, где x и y - координаты точки, относительно компонента. После этой команды перо установлено, но точка не нарисована.

Canvas.LineTo(x,y) – провести линию от текущего положения пера до заданной точки

(x,y).

Canvas.Pixels[x,y]:=ЦВЕТ_ТОЧКИ – поставить на холсте точку с координатами (x,y) определенного цвета.

Canvas.Pen.Width:=ТОЛЩИНА_В_ТОЧКАХ.

Canvas.Pen.Color:=ЦВЕТ.

План разработки программы

Откройте новый проект.

Разместите в блоке реализации после слова **implementation** описание констант и переменных:

```
Const N_MAX=100; //Максимальное количество строк
      M_MAX=100; //Максимальное количество столбцов
      K_MAX=20;  //Максимальное количество цветов палитры
      RAZX=350;   //Размер поля рисунка  по оси X
      RAZY=350;   //Размер поля рисунка  по оси Y
      COLORS:array[1..20]of tcolor= //Массив палитры цветов
             (clblack, clmaroon, clgreen, clolive, clnavy,
             clpurple, clteal, clgray, clsilver, clred,
             cllime, clyellow, clblue, clfuchsia, claqua,
             clwhite, clmoneygreen, clskyblue, clcream, clmedgray);
Type AR_TYPE= Array[1..RAZX, 1..RAZY] of Integer;
Var N_SIZE : Integer;  // Количество строк узора
    M_SIZE : Integer;  // Количество столбцов узора
    K_SIZE : Integer;  // Количество цветов в узоре
    C_X : Integer;     // Размер прямоугольника (клетки)по X
    C_Y : Integer;     // Размер прямоугольника (клетки)по Y
    FLAG: Boolean;  // Флаг останова или продолжения построения
    AR, NEW_AR:AR_TYPE;
```

Массивы цветов до и после преобразования

Разместите в форме компоненты в соответствии с Рис.3. Для этого выделите объект

Form1, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**),

найдите событие **OnCreat**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
// Формирование элементов формы

Form1.Caption:='Узор';

Button1.Font.Size:=9;
```

```
Button1.Caption:='Начать';

Button2.Font.Size:=9;

Button2.Caption:='Остановить';

Button2.Enabled:=False;   //Кнопка недоступна

Button3.Font.Size:=9;

Button3.Caption:='Выход';

SpinEdit1.MinValue:=3;    //Минимальное количество строк

SpinEdit1.MaxValue:=N_MAX;  //Максимальное количество
SpinEdit1.Value:=10;      //Текущее значение

SpinEdit2.MinValue:=3;    //Минимальное количество столбцов

SpinEdit2.MaxValue:=M_MAX;  //Максимальное количество

SpinEdit2.Value:=10;      //Текущее значение

SpinEdit3.MinValue:=2;    //Минимальное количество

SpinEdit3.MaxValue:= K_MAX;  //Максимальное количество
SpinEdit3.Value:=3;       //Текущее значение
PaintBox1.Width:=RAZX;    //Определение размера поля PaintBox
PaintBox1.Height:=RAZY;
FLAG:= True;
```

4. Выделите объект **Button1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **OnClick** , справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
procedure TForm1.Button1Click(Sender:
TObject); var i,j:integer;

begin
```

6. Определение текущих значений

```
N_SIZE:=SpinEdit1.Valu
e;
M_SIZE:=SpinEdit2.Valu
e;
K_SIZE:=SpinEdit3.Valu
e;
```

7. Установка недоступности компонент

```
SpinEdit1.Enabled:=False;

SpinEdit2.Enabled:=False;

SpinEdit3.Enabled:=False;


Button2.Enabled:=True;   //Кнопка доступна
```

9. Заполнение элементов массива

```
Randomize;

for i:=1 to N_SIZE do
  for j:=1 to M_SIZE
  do

    AR[i,j]:= 1 + Random(K_SIZE);
```

10.Очистка поля для построения узора

```
Form1.PaintBox1.Canvas.Pen.Color:=clBtnFace;
Form1.PaintBox1.Canvas.Brush.Color:=
clBtnFace;
Form1.PaintBox1.Canvas.Rectangle(0,0,RAZX,RAZY
);
```

11.Определение размера одной клетки узора

```
C_X:=RAZX div N_SIZE;

C_Y:=RAZY div M_SIZE;


//Процедура первоначального формирования узора

IMAG(AR,N_SIZE,M_SIZE);


FLAG:=True;
```

```
 //Процедура формирования узора
 USOR;

end;
```

5. Разместите после блока описания переменных (см.п.2) две процедуры, которые были использованы в предыдущем п.4.

Процедура построения изображения узора

```
procedure
IMAG(AR:AR_TYPE;N,M:Integer); var
I,J:Integer;

begin

// Построение узора

 for  i:=1  to  N
  do for j:=1 to
  M do begin

    Form1.PaintBox1.Canvas.Pen.Color:= Colors[AR[i,j]];
    Form1.PaintBox1.Canvas.Brush.Color:= Colors[AR[i,j]];

Form1.PaintBox1.Canvas.Rectangle(C_X*(i-1),C_Y*(j-1),C_X*i-1,C_Y*j-
1);

   end;

end;
```

Пояснение

В процедуре организованы два цикла - по количеству строк и по количеству столбцов для перебора всех элементов массива AR. Для каждого элемента определяется цвет выводимого прямоугольника и его координаты.

Лабораторная работа №12, ОБРАБОТКА УЗОРА

Процедура обработки узора

Сначала приведем общий вид данной процедуры и переменные, которые будут использованы.

```
Procedure USOR;
```

```
Var C,  // Новый цвет текущей клетки
    U,  // Индекс клетки сверху
    D,  // Индекс клетки снизу
    R,  // Индекс клетки справа
    6. // Индекс клетки слева
       :Integer;
    i,j: Integer;       // Переменные цикла
begin
```

```
While FLAG do
// Бесконечный цикл, пока флаг остановки будет равен TRUE
begin
//Блок обработки массива AR
...

IMAG(AR,N_SIZE,M_SIZE)
; Sleep(100);

Application.ProcessMessages; //Обработка всей очереди
сообщений end;

end;
```

Пояснение

В процедуре используется бесконечный цикл **While FLAG do**, т.к. пероначально переменная **FLAG** равна **True**. Этот цикл сначала обрабатывает все элементы массива AR, а затем выполняет формирование нового изображения (процудура **IMAGE**). В конце цикла записаны две процедуры:

Sleep(100);

Application.ProcessMessages;

Примечание

- Процедура **Sleep** выполняет остановку программы на заданное время в милисекундах для того, чтобы мы смогли зафиксировать свой взгляд на изображении. **Sleep(100)** – задержка на 0,1 сек.
- Процедура **Application.ProcessMessages** заставляет программу взять все посланные приложению сообщения, которые находятся на данный момент в очереди сообщений, и обработать их. Это нам необходимо для того, чтобы можно было бы остановить работу программу.

Блок обработки элементов массива AR

```
for i:=1 to N_SIZE do for
   j:=1 to M_SIZE do

     begin

      // Определение нового цвета клетки

           ...

     end;
```

```
//Перезапись массива NEW_AR в AR
for i:=1 to N_SIZE do

   for j:=1 to M_SIZE do
      AR[i,j]:= NEW_AR[i,j];

IMAG(AR,N_SIZE,M_SIZE);
```

Пояснение

В начале записаны два вложенных цикла, которые позволяют просмотреть все значения элементов массива **AR** и сформировать новый массив **NEW_AR** с измененными значениями. Затем переписываем элементы нового массива **NEW_AR** в старый массив **AR** и передаем управление процедуре формирования нового узора на экране.

Текст блока определения нового цвета клетки

```
C:=AR[i,j]+1; // Вычисление следующего цвета клетки

if C>K_SIZE then C:=1; // После последнего цвета идет первый

// Вычисление индексов клеток
U:=i-1; // сверху

if U=0 then U:=N_SIZE;
D:=i+1; // снизу

if D>N_SIZE then D:=1;
L:=j-1; // слева

if L=0 then L:=M_SIZE;
R:=j+1; // справа

if R>M_SIZE then R:=1;
```

```
// Если среди соседей есть «следующий» цвет, то новая клетка

// приобретает этот цвет, в противном случае она не изменяется
if (AR[U,j]=C)or (AR[D,j]=C)or AR[i,L]=C)or (AR[i,R]=C)

    then  NEW_AR[i,j]:=C

    else  NEW_AR[i,j]:=AR[i,j];
```

---

6. Выделите объект **Button2**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **OnClick**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
If FLAG=True

Then
  begin

   FLAG:=False;

   Button2.Caption:='Продолжить'; //
  Установка доступности компонент

   SpinEdit1.Enabled:=True;

   SpinEdit2.Enabled:=True;

   SpinEdit3.Enabled:=True
  ; end

 Else

  begin

   FLAG:=True;

   Button2.Caption:='Остановить';

   // Установка недоступности компонент

   SpinEdit1.Enabled:=False;

   SpinEdit2.Enabled:=False;

   SpinEdit3.Enabled:=False;

   USOR;

  end;
```

---

7. Самостоятельно добавьте обработку кнопки «Выход».

8. Сохраните проект окончательно, запустите и протестируйте его.

Задание для самостоятельного выполнения

1. Если вы внимательно протестировали программу, то обратили внимание на не соответствие свойств Caption кнопок Button1 и Button2 после следующей последовательности нажатия кнопок: Button1 («Начать») - Button2 («Остановить») - Button1 («Начать»). Самостоятельно исправьте эту ошибку.

2. Напишите программу, которая будет выводить на экран цветные окружности, генерируя случайным образом координаты, радиус и цвет.

Внесите изменения в программу, чтобы на экран выводилось 10 окружностей, затем первая окружность стиралась и выводилась на экран следующая, которая становится 10-ой. Этот процесс повторяется до нажатия любой клавиши.

Лабораторная работа №13 Программирование разветвляющихся алгоритмов

***Цель лабораторной работы**:* освоить использование простейших компонентов-переключателей и создать приложение, которое использует разветвляющийся алгоритм.

2.1. Пример создания приложения

Задание: создать Windows-приложение для вычисления выражения
Z=

$$\begin{cases} f(x), x < y \\ y, иначе \end{cases}$$

, где

$$f(x) = \begin{cases} \sin(x) \\ \cos(x) \end{cases}$$

по желанию пользователя. В панели интерфейса предусмотреть возможность управления контрольным выводом исходных данных.

Один из возможных вариантов панели интерфейса создаваемого приложения показан на рис. 2.1.

2.1.1. Размещение компонентов на Форме

Будем размещать компоненты на Форме так, чтобы они соответствовали панели, показанной на рис 2.1.

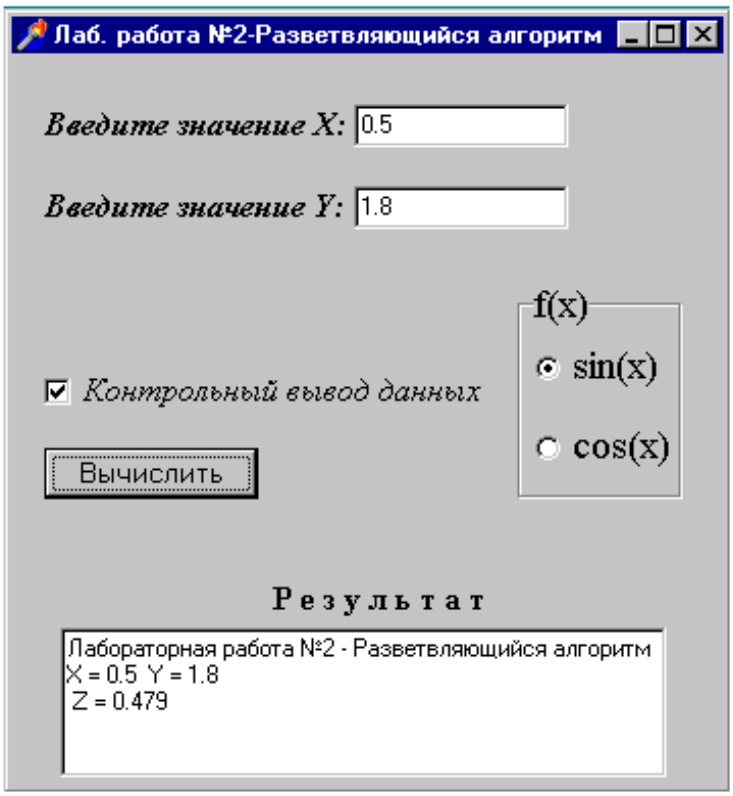


Рис. 2.1

CheckBox RadioGroup

При создании приложений в DELPHI часто используются компоненты в виде кнопок-переключателей. Состояние такой кнопки (включено - выключено) визуально отражается на Форме. На панели (рис. 2.1) представлены кнопки-переключатели двух типов: CheckBox и RadioGroup . Компонент **CheckBox** организует кнопку независимого переключателя, с помощью которой пользователь может указать свое решение типа “да/нет”. Компонент **RadioGroup** организует *группу кнопок* - зависимых переключателей. При нажатии одной из кнопок группы все остальные кнопки выключаются.
Поместите на Форму компоненты Label, Edit и Memo в соответствии с рис.2.1. Выберите в Палитре Компонентов на странице Standard пиктограмму

компонента CheckBox и разместите ее в нужном месте Формы. . В свойстве Caption Инспектора Объектов замените надпись **CheckBox1** на ***Контрольный вывод данных***. . Чтобы при запуске приложения кнопка CheckBox оказалась включена, свойство Checked установите равным True.

Выберите в Палитре Компонентов Standard пиктограмму

компонента RadioGroup и поместите ее в нужное место Формы. В свойстве Caption измените заголовок **RadioGroup1** на **f(x)**. Для размещения кнопок в один столбец, свойство Columns установите равным 1. Дважды щелкните “мышью” по правой части свойства Items - появится строчный редактор списка наименований кнопок. Наберите 2 строки с именами: в первой строке - sin(x), во второй - cos(x) и нажмите ОК. После этого на Форме появится группа из двух кнопок - переключателей с соответствующими надписями. Чтобы при запуске приложения первая кнопка RadioGroup оказалась включена, свойство ItemIndex установите равным 0.

2.1.2. Создание процедур обработки событий FormCreate и Button1Click

Технология создания процедур обработки событий FormCreate и Button1Click ничем не отличается от предыдущей работы. Внимательно наберите операторы этих процедур, используя текст модуля UnRazvAlg.

2.1.3. Текст модуля UnRazvAlg

```
Unit UnRazvAlg;


interface


uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, ExtCtrls;


type
TForm1 = class(TForm)
Label1: TLabel;
Edit1: TEdit;
Label2: TLabel;
Edit2: TEdit;
Label4: TLabel;
Memo1: TMemo;
Button1: TButton;
RadioGroup1: TRadioGroup;
CheckBox1: TCheckBox;
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;


var
Form1: TForm1;


implementation
{$R *.DFM}
// Процедура обработки события создания Формы:
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Text:='0.5'; // начальное значение X
Edit2.Text:='1.8'; // начальное значение Y
Memo1.Clear; // очистка Memo1
// Вывод строки в Memo1:
Memo1.Lines.Add('Лабораторная работа №2 - Разветвляющийся
алгоритм');
end;
// Процедура обработки события нажатия кнопки Button1:
procedure TForm1.Button1Click(Sender: TObject);
var
x,y,z,fx : extended; // объявление локальных переменных
begin
```

```
x:=StrToFloat(Edit1.Text); // X присваивается содержимое Edit1
y:=StrToFloat(Edit2.Text); // Y присваивается содержимое Edit2
fx:=sin(x); // fx присваивается начальное значение
// Выбор функции, соответствующей нажатой кнопке:
case RadioGroup1.ItemIndex of
0:fx:=sin(x);
1:fx:=cos(x);
end;
// Вычисление выражения:
if xthen
z:=fx
else
z:=y;
// Проверка состояния кнопки CheckBox1:
if CheckBox1.Checked then
Memo1.Lines.Add('X = '+Edit1.Text+
' Y = '+Edit2.Text); // контрольный вывод X, Y в Memo1
// Вывод результата в Memo1:
Memo1.Lines.Add(' Z = '+FloatToStrF(z,ffFixed,8,3));
end;
end.
```

Если нажата первая кнопка RadioGroup1, в переменную целого типа RadioGroup1.ItemIndex заносится нуль, если вторая – единица. Если кнопка CheckBox1 нажата, логическая переменная CheckBox1.Checked имеет значение True, если нет – False.

2.1.4. Работа с приложением

Запустите созданное приложение. Используя все управляющие компоненты панели интерфейса, убедитесь в правильном функционировании приложения во всех предусмотренных режимах работы.

2.2. Выполнение индивидуального задания

По указанию преподавателя выберите свое индивидуальное задание. Создайте приложение и протестируйте его работу.

Индивидуальные задания

Для заданий №1-№15 на панели интерфейса предусмотреть возможность выбора одной из трех функций f(x): sh(x), x^2, e^x.

1.
$$a=\begin{cases}(f(x)+y)^2-f(x)y, & x\le 0\\ (f(x)+y)^3+f(x)y, & 0<x\le 1\\ (f(x)+y)^4+1, & \text{иначе.}\end{cases}$$

2.
$$b=\begin{cases}\ln(f(x))+(f(x)^2+y)^3, & x/y\ge 0\\ \ln f(x)/y+(f(x)+y)^2, & x/y<0\\ (f(x)^3+y)^2, & -2\le x<2\\ 0, & \text{иначе.}\end{cases}$$

3. $$c = \begin{cases} f(x)^2 + y^2 + \sin(y), & x + y > 0 \\ (f(x) - y^3)^2 + \cos(y), & x \ge 0, y < 0 \\ (y - f(x))^2 + tg(y), & x - y < 0. \end{cases}$$

4. $$d = \begin{cases} f^2(x) + arctg(f(x)), & 1 \le x < 5 \\ (y - f(x))^2 + arctg(f(x)), & y \ge x \\ (y + f(x))^3 + 0.5, & \text{иначе.} \end{cases}$$

5. $$e = \begin{cases} i\sqrt{f(x)}, & i - \text{нечетное}, x \ge 0 \\ i/2\sqrt{|f(x)|}, & i - \text{четное}, x < 0 \\ \sqrt{|if(x)|}, & \text{иначе} \end{cases}$$

6. $$g = \begin{cases} e^{f(x)-|b|}, & 0.5 < x \le 10 \\ \sqrt{|f(x)+b|}, & 0.1 < x \le 0.5 \\ 2f(x)^2, & \text{иначе} \end{cases}$$

7. $$s = \begin{cases} e^{f(x)}, & 1 < x \le 10 \\ \sqrt{|f(x)+4*b|}, & 12 < x \le 40 \\ bf(x)^2, & \text{иначе} \end{cases}$$

8. $$j = \begin{cases} \sin(5f(x) + 3m\, f(x)), & -1 < m < x \\ \cos(3f(x) + 5m\, f(x)), & x \le m \\ (f(x) + m)^2, & \text{иначе.} \end{cases}$$

Лабораторная работа №14 Программирование циклических алгоритмов

Цель лабораторной работы: освоить простейшие средства отладки модулей проекта и создать приложение, которое использует циклический алгоритм.

3.1. Отладка модулей проекта

Отладка представляет собой процесс обнаружения, локализации и устранения ошибок в проекте. Она занимает значительную часть рабочего времени программиста, нередко большую, чем разработка проекта.

Практически любой нетривиальный проект перед началом отладки содержит хотя бы одну синтаксическую или логическую ошибку.

3.1.1. Отладка синтаксических ошибок

Синтаксические ошибки состоят в нарушении формальных правил использования операторов. Эти ошибки появляются в результате недостаточного знания разработчиком языка программирования и невнимательности при наборе операторов на экране дисплея.

Поиск синтаксических ошибок в модулях проекта осуществляется компилятором. Чтобы дать программисту как можно больше информации об ошибках, допущенных в модуле, компилятор отмечает ошибки и продолжает работу до тех пор, пока не будут обработаны все операторы модуля. Следует иметь в виду, что:

1. компилятор распознает *не все ошибки*;
2. некоторые ошибки могут повлечь за собой то, что правильные операторы будут восприниматься компилятором как ошибочные, и наоборот – ошибочные операторы компилятор воспримет как верные;
3. ошибка в одном месте модуля может повлечь за собой серию диагностических сообщений компилятора в других местах модуля;
4. из-за некоторых ошибок компиляция модуля может вообще прекращаться и проверка последующих операторов не производится.

Информация обо всех ошибках, найденных в модуле, выводится в специальное окно, которое появляется в нижней части экрана. Каждая строка этого окна содержит имя файла, номер строки, в которой обнаружена ошибка и характер ошибки. Если дважды щелкнуть “мышью” на строке с описанием ошибки, курсор

установится в той строке модуля, где обнаружена ошибка. Следует исправлять ошибки последовательно, сверху вниз и после исправления каждой ошибки компилировать программу заново. С целью сокращения времени компиляции рекомендуется осуществлять проверку наличия ошибок в режимах Syntax Check и Compile меню Project. Для получения более полной информации о характере ошибки можно обратится к HELP нажатием клавиши F1.
Отладка синтаксиса считается завершенной, когда после очередной компиляции в режиме Build All меню Project отсутствуют диагностические сообщения об ошибках.

3.1.2. Отладка логических ошибок

Логические ошибки условно можно разделить на ошибки алгоритма и семантические ошибки. Причинами таких ошибок могут быть несоответствие алгоритма поставленной задаче, неправильное понимание программистом смысла (семантики) операторов языка программирования, нарушение допустимых пределов и правил представления данных, невнимательность при технической подготовке проекта к обработке на компьютере.
Для выявления ошибок служат *тесты*. Тест – это такой набор исходных данных, который дает результат, не вызывающий сомнений. Промежуточные и конечные результаты теста используются для контроля правильности выполнения приложения.
Составление тестов – непростая задача. Тесты должны быть с одной стороны, достаточно простыми, чтобы результат легко проверялся, с другой стороны – достаточно сложными, чтобы комплексно проверить алгоритм.
Тесты составляются по схеме алгоритма *до программирования*, так как составление тестов помогает выявить многие ошибки в алгоритмизации.
Количество тестов и их сложность зависят от алгоритма. Комплекс тестов должен быть таким, чтобы все ветви схемы алгоритма были пройдены, по крайней мере, по одному разу. Несовпадение результатов, выдаваемых приложением с результатами тестов – признак наличия ошибок. Эти ошибки проявляются в том, что результат расчета оказывается неверным либо происходит переполнение, деление на 0 и др.
Для локализации места ошибки рекомендуется поступать следующим образом. В окне Редактора Кода установите курсор в строке перед подозрительным участком и нажмите клавишу F4 (выполнить до курсора). Выполнение приложения будет остановлено на той строке модуля, в которой был установлен курсор. Текущее значение любой переменной можно увидеть, если накрыть курсором идентификатор переменной на 1-2 сек. Нажимая клавишу F8 (пошаговое выполнение), можно построчно выполнять программу, контролируя содержимое переменных и правильность вычислений.

3.2. Пример создания приложения

Задание: создать Windows-приложение, которое выводит таблицу значений функции

$$Y(x) = (1 - \frac{x^2}{2})\cos x - \frac{x}{2}\sin x$$

и ее разложения в ряд в виде суммы

$$S(x) = \sum_{n=0}^{n} (-1)^n \frac{2n^2 + 1}{(2n)!} x^{2n}$$

для значений x от x_n до x_k с шагом $h=(x_k - x_n)/10$.
В панели интерфейса предусмотреть возможность управления выводом исходных данных и погрешности вычислений.
Один из возможных вариантов панели интерфейса создаваемого приложения показан на рис.3.1.

3.2.1. Размещение компонентов на Форме

Вместо компонента Edit используем компонент SpinEdit, который обеспечивает отображение и редактирование целого числа с возможностью его изменения посредством двойной кнопки.

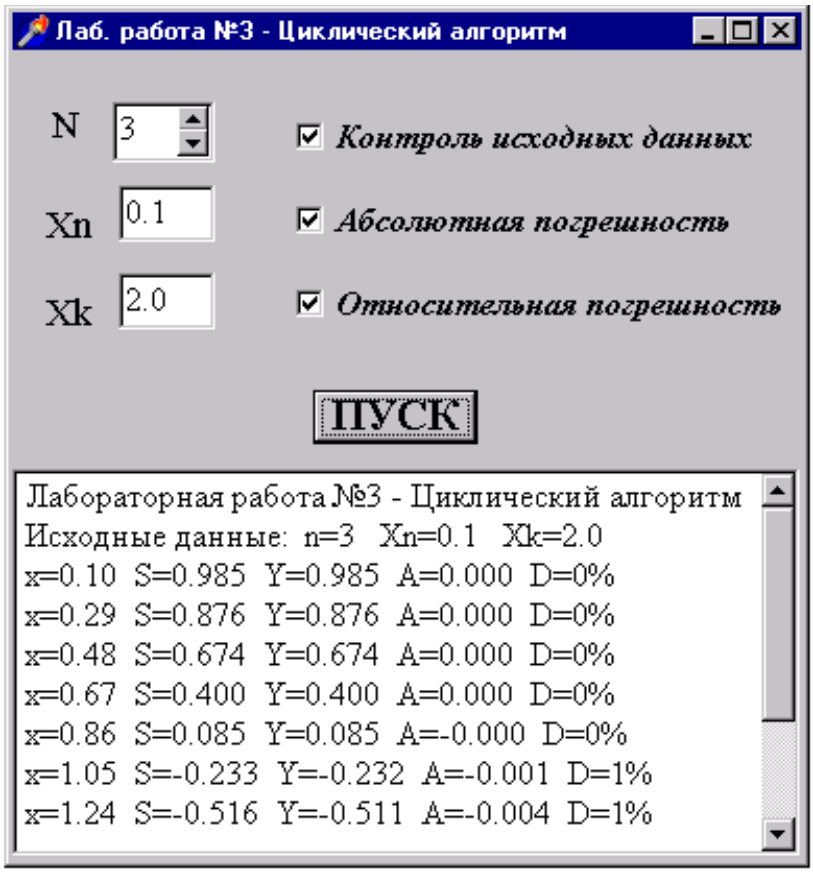


Рис. 3.1

Компонент SpinEdit находится на странице Samples Палитры Компонентов. В тех случаях, когда объем выводимой информации превышает размер поля компонента Memo, целесообразно снабдить его линейками прокрутки. В свойстве ScrollBars компонента Memo1 установим значение ssVertical – появится вертикальная линейка прокрутки. Присвоим модулю имя UnCiklAlg.

3.2.2. Текст модуля UnCiklAlg

```
Unit UnCiklAlg;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, ExtCtrls, Spin;
type
TForm1 = class(TForm)
Memo1: TMemo;
Button1: TButton;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Edit1: TEdit;
Edit2: TEdit;
SpinEdit1: TSpinEdit;
CheckBox1: TCheckBox;
CheckBox2: TCheckBox;
```

```
CheckBox3: TCheckBox;
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
SpinEdit1.text:='3'; // начальное значение N
Edit1.text:='0.1'; // начальное значение Xn
Edit2.text:='2.0'; // начальное значение Xk
Memo1.Clear;
Memo1.Lines.Add('Лабораторная работа №3 - Циклический алгоритм');
end;
procedure TForm1.Button1Click(Sender: TObject);
var xn,xk,x,h,c,s,y,al,del:extended;
n,k:integer;
begin
n:=StrToInt(SpinEdit1.Text);
xn:=StrToFloat(Edit1.Text);
xk:=StrToFloat(Edit2.Text);
if CheckBox1.Checked then
Memo1.Lines.Add('Исходные данные: n='+IntToStr(n)+
' Xn='+FloatToStrF(xn,ffFixed,6,1)+
' Xk='+FloatToStrF(xk,ffFixed,6,1));
h:=(xk-xn)*0.1; // шаг h
x:=xn;
repeat // цикл по x
c:=-x*x*0.5;
S:=1;
for k:=1 to n do
begin
s:=s+c*(2*k*k+1);
c:=-c*x*x/((2*k+1)*(2*k+2));
end;
y:=(1-x*x*0.5)*cos(x)-0.5*x*sin(x);
if CheckBox2.Checked then
if CheckBox3.Checked then
begin
al:=s-y; // абсолютная погрешность
del:=abs((s-y)/y)*100; // относительная погрешность
Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
' S='+ FloatToStrF(s,ffFixed,6,3)+
' Y='+ FloatToStrF(y,ffFixed,6,3)+
' A='+ FloatToStrF(al,ffFixed,6,3)+
' D='+ FloatToStrF(del,ffFixed,6,0)+'%');
end
else
begin
al:=s-y;
```

```
Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
' S='+ FloatToStrF(s,ffFixed,6,3)+
' Y='+ FloatToStrF(y,ffFixed,6,3)+
' A='+ FloatToStrF(al,ffFixed,6,3));
end
else
if CheckBox3.Checked then
begin
del:=abs((s-y)/y)*100;
Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
' S='+ FloatToStrF(s,ffFixed,6,3)+
' Y='+ FloatToStrF(y,ffFixed,6,3)+
' D='+ FloatToStrF(del,ffFixed,6,0)+'%');
end
else
Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
' S='+ FloatToStrF(s,ffFixed,6,3)+
' Y='+ FloatToStrF(y,ffFixed,6,3));
x:=x+h;
until x>xk;
end;
end.
```

3.3. Выполнение индивидуального задания

По указанию преподавателя выберите свое индивидуальное задание. Создайте приложение и протестируйте его работу.

Индивидуальные задания

В заданиях с №1 по №15 необходимо вывести на экран таблицу значений функции Y(x) и ее разложения в ряд S(x) для значений x от x_n до x_k с шагом $h = (x_k - x_n) / 10$. Близость значений S(x) и Y(x) во всем диапазоне значений x указывает на правильность вычисления S(x) и Y(x).

№	x_n	x_k	S(x)	n	Y(x)
1	0.1	1	$x - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$	16	$\sin x$
2	0.1	1	$1 + \frac{x^2}{2!} + \ldots + \frac{x^{2n}}{(2n)!}$	10	$\frac{e^x + e^{-x}}{2}$
3	0.1	1	$1 + \frac{\cos\frac{\pi}{4}}{1!}x + \ldots + \frac{\cos n\frac{\pi}{4}}{n!}x^n$	12	$e^{x\cos\frac{\pi}{4}}\cos(x\sin\frac{\pi}{4})$

4	0.1	1	$1-\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$	8	$\cos x$
5	0.1	1	$1-\frac{x^2}{1!}+\frac{x^4}{2!}-\frac{x^6}{3!}+\ldots+(-1)^n\frac{x^{2n}}{n!}$	14	e^{-x^2}
6	0.1	1	$x+\frac{x^3}{3!}+\ldots+\frac{x^{2n+1}}{(2n+1)!}$	8	$\frac{e^x-e^{-x}}{2}$
7	0.1	1	$\frac{x^3}{3}-\frac{x^5}{15}+\ldots+(-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$	12	$\frac{1+x^2}{2}arctgx\frac{x}{2}$
8	0.1	1	$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	10	e^{2x}
9	0.1	1	$1+2\frac{x}{2}+\ldots+\frac{n^2+1}{n!}\frac{x}{2}^n$	14	$\frac{x^2}{4}+\frac{x}{2}+1e^{\frac{x}{2}}$
10	0.1	0.5	$x-\frac{x^3}{3}+\ldots+(-1)^n\frac{x^{2n+1}}{2n+1}$	15	$arctg,$
11	0.1	0.8	$\frac{x^2}{2}-\frac{x^4}{12}+\ldots+(-1)^{n+1}\frac{x^{2n}}{2n(2n-1)}$	10	$xarctgx\ln\sqrt{1+x^2}$
12	0.1	1	$-\frac{(2x)^2}{2}+\frac{(2x)^4}{24}-\ldots+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	8	$2(\cos^2 x-1)$

Лабораторная работа №15, 16 Программирование АЛГОРИТМОВ с использованием массивов

***Цель лабораторной работы**:* освоить применение компонента StringGrid и создать приложение, в котором используются массивы.

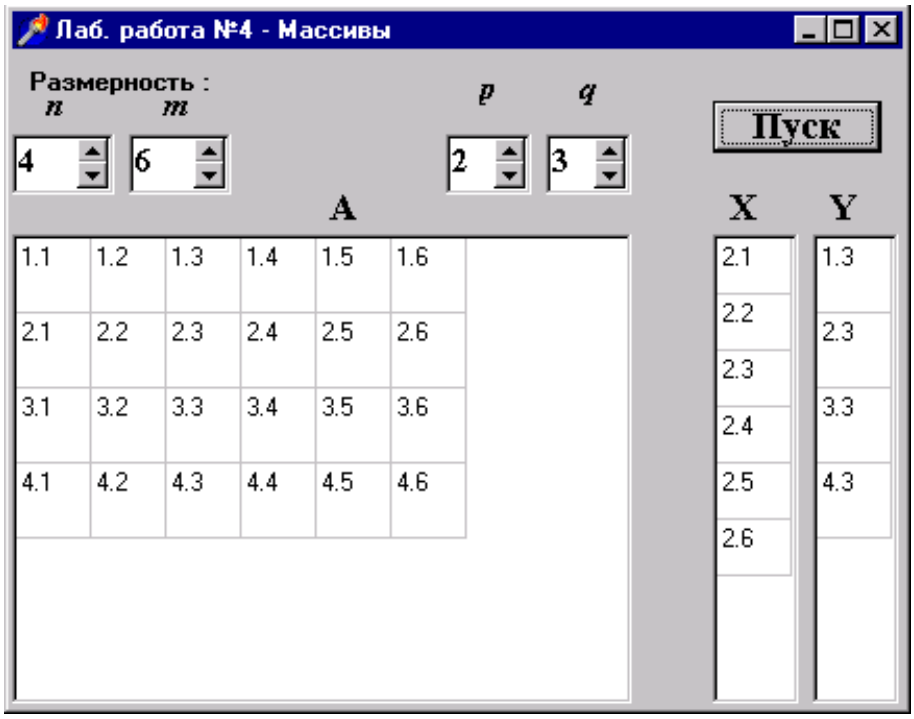
4.1. Пример создания приложения

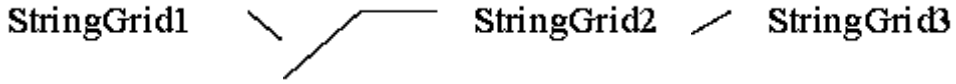
Задание: создать Windows-приложение для вычисления вектора $x=\{x_1,x_2,\ldots,x_m\}$,равного p-й строке матрицы $A=\{a_{ij}\}(x_j=a_{pj},\ j=1,2,\ldots,m)$ и вектора $y=\{y_1,y_2,\ldots,y_n\}$, равного q-му столбцу матрицы $A=\{a_{ij}\}(y_i=a_{iq},\ i=1,2,\ldots,n)$ $(n\le 6,m\le 8)$. В панели интерфейса предусмотреть возможность управления размерностью массивов.

Один из возможных вариантов панели интерфейса создаваемого приложения показан на рис.4.1.

4.1.1. Размещение компонентов на Форме

При работе с массивами ввод и вывод информации на экран удобно





организовывать с помощью компонента StringGrid.

Рис. 4.1

Компонент **StringGrid** используется для отображения информации в виде таблицы. Таблица содержит две зоны – фиксированную и рабочую.
Фиксированная зона служит для вывода наименований строк и столбцов рабочей зоны и управления их размерами с помощью “мыши”. Фиксированная зона выделена другим цветом и в нее запрещен ввод информации с клавиатуры. Количество строк и столбцов фиксированной зоны устанавливается в свойствах FixedRows и FixedCols, соответственно.
Рабочая зона содержит RowCount строк и ColCount столбцов информации, которую можно изменять как программно, так и с помощью “мыши” или клавиатуры.
Доступ к информации в программе осуществляется с помощью свойства Cells[ACol, ARow: integer]: string, где ACol-номер столбца, а ARow – номер строки таблицы, причем *нумерация начинается с нуля.*
Пиктограмма

компонента StringGrid находится на странице Additional Палитры Компонентов. Так как в нашем задании для всех компонентов StringGrid фиксированная зона не используется, в Инспекторе Объектов значения свойств FixedCols и FixedRows установите равными 0. В соответствии с заданием установите предельные значения количества строк n и столбцов m для компонента StringGrid1: ColCount=8, а RowCount=6 (восемь столбцов и шесть строк). Для компонента StringGrid2 ColCount=1, RowCount=8, а для компонента StringGrid3 ColCount=1, RowCount=6.
По умолчанию в компонент StringGrid запрещен ввод информации с клавиатуры, поэтому для компонента StringGrid1 необходимо в Инспекторе Объектов дважды щелкнуть “мышью” на символе + свойства +Options и в открывшемся списке опций установить значение goEditing в True.

Для удобства работы с компонентами SpinEdit установите для компонента SpinEdit1 значения свойств: MinValue=1, MaxValue=6, а для компонента SpinEdit2: MinValue=1, MaxValue=8.

4.1.2. Создание процедур обработки событий SpinEdit1Change и SpinEdit2Change

События SpinEdit1Change и SpinEdit2Change возникают при любом изменении значения в поле редактора SpinEdit1 и SpinEdit2 соответственно. Создадим процедуры обработки этих событий, в которых присвоим значения n и m, полученные из полей редакторов SpinEdit, свойствам ColCount и RowCount компонентов StringGrid. Это позволит управлять размерами таблиц StringGrid с помощью компонентов SpinEdit без дополнительных кнопок, так как изменение значений в поле редактора SpinEdit сразу приведет к изменению размера таблиц StringGrid. Дважды щелкните “мышью” на компоненте SpinEdit1 – курсор установится в тексте процедуры-обработчика события SpinEdit1Change: **procedure TForm1.SpinEdit1Change(Sender: TObject)**. Внимательно наберите операторы этой процедуры, используя текст модуля UnMas(см. п.4.1.3)**.** Аналогичным образом создайте процедуру-обработчик события SpinEdit2Change: **procedure TForm1.SpinEdit2Change(Sender: TObject)**.

4.1.3. Текст модуля UnMas

```
Unit UnMas;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Spin, Grids;

type
TForm1 = class(TForm)
Label1: TLabel;
SpinEdit1: TSpinEdit;
SpinEdit2: TSpinEdit;
Label8: TLabel;
StringGrid1: TStringGrid;
StringGrid2: TStringGrid;
StringGrid3: TStringGrid;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
SpinEdit3: TSpinEdit;
SpinEdit4: TSpinEdit;
Label6: TLabel;
Label7: TLabel;
Button1: TButton;
procedure FormCreate(Sender: TObject);
procedure SpinEdit1Change(Sender: TObject);
procedure SpinEdit2Change(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
```

```
{$R *.DFM}
var
A:array[1..6,1..8] of extended;// объявление двумерного массива A
X:array[1..8] of extended; // объявление одномерного массива X
Y:array[1..6] of extended; // объявление одномерного массива Y
n,m,p,q:integer; // объявление глобальных переменных
procedure TForm1.FormCreate(Sender: TObject);
begin
SpinEdit1.Text:='4'; // начальное значение n
SpinEdit2.Text:='6'; // начальное значение m
SpinEdit3.Text:='2'; // начальное значение p
SpinEdit4.Text:='3'; // начальное значение q
StringGrid1.RowCount:=4; // количество строк массива A
StringGrid1.ColCount:=6; // количество столбцов массива A
StringGrid2.RowCount:=6; // количество строк массива X
StringGrid3.RowCount:=4; // количество строк массива Y
end;
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
n:=StrToInt(SpinEdit1.Text);// n присваивается содержимое поля редактора
StringGrid1.RowCount:=n; // устанавливается количество строк массива A
StringGrid3.RowCount:=n; // устанавливается количество строк массива Y
end;
procedure TForm1.SpinEdit2Change(Sender: TObject);
begin
m:=StrToInt(SpinEdit2.Text);// m присваивается содержимое поля редактора
StringGrid1.ColCount:=m; // устанавливается количество столбцов массива A
StringGrid2.RowCount:=m; // устанавливается количество строк массива X
end;
procedure TForm1.Button1Click(Sender: TObject);
var
i,j:integer; // объявление локальных переменных
begin
n:=StrToInt(SpinEdit1.Text);
StringGrid1.RowCount:=n;
StringGrid3.RowCount:=n;
m:=StrToInt(SpinEdit2.Text);
StringGrid1.ColCount:=m;
StringGrid2.RowCount:=m;
p:=StrToInt(SpinEdit3.Text);
q:=StrToInt(SpinEdit4.Text);
// Ввод значений из таблицы в массив A
for i:=1 to n do
for j:=1 to m do
A[i,j]:=StrToFloat(StringGrid1.Cells[j-1,i-1]);
for j:=1 to m do // формирование массива X и вывод его значений в таблицу
begin
X[j]:=A[p,j];
StringGrid2.Cells[0,j-1]:=FloatToStrF(X[j],ffFixed,3,1);
end;
for i:=1 to n do // формирование массива Y и вывод его значений в таблицу
begin
Y[i]:=A[i,q];
StringGrid3.Cells[0,i-1]:=FloatToStrF(Y[i],ffFixed,3,1);
end;
end;
```

end.

4.1.4. Работа с приложением

Запустите созданное приложение. Занесите числовые значения в элементы матрицы A и убедитесь в том, что приложение функционирует в соответствии с заданием.

4.2. Выполнение индивидуального задания

Изучите в приложении 2 описание компонентов StringGrid и DrawGrid.
По указанию преподавателя выберите свое индивидуальное задание. Создайте приложение и протестируйте его работу.

Индивидуальные задания

1. Задана целочисленная матрица *A* размером *NxM.* Получить массив *B*, присвоив его *k*-му элементу значение *0*, если все элементы *k*-го столбца матрицы нулевые, и значение *1* в противном случае(*k=1,2,…,M*).
2. Задана целочисленная матрица *A* размером *NxM.* Получить массив *B*, присвоив его *k*-му элементу значение *1*, если элементы *k*–й строки матрицы упорядочены по убыванию, и значение *0* в противном случае(*k=1,2,…,N*).
3. Задана целочисленная матрица *A* размером *NxM.* Получить массив *B*, присвоив его *k*-му элементу значение *1*, если *k*-я строка матрицы симметрична, и значение *0* в противном случае(*k=1,2,…,N*)..
4. Задана целочисленная матрица размером *NxM.* Определить *k*–количество “особых” элементов матрицы, считая элемент “особым”, если он больше суммы остальных элементов своего столбца.
5. Задана целочисленная матрица размером *NxM.* Определить *k*–количество “особых” элементов матрицы, считая элемент “особым”, если в его строке слева от него находятся элементы, меньшие его, а справа – б*о*льшие.
6. Задана символьная матрица размером *NxM*. Определить *k*-количество различных элементов матрицы (т.е. повторяющиеся элементы считать один раз).
7. Дана вещественная матрица размером *NxM*. Упорядочить ее строки по неубыванию их первых элементов.
8. Дана вещественная матрица размером *NxM*. Упорядочить ее строки по неубыванию суммы их элементов.
9. Дана вещественная матрица размером *NxM*. Упорядочить ее строки по неубыванию их наибольших элементов.
10. Определить является ли заданная квадратная матрица *n*-го порядка симметричной относительно побочной диагонали.
11. Для заданной целой матрицы размером *NxM* вывести на экран все ее седловые точки. Элемент матрицы называется седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или, наоборот, является наибольшим в своей строке и наименьшим в своем столбце.
12. В матрице *n*-го порядка переставить строки так, чтобы на главной диагонали матрицы были расположены элементы, наибольшие по абсолютной величине.

Рекомендуемая литература:

Основная литература:

1. Лубашева Т.В. Основы алгоритмизации и программирования [Электронный ресурс] : учебное пособие / Т.В. Лубашева, Б.А. Железко. — Электрон. текстовые данные. — Минск: Республиканский институт профессионального образования (РИПО), 2016. — 379 c. — 978-985-503-625-9. — Режим доступа: http://www.iprbookshop.ru/67689.html

2. Литвин Д.Б., Мелешко С.В., Мамаев И.И. Элементы теории игр и нелинейного программирования: Учебное пособие 2 УДК 51 (075.8) ББК 22.1я73 Литвин, Д.Б. Элементы теории игр и нелинейного программирования: Учебное пособие / Д.Б. Литвин, С.В. Мелешко, И.И. Мамаев. Ставрополь : Сервисшкола, 2017. 84с.

3. Борисенко В.В. Основы программирования [Электронный ресурс]/ Борисенко В.В.— Электрон. текстовые данные.— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 323 c.— Режим доступа: http://www.iprbookshop.ru/52206.— ЭБС «IPRbooks»

Дополнительная литература:

1. Борисенко, В. В. Основы программирования / В. В. Борисенко. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 323 c. — ISBN 978-5-9556-00039-0. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/52206.html

2. Зайцев М.Г. Програмирование. Структурное програмирование, подпрограммы, строки: учеб. Пособие Лебеденко Зайцев М.Г. З-177 Програмирование. Структурное програмирование, подпрограммы, строки : учеб. пособие / М.Г. Зайцев. Новосибирск : Изд-во НГТУ, 2016. 103 с.