Министерство образования и науки Украины

Донбасская государственная машиностроительная академия

О. А. Медведева

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ЛАБОРАТОРНЫМ РАБОТАМ ПО ДИСЦИПЛИНЕ «ИНФОРМАТИКА»

Работа с основными компонентами визуальной среды Lazarus

для студентов технических специальностей
всех форм обучения

УТВЕРЖДЕНО
на заседании
методического семинара кафедры
прикладной математики
Протокол № 9 от 16.05.2013

Краматорск
ДГМА
2013

СОДЕРЖАНИЕ

Лабораторная работа № 1.

Знакомство с визуальной средой *Lazarus*, создание простейшего приложения.

Цель: Изучение среды ***Lazarus,*** освоение основных приемов работы в среде. Изучение основных компонентов. Работа со свойствами компонентов. Проектирование простейших обработчиков событий.

Рабочее задание: Провести исследование рабочей среды ***Lazarus*** и основных элементов управления. Создать простейшее приложение в среде ***Lazarus***, в котором пользователь вводит произвольный текст и при щелчке на командной кнопке выводит этот текст на экран, добавляя перед ним постоянную фразу “**Привет! Моя первая программа**”. Пользователь должен иметь возможность управления цветом и параметрами шрифта отображаемого текста.

Краткие теоретические сведения

Основным строительным блоком в ***Lazarus*** является форма. На форме располагаются различные компоненты – элементы управления. Каждый компонент относится к какому-то типу. Перечень типов компонент расположен в палитрах компонент. На первом этапе знакомства с ***Lazarus*** будут использоваться в основном компоненты из палитры **Standard**, поэтому

в этом случае не будем оговаривать местоположение прототипа (образца) компонента. В противном случае местоположение (палитра) оговаривается.

Каждый компонент имеет набор присущих типу, к которому он относится, ***свойств, методов и событий***, на которые он может реагировать.

Свойства определяют внешний вид, положение и поведение компонента. Перечень всех свойств, доступных для текущего компонента на этапе проектирования содержится в окне *Инспектора объектов.*

В момент создания компонента ***Lazarus*** автоматически присваивает всем его свойствам некоторые начальные значения (значения по умолчанию). Если эти значения не подходят, их можно изменить. Для этого выделить нужное свойство и в поле для значения ввести новое значение.

Среди всех свойств компонентов особенно выделяется свойство **Name** – внутреннее имя компонента. Это свойство имеет каждый компонент (в том числе и форма). Обычно ***Lazarus*** автоматически формирует это свойство, добавляя к имени типа компонента его порядковый номер в пределах типа. Первый символ (“T”) при этом удаляется. Конечно, это имя можно изменить. Однако следует иметь в виду, что для имени компонента можно использовать только латинские символы, цифры и некоторые специальные символы (например, символ подчеркивания). Символы кириллицы и пробелы использовать нельзя. Это связано с тем, что имя компонента, указанное в свойстве **Name** используется для ссылок на компонент в программном коде (например, для доступа к его свойствам). В этой связи с именем каждого компонента надо определиться на ранней стадии разработки проекта, еще до написания программного кода, чтобы не пришлось корректировать код при изменении имен компонент. Это касается и форм, к именам которых предъявляются определенные требования (см. выше).

Многие компоненты имеют свойство **Caption** или сходное с ним по назначению. Это свойство может содержать любые символы, включая символы кириллицы. Оно обычно используется для идентификации компонента на этапе выполнения приложения.

Значения свойств (кроме **Name**) могут изменяться как на этапе проектирования, так и на этапе выполнения проекта (т. е. программно).

Событие – это какое-то воздействие на компонент, например, щелчок клавишей мыши, двойной щелчок, потеря фокуса, изменение размера и т.д., на которое компонент в состоянии реагировать. Реакция на событие реализуется в подключаемой для этого подпрограмме – ***обработчике события***. Для обработки каждого события разрабатывается своя подпрограмма. *Если для какого-то события обработчик события не подключен, компонент не будет реагировать на это событие.* В частности, только созданный компонент еще не имеет ни одного разработчика событий. Поэтому он не реагирует ни на какие действия пользователя (события). Перечень всех событий для текущего компонента тоже находится в окне *Инспектора объектов.* Однако для доступа к событиям необходимо перейти

на вкладку **Events (События).** Чтобы подключить требуемый обработчик события, необходимо в окне *Инспектора объектов* выбрать соответствующее событие и сделать двойной щелчок в правой колонке (в этой колонке показываются имена уже подключенных обработчиков).

Пожалуй, наиболее часто используемое событие – **Click**, которое наступает для компонента при щелчке на нем.

Методы компонентов – это заранее написанные подпрограммы, обращение к которым (вызов) позволяет воздействовать на компонент, например, изменить некоторые его свойства. Методы вызываются из программных кодов.

В лабораторной работе будут использованы наиболее употребляемые в приложениях компоненты (элементы управления) следующих типов:

командные кнопки – ***TButton*** для управления работой приложения;

специализированная командная кнопка – ***TBitBtn*** для тех же целей;

однострочный текстовый редактор (поле ввода) – ***TEdit*** для ввода данных (текста);

метка – ***TLabel*** для вывода постоянного текста (метки);

многострочный текстовый редактор – ***TMemo*** для отображения на экране результата (итогового текста);

флажки (***TCheckBox***) для установки или снятия опций (параметров) шрифта, которым будет отображаться результат; если флажок установлен, его свойство **State** имеет значение ***cbChecked*.**

переключатели **–** ***TRadioButton*** для выбора одного из предлагаемых вариантов цвета отображаемого итогового текста; свойство **Checked** включенного переключателя имеет значение ***true***; у остальных переключателей – ***false*.**

Текст, введенный пользователем в компоненте ***TEdit***, содержится в свойстве **Text** этого компонента. В компонент ***TMemo*** данные помещаются с помощью метода **Add** этого компонента. Все параметры шрифта отображаемого в компоненте ***TMemo*** текста содержатся в его свойстве **Font.** Это свойство, в свою очередь, тоже является объектом (сложным свойством) и тоже имеет ряд своих свойств:

- **Color** – цвет (заносится цвет, соответствующий выбранному переключателю);
- **Size** – размер шрифта (будет вводиться в диалоге функцией **InputBox**);
- **Style** – набор стилей (множество), в который можно занести в любом сочетании следующие стили (заносятся в том случае, когда включен соответствующий компонент ***TCheckBox***):
  - ***FsBold*** – жирный;
  - ***FsItalic*** – курсив;
  - ***FsUnderLine*** – подчеркнутый;
  - ***FsStriceOut*** – зачеркнутый.

Среди всего множества компонентов, предлагаемых ***Lazarus***, можно выделить компоненты-контейнеры. Это такие компоненты, на которых можно размещать другие компоненты. Размещенные в контейнере компоненты перемещаются, скрываются или удаляются вместе с самим контейнером. Такие компоненты (контейнеры) позволяют группировать компоненты, объединять их по определенному признаку. Форма является одним из таких контейнеров (но самого высокого уровня). Кроме того, в лабораторной работе будут использованы также контейнеры типа:

TRadioGroup для объединения в одну группу переключателей выбора цвета. Можно установить только один переключатель в группе, т.е., выбрать только один цвет.

TGroupBox для выделения флажков, управляющих параметрами шрифта. Флажки могут устанавливаться независимо друг от друга.

В контейнере ***TRadioGroup*** будут размещены переключатели типа ***TradioButton,*** посредством которых будет сделан выбор цвета. Можно поступить проще: воспользоваться сложным свойством **Items** компонента ***TRadioGroup*** , определив в нем перечень возможных цветов (наименований). Однако в этом случае нельзя будет окрасить каждый вариант выбора в свой цвет.

Все прототипы компонентов, кроме ***TBitBtn***, находятся в палитре **Standard.** Прототип компонента ***TBitBtn*** – в странице **Additional.**

Как уже отмечалось, при создании каждого экземпляра компонента ***Lazarus*** автоматически присваивает ему имя. Эти имена сохранены в нижеследующем примере выполнения работы. Эти имена, естественно, можно произвольно изменить, придав им какой-то смысл. При этом везде (во всех процедурах) вместо автоматически формируемого имени следует использовать имя, заданное в свойстве **Name.**

Пример выполнения работы

1. Создать на диске папку для лабораторной работой **№ 1**, присвоив этой папке имя **Lab1**

2. Войти в среду ***Lazarus*.** Будет автоматически создан новый проект с единственной (пустой) формой. Этот проект уже готов к выполнению. Запустить его, щелкнув кнопку **Запуск⇒Запустить (F9)**. Легко убедиться, что появившееся на экране окно-форма обладает всеми свойствами окна ***Windows***: его можно перемещать по экрану, изменять размеры, сворачивать в значок и разворачивать на весь экран. Окно имеет стандартные интерфейсные элементы: заголовок, оконное меню, кнопки управления размером и кнопку закрытия. Однако на этом возможности окна и исчерпываются. Это естественно, ведь в проекте еще ничего, кроме формы не создано.

3. Изменить значение свойства **Name** (имя формы) **Form1** в соответствии с изложенными выше требованиями. Например**,** ***L1_Ivanov_Form1***. В свойство **Caption** занести текст, который будет

показываться в заголовке формы при выполнении приложения. Например, ***“Лаб 1. Выполнил Иванов И.И., группа ЭСА11-1”.***

4. Свойству **BorderStyle** формы (стиль рамки окна) присвоить значение ***bsDialog***. Это значение определяет окно как диалоговое, его размеры на этапе прогона приложения (в процессе работы приложения) не могут быть изменены.

5. Поместить на форму компонент **Button1** (командная кнопка) и написать на ней ***“Вывод текста”*** (т.е. занести этот текст в свойство **Caption**).

6. Аналогично создать кнопку **Button2** и написать на ней ***“Запуск формы”.***

7. Поместить на форму компонент **BitBtn1** из палитры **Additional**. Для свойства **Kind** выбрать из предлагаемого перечня значение ***bkClose***. При этом автоматически на кнопку помещается надпись ***“Закрыть”*** и добавляется значок «косой крестик»**.**

8. Поместить на форму компонент **Memo1** (многострочный редактор). Двойным щелчком на свойстве **Lines** вызвать окно ***«Диалог ввода строк»*** (**String List Editor)** и удалить из него все строки.

9. Поместить на форму компонент **GroupBox** и в свойстве **Caption** написать “**Стиль шрифта**”. Это компонент-контейнер.

10. Разместить в контейнере **GroupBox** четыре компонента **CheckBox1**, …, **CheckBox4**, подписав их как “**Полужирный**”, ”**Курсивный**”, ”**Подчеркнутый**” и “**Перечеркнутый**”.

11. Поместить на форму компонент **Edit1** и очистить его свойство **Text.**

12. Слева от компонента **Edit1** поместить компонент **Label1** с подписью (свойство **Caption**) “***Введите текст***”.

13. Поместить на форму компонент **RadioGroup** и в свойстве **Caption** этого компонента написать “**Цвет шрифта**”. Это компонент-контейнер.

14. Разместить в контейнере **RadioGroup** пять компонентов-радиокнопок **RadioButton1**, …, **RadioButton5**, подписав их как “**Красный**”, ”**Синий**”, ”**Зеленый**”, “**Желтый**” и “**Черный**”. В соответствии с этими подписями установить значение свойства **Color** (цвет фона) каждого из этих компонентов: ***clRed, clBlue, clGreen, clYellow, clBlack.*** Чтобы на этом фоне смотрелись надписи, для компонентов синего, зеленого и черного цвета изменить цвет символов (подсвойство **Color** сложного свойства **Font**) на белый (***clWhite***). Особенностью контейнера **RadioGroup** является то, что только один из размещенных на нем компонентов **RadioButton** может быть включенным. При включении любого из них все остальные автоматически выключаются.

15. Используя команду **Файл⇒Создать форму** создать новую форму **Form2,** определив ее имя (свойство **Name**) в соответствии с требованиями, изложенными ранее (в разделе 1). В свойство **Caption** этого компонента занести текст “***О программе***”. Цветовое оформление, стиль и т.п. выбрать по своему усмотрению. С помощью компонента **Shape** из панели ***Additional*** можно создать различные элементы оформления. Например, на рис. 1. с

помощью этого компонента создан эллипс (свойство **Shape** установлен в ***stEllipse***).

16. Поместить на новую форму компонент **Button1** (кнопка). На кнопке написать «**Скрыть форму**».

17. Поместить на форму **Form2** компонент **Image1** (отображение картинок) из палитры **Additional** и загрузить в него соответствующий тематике рисунок (можно загрузить файл ***lab1.jpg***, прилагающийся к электронному варианту рабочего задания). Загрузка рисунка производится при двойном щелчке на свойстве **Picture**. Можно также щелкнуть на кнопке с многоточием. В обоих случаях откроется окно **Диалог загрузки изображения (Picture Editor)**, с помощью которого и загружается изображение. Если изображение не помещается в отведенную область компонента **Image1**, то свойству **Stretch** необходимо присвоить значение **true**, т.е. «подгонка» рисунка под отведенную область.

18. В результате проведенных операций должно получиться две формы примерно такого вида, как приведено на рис.6 (показан вид, которые формы будут иметь на этапе выполнения).

Формы не должны обязательно иметь представленный выше вид, но общая идея лабораторной работы, т.е. знакомство с основными компонентами и их свойствами должна быть соблюдена.

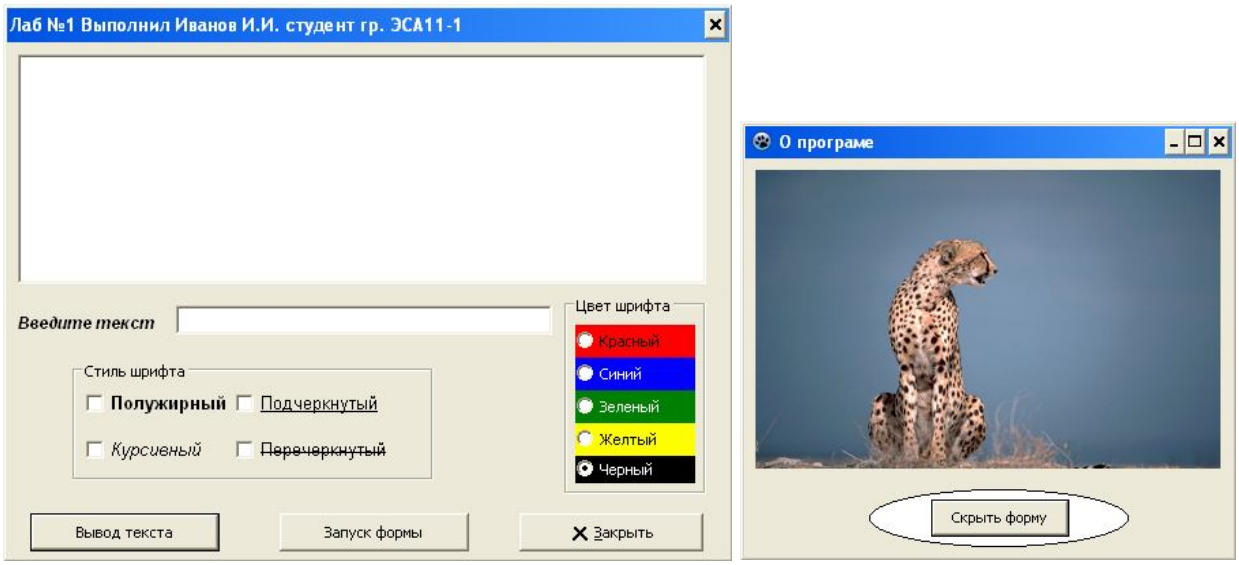

Рисунок 6 – Вид форм для лабораторной работы

19. Запустить приложение на выполнение. Несмотря на наличие в форме всевозможных элементов управления, они не выполняют никаких действий. Например, можно выбрать любой стиль шрифта или цвет, но этот выбор нигде не отразится. Это естественно: в проекте еще не разработан ни один обработчик события, поэтому ни одно событие не обрабатывается, т.е. приложение не реагирует еще ни на одно событие. Форма **Form2** вообще никогда не сможет появиться на экране, несмотря на то, что она зарегистрирована в проекте. Наличие командной кнопки ***“Запуск формы”*** еще не обеспечивает ее показ на экране. Необходим соответствующий

программный код, который бы в ответ на щелчок на этой кнопке показывал форму. Работает только один единственный компонент – командная кнопка ***"Закрыть"***, так как в этом компоненте автоматически генерируется соответствующий код для завершения выполнения приложения. С помощью этой кнопки закрыть приложение.

20. Сделать первую форму активной**.** Щелчком на компоненте **Button2** (командная кнопка ***"Запуск формы"***) выбрать (активизировать) его. Кстати, выбрать компонент можно и из списка компонентов в верхней части ***Инспектора объектов***. В окне ***Инспектора объектов*** (**Object Inspector)** отображаются свойства и события выбранного компонента, т.е. при активизации **Button2** будут показаны свойства и события именно этого объекта. В окне **Object Inspector** перейти на вкладку **События** (**Events**)**.** Найти строку с именем события **OnClick** (щелчок)**.** Двойной щелчок в правой колонке этой строки переводит в окно ввода кода обработчика этого события. Кстати, для объекта типа **TButton** событие **OnClick** является событием по умолчанию, код которого раскрывается двойным щелчком на самом объекте. Если обработчик события еще не разработан, **Lazarus** создает заготовку для его создания. Создать следующую процедуру обработки этого события (эта процедура показывает форму Form2, делая ее видимой путем установки в ***true*** свойства **Vsible**):

```
procedure TL1_Ivanov_Form1.Button2Click (Sender: TObject);
begin
   L1_Ivanov_Form2.Visible:=True;
end;
```

В этой процедуре делается ссылка на форму **Form2**. В модуле формы **Form1** такой компонент неизвестен. Поэтому в таком виде проект не откомпилируется. Необходимо в модуль формы **Form1** добавить ссылку на модуль **Unit2**. Однако перед этим модулю должно быть присвоено имя в соответствии с разделом 1: **L1_Ivanov_Unit2**. Для этого модуль необходимо сохранить в папке размещения проекта, указав при сохранении требуемое имя. Указанное имя будет занесено и в первую строку модуля. Сама ссылка на модуль **Unit2** задается путем добавления в объектном коде первой формы в разделе реализации модуля **implementation** перед директивой {$R *.DFM} команды присоединения второго модуля:

uses L1_Ivanov_Unit2;

21. Сделать активной вторую форму (**Form2**). В окне **Инспектора объектов** перейти на вкладку **События.** Из списка компонентов, установленных на форме выбрать **Button1**. Найти событие **OnClick.** Двойным щелчком по этому событию перейти в окно ввода кода. Создать процедуру обработки этого события, закрывающую (делающую невидимой) форму **Form2**:

```
procedure TL1_Ivanov_Form2.Button1Click (Sender: TObject);
```

```
begin
    L1_Ivanov_Form2.Visible:=False;
end;
```

22. Запустить приложение на выполнение. Теперь при щелчке на командной кнопке **Button2** (***"Запуск формы"***) на экране появится форма **Form2,** т.к. событие Щелчок на этой кнопке будет обработано и этот обработчик покажет форму (сделает ее видимой). Аналогично щелчок на соответствующей кнопке формы **Form2** уберет с экрана эту форму (сделает ее невидимой). Закрыть приложение.

23. Активизировать форму **Form1** непосредственно. В окне ***Инспектора объектов*** (**Object Inspector)** найти событие **OnActivate,** наступающее в момент активации формы**.** Двойным щелчком на этом событии перейти в окно кода, где создать процедуру, которая при запуске проекта на выполнение будет выделять **RadioButton5** (т.е. «включать» черный цвет):

```
procedure TL1_Ivanov_Form1.FormActivate (Sender: TObject);
begin
    RadioButton5.Checked:=True;
end;
```

24. Сделать окно первой формы активным**.** Щелчком на компоненте **Button1** (командная кнопка ***"Вывод текста"***) выбрать (активизировать) его. В окне **Инспектора объектов** перейти на вкладку **События.** Найти строку с именем события **OnClick** (Щелчок)**.** Двойной щелчок в правой колонке этой строки (события) переводит в окно ввода кода обработчика этого события. Создать следующую процедуру обработки этого события:

```
procedure TL1_Ivanov_Form1.Button1Click(Sender: TObject);
    var i_str,str:string;i_int:integer;k:TColor;
begin
    // Очищаем многострочный редактор.
    Memo1.Lines.Clear;
    // Определяем, какая радиокнопка нажата, и в зависимости
    // от этого выбираем цвет для отображения текста в
    // редакторе. Выбранный цвет присваиваем свойству
    // Font.Color объекта Memo1.
    if RadioButton1.Checked then k:=clRed;
    if RadioButton2.Checked then k:=clBlue;
    if RadioButton3.Checked then k:=clGreen;
    if RadioButton4.Checked then k:=clYellow;
    if RadioButton5.Checked then k:=clBlack;
    Memo1.Font.Color:=k;
    // Формируем текст для вывода в многострочном редакторе.
    // Текст компонуется из двух частей: постоянный текст
    // (константа) и текст, введенный в поле однострочного
    // редактора Edit1.
    str:='Привет! Моя первая программа '+Edit1.Text;
```

```
    // Запрашиваем размер шрифта и присваиваем его свойству
    // Font.Size объекта Memo1.
    i_str:= InputBox ('Блок ввода', 'Введите размер
    шрифта','10');
    i_int:=StrToInt(i_str);
    Memo1.Font.Size:=i_int;
    // Формируем стиль шрифта в зависимости от установок
    // соответствующих переключателей на форме.
    Memo1.Font.Style:=[];
    if CheckBox1.State = cbChecked then
          Memo1.Font.Style:=[fsBold];
    if CheckBox2.State = cbChecked then
          Memo1.Font.Style:=Memo1.Font.Style+[fsItalic];
    if CheckBox3.State = cbChecked then
          Memo1.Font.Style:=Memo1.Font.Style+[fsUnderline];
    if CheckBox4.State = cbChecked then
          Memo1.Font.Style:=Memo1.Font.Style+[fsStrikeOut];
    // Помещаем скомпонованную строку в многострочный
    // текстовый редактор Memo1.
    Memo1.Lines.Add(str);
end;
```

25. Нажав клавишу **F9**, запустить приложение на выполнение. Объяснить логику функционирования приложения. В случае необходимости выполнить отладку.

26. Сохранить форму и сам проект в папке размещения проекта, присвоив им имена, как указано в разделе 1.

27. По результатам работы составить отчет.

Лабораторная работа № 2

Условные операторы. Вычисление значения функции, заданной условно.

Цель: Создание простого законченного приложения. Знакомство с возможностями ***Lazarus***. Изучение основных операторов ***Pascal*** Проектирование разветвляющегося вычислительного процесса.

Рабочее задание: Создать приложение для вычисления и вывода на экран значения функции:

$$y=\begin{cases} f_1(x), \text{ если} & x \le 0 \\ f_2(x), \text{ если} & 0 < x \le a \\ f_3(x), \text{ если} & x > a \end{cases}$$

Выражения для функции $f_1(x)$, $f_2(x)$ и $f_3(x)$ выбрать из таблицы 1 в соответствии с номером своего варианта. В форме предусмотреть поля для ввода значения параметра **a** и переменной **x**, вывода результата вычисления **y**, а также командные кнопки для осуществления расчета и выхода из приложения. Вводимые данные должны подвергаться контролю на допустимость ввода.

Таблица 1 – Функции

вариант	**$f_1(x)$**	**$f_2(x)$**	**$f_3(x)$**
1	$\mathrm{tg}(2ax)$	$\sin(3x+\sqrt[3,5]{\lvert a\rvert})$	$\cos(x-2a)$
2	$5ax+2$	$5/\left(x+0,5\lvert a\rvert^{1,5}\right)$	$0,5/\sin(ax+1)$
3	$\sqrt[3]{\lvert x-1\rvert}-a$	$x^4/7+a$	$\sin^3(2ax+\pi)$
4	$\sqrt[3]{\sin^2(ax)+\cos^4(ax)}$	$\mathrm{ctg}(ax+0,4)$	$\ln(2x+0,5a)$
5	$ax^3-\ln\lvert x-1\rvert$	$\ln^3(ax+4)$	x^4-x^{2-a}
6	$\sin(x^2+a^3)$	$e^{-x}+1/ax$	$\ln(ax^3+x^2)$
7	$(3ax-1)/x^5$	$\ln^2\lvert\sqrt{x+5a}\rvert$	$\sqrt{1+(ax)^{2,5}}$
8	$a^2\mathrm{ch}(x)+x$	$1/(\mathrm{arctg}(2x)+a)$	ax^2e^{-x}
9	$\lvert ax-1\rvert^{1,2}\sin(3x)$	$\mathrm{sh}^2(x)+ax$	$\arcsin(ax)+x^{0,25}$
10	$\mathrm{th}(x)+x^{\sin(ax)}$	$\sqrt{\lvert\arcsin^2(x)+\arccos(a$	$\ln^2(x^2)+\sqrt{x^{1,25}}$
11	$\sin^3\lvert x+a\rvert_{3^{\cdot}32}+a\rvert$	$\sin(a/(2x^2+1))$	$2\sin(ax-e^{-x})$
12	$e^{\lvert x+a\rvert^x}$	$\cos(\lvert ax\rvert^{\sin(ax)})$	$\lvert x\rvert^{\sqrt{\lvert x\rvert}}+\lvert a\rvert^{\sqrt[4]{\lvert a\rvert}}$
13	$\log_5\lvert x^2+\sin(a)\rvert$	$(a+\sin(e^{-x}))\mathrm{th}(x)$	$(1/a)\mathrm{tg}(x)$
14	$\log_3\lvert x^2+a^2\rvert$	$(x+a)\sqrt[5]{\lvert\sin(ax)+\cos(a$	$\sin(ax)+1$

вариант	$f_1(x)$	$f_2(x)$	$f_3(x)$
15	$ax+1/(ax+2)$	$\left(\sqrt{\lvert x^3+a\rvert}\right)\cos(x+1)$	$x^2+\sin(5ax)$
16	$x^2+100\cdot\sin(ax^3)$	$\ln(4ax+1)^2$	$\ln\sqrt[5]{x+x^2}$
17	$\lvert x-1\rvert^{4,1}+a\lvert x-1\rvert^{2,4}+\lvert a\rvert$	$\sin(2x)+\cos^2(x+a)$	$\lvert x\rvert^{0,5x+1}+\lvert a\rvert^{1,2}$
18	$0,5a+\lvert x\rvert^5\,\mathrm{ctg}\,2\lvert x\rvert^3$	$\ln(x+1)\Big/\left(\lvert x+a\rvert^{1,3}\right)$	$e^{-2x}-\sqrt[3]{x}+0,1a$
19	$(a+5)\sin(4x)$	$\sqrt[5]{6x-x^2+a^2}$	$\lvert x\rvert^{0,1ax}+\lvert a\rvert^{0,1ax}$
20	$\mathrm{ctg}(2ax-1)^2$	$0,1a+xe^{-x}$	$\sin^3\left(x^2+1\right)+\mathrm{th}(a)$
21	$x^3+x^2+\lvert x-1\rvert^{0,1ax}$	$(x-1)^3+\cos\left(2x^3+a\right)$	$2x+\sin(a)+\mathrm{th}(0,01x)$
22	$\mathrm{tg}^{0,4}(0,01\pi x^2)+ax$	$e^{x+1}-a\cdot\sin(x+\pi)$	$3\ln\sqrt[5]{\sin x+x^2}+0,01a$
23	$3x^5-\mathrm{ctg}\left(\pi x^3\right)+\lvert a\rvert^{x+\pi}$	$\ln(\sin 4x+\lvert a\rvert^{0,3})^2$	$ax-x^2$
24	$\lvert a\rvert^{\sin(x)+1}x+x\cdot\sin(x)$	$3^{x+3}+2^a$	$\sin(\pi a/x)$
25	$x^2+\arcsin(ax)-1$	$\lg\left\lvert x^a+a^x+10^{ax}\right\rvert$	$e^{-2x}+\sqrt[7]{\left(x^2+a^4\right)}$
26	$\lvert x-1\rvert^{\pi}+\lvert a\rvert^{\pi}+\pi^{0,01x}$	$\sin(ax+\pi)-\cos^2(x)$	ax^2-5x

Пример выполнения работы

Создать приложение для вычисления значения функции:

$$y=\begin{cases}\sin(x^2+ax), & \text{если}\quad x\le 0\\ 1-\dfrac{1+\sqrt{(x^2+ax)}}{e^{\sin(x)}(1+x)}, & \text{если}\quad 0<x\le a\\ \dfrac{\cos(x^2-a^2)}{\sqrt{1-\sin(a-x)}}-\dfrac{1-\sin(a-x)}{e^{\sin(x)}}, & \text{если}\quad x>a\end{cases}.$$

1. Создать на диске папку для лабораторной работой **№ 2**, присвоив этой папке имя **Lab2.**

2. Войти в среду ***Lazarus***. Создать новый проект. Присвоить ему имя в соответствии с требованиями, изложенными в разделе 1.

3. Используя палитру компонентов (элементов управления), спроектировать форму **Form1,** изменить значение свойства **Name** в соответствии с требованиями, изложенными в разделе 1 (L2_<ФИО>_Forma1). Для свойства **BorderStyle** лучше установить ***bsSingle*.** Это значение запрещает изменять с помощью мыши размеры формы в процессе выполнения приложения. Для **BorderIcons ⇒ biSystemMenu** установить ***True***, а для остальных трех свойств из раздела **BorderIcons** установить ***False***. Такие установки оставляют в заголовке окна приложения только кнопку закрытия, убирая остальные, не позволяя тем самым изменять размеры формы с помощью кнопок в заголовке окна. Таким образом, во время работы приложения размеры формы изменить нельзя никаким способом. Примерный вид формы показан на рисунке 7.

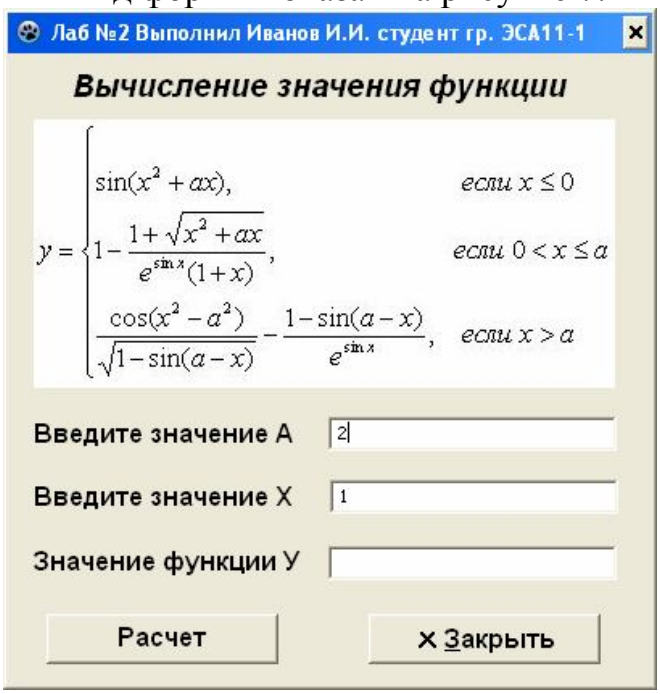

Рисунок 7 – Примерный вид проектируемой формы

4. Поместить на форму компонент **Label,** присвоив ему имя (свойство **Name**) **Label0.** В свойство **Caption** занести текст «**Вычисление значения функции**».

5. Поместить на форму компонент **Image1** из палитры **Additional** и загрузить в него заранее подготовленный рисунок, отображающий функцию. Загрузка рисунка производится с помощью стандартного окна диалога, открываемого при щелчке на кнопке с многоточием в свойстве **Picture.** Подготовить такой рисунок можно, используя редактор формул **Microsoft Equation** из состава **Microsoft Word** и любой графический редактор (например, **Photoshop** или **Paint).** Вначале набирается формула в редакторе формул, а затем при помощи буфера обмена она помещается в графический

редактор, после чего файл рисунка необходимо сохранить в папку, отведенную для текущего проекта.

6. Поместить на форму три компонента типа ***TEdit***: **Edit1, Edit2, Edit3.** Первые два предназначены для ввода соответственно значения параметра **A**, и значения переменной **X**. В свойства **Text** этих компонентов ввести какие-либо значения – значения по умолчанию. Эти значения будут показываться при запуске приложения на выполнение. При выполнении приложения их можно будет заменить другими. Компонент **Edit3** будет использован для вывода результата вычисления функции. Поэтому необходимо запретить ввод данных в него пользователем. Для этого свойству **ReadOnly (только чтение)** присвоить значение ***true***, запрещающее пользователю заносить в компонент какие-либо данные. Можно поступить другим способом: свойству **Enabled** этого компонента присвоить значение ***false***, что сделает компонент недоступным для пользователя. При этом, правда, значение будет показываться более бледным цветом.

7. Поместить на форму три компонента типа ***TLabel***: **Label1, Label2, Label3** для подписей к полям ввода. В поле свойства **Caption** написать "**Введите значение A"**, "**Введите значение X"** и **"Значение функции Y"** соответственно. При этом свойство **Color** лучше установить таким же, как и у формы. Параметры шрифта (сложное свойство **Font**) установить по своему усмотрению.

8. Включить в форму две командные кнопки: элемент управления типа ***TButton*** с именем **Button1** (свойство **Name**), и кнопку типа ***TBitBtn*** из палитры **Additional** с именем **BitBtn1**. Написать (в свойстве **Caption)** на первой кнопке «**Расчет**». Для свойства **Kind** второй кнопки в окне **Инспектор объектов** установить значение **bkClose.** Подобрать удобный для просмотра размер шрифта (свойство **Font⇒Size).**

9. Щелчком на кнопке «**Переключить форму/модуль» (F12)** на панели инструментов главного окна интегрированной среды перейти к окну для набора программного кода модуля **Unit1.** В разделе реализации модуля **implementation** сразу после директивы {$R *.DFM} задать свою функцию, например:

```
function f(x,a:real):real;
begin
   if x <= 0 then
      f:= sin(x*x + a*x)
   else if x<=a then
      f:=1-(1 + sqrt(x*x+a*x))/(exp(sin(x))*(1+x))
   else
      f:=cos(x*x-a*a)/sqrt(1-sin(a-x))-
         (1-sin(a-x))/exp(sin(x));
end;
```

10. Поскольку проект рассчитан на многократный расчет значения функции, то при вводе новых значений параметра X (или A) результат

предыдущего расчета (У) уже не будет соответствовать исходным данным. Поэтому необходимо скрывать «старый» ответ. Для этого необходимо с событиями **OnEnter** (***Получение фокуса***) для однострочных окон редактирования **Edit1** и **Edit2** связать программные коды, которые делают невидимым однострочное текстовое поле (**Edit3**):

```
procedure TL2_Ivanov_Form1.Edit1Enter(Sender: TObject);
begin
   Edit3.Visible:=False;
end;

procedure TL2_Ivanov_Form1.Edit2Enter(Sender: TObject);
begin
   Edit3.Visible:=False;
end;
```

11.Также необходимо проверять корректность входных данных, т.е. их соответствие числовому представлению исходных значений. Для этого добавляем код, обрабатывающий события **OnExit (*Потеря фокуса)*** для однострочных окон редактирования **Edit1** и **Edit2**:

```
procedure TL2_Ivanov_Form1.Edit1Exit(Sender: TObject);
begin
   if Edit1.Text<>'' then
   try
      StrToFloat(Edit1.Text);
   except
      on EConvertError do
         begin
           ShowMessage('Введено неверное значение A');
           Edit1.SetFocus;
         end;
   end;
end;

procedure TL2_Ivanov_Form1.Edit2Exit(Sender: TObject);
begin
   if Edit2.Text<>'' then
   try
      StrToFloat(Edit2.Text);
   except
      on EConvertError do
         begin
           ShowMessage('Введено неверное значение X');
           Edit2.SetFocus;
         end;
   end;
end;
```

12.С событием **OnClick** кнопки **Button1** связать программный код, который вычисляет значение функции и выводит его в поле однострочного редактора **Edit3**:

```
procedure TL2_Ivanov_Form1.Button1Click(Sender: TObject);
begin
   Edit3.Visible:=True;
   Edit3.Text:=FloatToStr(f(StrToFloat(Edit2.Text),
               StrToFloat(Edit1.Text)));
end;
```

13.Нажав клавишу **F9**, запустить приложение на выполнение. Объяснить логику функционирования приложения. В случае необходимости выполнить отладку.

14.Сохранить отлаженный проект в папке проекта.

15.По результатам работы составить отчет.

Математические функции

Функция	**Описание**	**Аргумент**
Abs(X)	абсолютное значение	целое или действительное выражение
Ceil(X)	округление до наименьшего целого, превышающего или равного аргументу	действительное выражение
Compare Value (A, B)	сравнение двух значений; возвращает —1, 0 или +1, если A соответственно меньше, равно или больше B	целые или действительные выражения
DivMod (Dividend, Divisor, Result, Remainder)	целочисленное деление: **Result** — результат, **Remainder** — остаток	целые выражения
EnsureRange (AValue, AMin, AMax)	возвращает ближайшее к **AValue** в диапазоне **AMin — AMax**	целые или действительные выражения
Exp(X)	экспонента	действительное выражение
Floor(X)	округление до наибольшего целого, меньшего или равного аргументу	действительное выражение
Frac(X)	дробная часть аргумента: ***X-Int(X)***	действительное выражение
Frexp(X, Mantissa,	выделяет мантиссу и показатель степени 2:	действительное выражение

Функция	Описание	Аргумент
Exponent)	$X = Mantissa * 2^{Exponent}$	
InRange (AValue, AMin, AMax)	определяет, лежит ли **AValue** в диапазоне **AMin** – **AMax**	целые или действительные выражения
Int(X)	целая часть аргумента	действительное выражение
IntPower (X, E)	возведение **X** в целую степень **E**: X^E	действительное и целое выражения
IsInfinite(X)	определяет, не равен ли аргумент бесконечности	действительное выражение
IsNan(X)	определяет, не равен ли аргумент **NaN** - нечисловой величине	действительное выражение
IsZero(X, Epsilon)	определяет, не отличается ли аргумент от нуля менее чем на **Epsilon**	целые или действительные выражения
Ldexp(X,P)	умножение **X** на 2 в целой степени **P**: $X*2^P$	действительное и целое выражения
Ln(X)	натуральный логарифм от **X**	действительное выражение
LnXPl(X)	натуральный логарифм от **X+1**	действительное выражение
Logl0(X)	десятичный логарифм от **X**	действительное выражение
Log2(X)	логарифм от **X** по основанию **2**	действительное выражение
LogN (N, X)	логарифм от **X** по основанию **N**	действительное выражение
Max(A,B)	максимум двух чисел	действительное или целое выражение
Min(A,B)	минимум двух чисел	действительное или целое выражение
Pi	число Пи: **3.1415926535897932385**	–
Poly(X, C)	вычисляет полином **X** с массивом коэффициентов **C**	действительные выражение и массив
Power(X, E)	возведение **X** в произвольную степень **E**: X^E	действительное выражение
Round(X)	ближайшее целое аргумента	действительное выражение
RoundTo (AValue,	округляет действительное число до заданного десятичного порядка	действительные и целые выражения

Функция	Описание	Аргумент
ADigit)		
SameValue (A, B, Epsilon)	сравнивает **A** и **B** с точностью до **Epsilon**	действительные выражения
Sign(X)	определяет знак аргумента	действительные и целые выражения
SimpleRoundTo (A Value, ADigit)	округляет действительное число до заданного десятичного порядка	действительные и целые выражения
Sqr(X)	квадрат аргумента: **X** * **X**	действительное выражение
Sqrt(X)	квадратный корень	действительное выражение
Trunc(X)	возвращает целую часть действительного выражения	действительное выражение

Комментарий:

Математические функции описаны в модуле *Math.* Этот модуль должен быть подключен к приложению оператором **uses.**

Функции **Ln, LnXPl, Logl0, Log2, LogN** вычисляют логарифмы по различным основаниям. Если аргумент отрицательный, генерируется исключение **EInvalidOp.**

Функция **Sqrt** при извлечении корня из отрицательного числа возвращает значение **NaN**.

Тригонометрические и гиперболические функции

Функция	Описание	Модуль
ArcCos(X)	арккосинус	*Math*
ArcCosh(X)	арккосинус гиперболический	*Math*
ArcCot(X)	арккотангенс	*Math*
ArcCotH(X)	арккотангенс гиперболический	*Math*
ArcCsc(X)	арккосеканс	*Math*
ArcCsc(X)	арккосеканс гиперболический	*Math*
ArcSec(X)	арксеканс	*Math*
ArcSecH(X)	арксеканс гиперболический	*Math*
ArcSin(X)	арксинус	*Math*

Функция	Описание	Модуль
ArcSinh(X)	арксинус гиперболический	*Math*
ArcTan(X)	арктангенс	*System*
ArcTan2(Y, X)	арктангенс от **Y / X**	*Math*
ArcTanh(X)	арктангенс гиперболический	*Math*
Cos(X)	косинус	*System, Math*
Cosecant(X)	косеканс	*Math*
Cosh(X)	косинус гиперболический	*Math*
Cot(X)	котангенс	*Math*
Cotan(X)	котангенс	*Math*
CotH(X)	котангенс гиперболический	*Math*
Csc(X)	косеканс	*Math*
CscH(X)	косеканс гиперболический	*Math*
Hypot(X, Y)	вычисление гипотенузы по заданным катетам **X** и **Y**	*Math*
Sec(X)	секанс	*Math*
Secant(X)	секанс	*Math*
SecH(X)	секанс гиперболический	*Math*
Sin(X)	синус	*System, Math*
SinCos(X, S, C)	синус и косинус	*Math*
Sinh(X)	синус гиперболический	*Math*
Tan(X)	тангенс	*Math*
Tanh(X)	тангенс гиперболический	*Math*

Комментарии:

Во всех функциях тип аргумента и тип возвращаемого значения - **Extended**.

Во всех тригонометрических функциях аргумент **X** — угол в радианах. Обратные тригонометрические функции возвращают главное значение в радианах. В функциях ArcSin и ArcCos аргумент должен лежать в пределах от -1 до 1. Функции ArcSin и ArcTan возвращают результат в пределах [-Pi/2 .. Pi/2], ArcCos — в пределах [0 .. Pi].

Лабораторная работа № 3.

Циклический алгоритм. Табулирование функции и поиск экстремумов.

Цель: Изучение операторов цикла. Создание циклических алгоритмов. Приобретение навыков создания проекта, работы с несколькими формами.

Рабочее задание: Создать приложение для вывода на экран в отдельную форму таблицы значений функции, заданной в лабораторной работе № 2. Диапазон и шаг изменения аргумента задавать при выполнении приложения. Определить максимальное и минимальное значения в заданном диапазоне методом перебора. При создании проекта использовать проект, созданный в лабораторной работе №2, не уничтожая последнего.

Краткие теоретические сведения

Алгоритм циклической структуры – это вычислительный процесс, содержащий многократные вычисления по одним и тем же математическим зависимостям, но для различных значений входящих в него переменных. Переменные, изменяющиеся в цикле, называются параметрами цикла.

Каждый алгоритм циклической структуры содержит такие элементы:

а) ***подготовка цикла*** – определение начальных значений параметров цикла;

б) ***тело цикла*** – действия, что повторяются многократно для разных значений параметров цикла;

в) ***модификация цикла*** – смена значений параметров цикла;

г) ***управление циклом*** – проверка условия выхода из цикла.

Табулирование функции – это формирование и вывод на экран или принтер таблицы значений функции для значений аргумента (**x**), изменяющихся от некоторого начального значения ($\mathbf{x_n}$) до конечного ($\mathbf{x_к}$) с некоторым шагом (**h**). Для этого используется цикл. При подготовке цикла аргументу присваивается начальное значение ($\mathbf{x=x_n}$), в теле цикла вычисляются и выводятся значения функции для текущего значения аргумента **x**. Модификация заключается в увеличении аргумента на величину **h** (**x:=x+h**). Цикл завершается, когда после очередного изменения значение аргумента превысит конечное значение ($\mathbf{x > x_к}$).

Поиск экстремумов функции на заданном отрезке методом перебора выполняется в цикле, аналогично табулированию. Вместо вывода значение функции в каждой точке сравнивается с наибольшим (наименьшим) из значений во всех предыдущих точках. Если текущее значение больше (меньше) наибольшего (наименьшего) из предыдущих, то его надо считать новым наибольшим (наименьшим) значением. В противном случае наибольшим (наименьшим) значением остается значение функции определенное в предыдущих точках.

Пример выполнения работы

1. На диске создать папку для проекта лабораторной работы №3. Присвоить ей имя **Lab3.**

2. Создаваемый проект рассчитывает таблицу значений функции, которая была задана в лабораторной работе № 2. Чтобы заново не создавать на форме необходимые компоненты – откроем предыдущий проект, выделим все размещенные на нем компоненты и скопируем их (например, Ctrl+C). После этого закроем проект второй лабораторной работы и создадим новый (**Проект⇒Создать проект**), на форму которого поместим скопированные компоненты.

3. Изменить имя формы применительно к лабораторной работе № 3. Например, **L3_Ivanov_Form1**. Создать соответствующую подпись в строке заголовка формы (свойство **Caption**).

4. В свойство **Caption** компонента **Label0** занести новый текст, соответствующий новой работе "**Табулирование функции**".

5. Внести изменения в исходную форму и привести ее к виду, показанному на рис.8. Для этого поместить на форму (добавить) однострочное окно редактирования с именем **Edit4** и метку **Label4**. Надписи в метках изменить в соответствии с рис. 8. Свойству **ReadOnly** однострочного окна редактирования **Edit3** вернуть значение **true**. Очистить свойства **Text** всех однострочных окон редактирования, а лучше ввести в них числовые значения – значения по умолчанию.

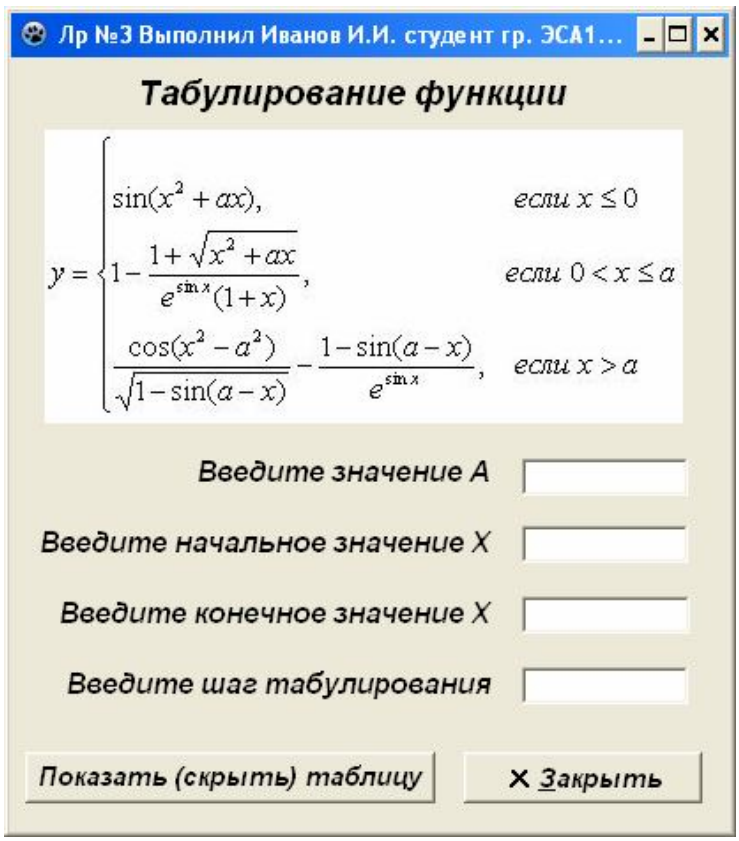

Рисунок 8 – Примерный вид проектируемой формы

6. В Элементе управления **Button1** изменить значение свойства **Caption** на “**Показать (скрыть) таблицу”**. Подобрать удобный для просмотра размер шрифта (**Font⇒Size).**

7. Компонент **Image** из палитры **Additional** с загруженным в него рисунком, отображающим исследуемую функцию, не меняется.

8. Создать новую форму **Файл⇒Создавать форму.** Дать ей имя в соответствии с требованиями, изложенными в разделе 1. В поле свойства **Caption** написать “**Таблица значений”**. Значение свойства **BorderStyle** принять **bsSingle.**

9. Взаимно связать обе формы, поместив в их модули ссылки друг на друга так, как это делалось в лабораторной работе № 1.

10.Поместить на новую форму компонент **StringGrid1** из палитры **Additional**. Этот компонент предназначен для создания текстовых таблиц. Будем выводить в него формируемую таблицу значений функции. Свойству **ColCount (**количество колонок) присвоить значение 2. Количество строк **RowCount** задать 2. Это, впрочем, не имеет никакого значения, т.к. реальное количество строк в таблице будет определено и уточнено в процессе выполнения программы в зависимости от количества точек в таблице. Ширина столбцов по умолчанию **DefaultColWidth** равна 64. Высота строки по умолчанию **DefaultRowHeight** равна 24. Свойство **ScrollBars** установить равным **ssVertical** (только вертикальная полоса прокрутки). Свойству **FixedCols** задать значение 0 (снять выделение начального столбца).

11.Изменить размеры второй формы таким образом, чтобы большую часть ее площади занимала полученная таблица (рисунок 9).

Рисунок 9 – Примерный вид формы для вывода таблицы значений.

12. На вторую форму поместить компоненты **Label1** и **Label2**, в свойство **Caption** которых занести: “**Максимум”** и “**Минимум”** соответственно.

13. На вторую форму поместить также однострочные окна редактирования **Edit1** и **Edit2**. Они будут использованы только для вывода максимального и минимального значений функции, вычисленных в процессе работы приложения. Ввод данных пользователем в эти окна не допускается. Поэтому свойству **ReadOnly** каждого компонента присвоить значение ***true*** (запрет ввода информации). Значение свойства **BorderStyle** принять **bsNone** для обоих компонентов. Цвет компонентов установить таким же, как и у формы, на которую выводится таблица значений (**Color⇒clForm**). Значение свойства **Text** можно очистить.

14. Отредактировать программный код для модуля первой формы (**Unit1**). Подпрограмма-функция **f(x)**, осуществляющая вычисление значения функции не изменяется по сравнению с предыдущей лабораторной работой. Однако чтобы эта функция была доступна в модуле второй формы, где она и будет использована, в интерфейсный раздел модуля первой формы (**interface**) необходимо добавить описание этой функции:

```
function f(x,a:real):real;
```

В обработчиках событий **OnExit** однострочных окон редактирования, перешедших из предыдущей лабораторной работы подкорректировать текст сообщения об ошибке в соответствии с новым назначением окон. Для нового однострочного окна редактирования **Edit4** и для окна редактирования **Edit3** создать обработчики событий **OnEnter** и **OnExit**, аналогичные соответствующим обработчикам для остальных окон редактирования. При этом в момент исполнения события **OnEnter** должна становиться невидимой вторая форма. Обработчик события **OnClick** для командной кнопки **Button1 (“скрыть/показать таблицу”)** переписать заново. В этом обработчике сначала делается проверка исходных данных. Если какой-то параметр не задан или конечное значение диапазона табулирования окажется меньше начального – об этом выводится сообщение и фокус передается на соответствующий компонент для ввода данных. Если все данные введены, значение свойства **Visible** второй формы изменяется на противоположное. Таким образом, при каждом щелчке на кнопке **Button1** состояние формы попеременно изменяется: она то показывается, то скрывается. Расчет таблицы значений выполняется непосредственно при показе второй формы (формы с таблицей).

```
procedure TL3_Ivanov_Form1.Button1Click(Sender: TObject);
begin
  If  (Edit1.Text='') Then
    Begin
      ShowMessage('Отсутствует значение A');
      Edit1.SetFocus;
```

```
    End
  Else If (Edit2.Text='') Then
    Begin
      ShowMessage('Отсутствует начальное значение X');
      Edit2.SetFocus;
    End
  Else If  (Edit3.Text='') Then
    Begin
      ShowMessage('Отсутствует конечное значение X');
      Edit3.SetFocus;
    End
  Else If  (Edit4.Text='') Then
    Begin
      ShowMessage('Отсутствует шаг H');
      Edit4.SetFocus;
    End
  Else if (StrToFloat(Edit3.Text))<=
          (StrToFloat(Edit2.Text)) Then
    Begin
       ShowMessage('Начальное значение X должно быть' +
                   ' меньше конечного!');
       Edit3.SetFocus;
    End
  Else
    L3_Ivanov_Form2.Visible:= not L3_Ivanov_Form2.Visible
end;
```

15.Создать программный код для второго модуля второй формы (**Unit2**). Этот код содержит единственную процедуру – процедуру обработки события **OnShow (показ формы),** в которой и осуществляется расчет таблицы значений. Организуется цикл для расчета каждого значения. Результаты выводятся в компонент **StringGrid1** (таблица строк). Количество строк в таблице формируется динамически: после формирования очередной строки общее количество строк (свойство **RowCount**) увеличивается на 1. Первоначально устанавливается количество строк, равное 2, т.е. на 1 больше, чем надо (первая строка – заголовок, составляет фиксированную зону). Это вызвано тем, что таблица должна иметь хотя бы одну информационную строку. Поэтому по завершению цикла таблица усекается на одну строку.

```
procedure TL3_Ivanov_Form2.FormShow(Sender: TObject);
   var  H,Xn,Xk,A,X,Y,Ymax,Ymin,Xmax,Xmin:Real;
begin
   // Формируем заголовок грида.
   StringGrid1.Cells[0,0]:='        X';
   StringGrid1.Cells[1,0]:='       f(X)';
   // Толщина разграничительных линий.
   // Можно было установить на этапе проектирования.
   StringGrid1.GridLineWidth:=2;
```

```
   // Количество строк устанавливаем 2, потом после
добавления
   // каждой строки будем наращивать по 1. В итоге получим
   // на одну строку больше.
   StringGrid1.RowCount:=2;
   // Переносим параметры из формы в рабочие переменные.
   A:=StrToFloat(L3_Ivanov_Form1.Edit1.Text);
   Xn:=StrToFloat(L3_Ivanov_Form1.Edit2.Text);
   Xk:=StrToFloat(L3_Ivanov_Form1.Edit3.Text);
   H:=StrToFloat(L3_Ivanov_Form1.Edit4.Text);
   // Присваиваем начальные значения для цикла.
   X:=Xn; Ymax:=-1e30;Ymin:=1e30;
   // Организуем цикл.
   // В цикле вычисляем значение функции в очередной точке,
   // Уточняем max и min.
   // Заносим данные в таблицу и увеличиваем счетчик строк.
   While X<=Xk Do
   Begin
      Y:=f(X,A);
      If Y>Ymax Then Begin Ymax:=Y;Xmax:=X; End;
      If Y<Ymin Then Begin Ymin:=Y;Xmin:=X; End;
      StringGrid1.Cells[0,StringGrid1.RowCount-1]:=
            FloatToStrF(X,ffGeneral,6,6);
      StringGrid1.Cells[1,StringGrid1.RowCount-1]:=
            FloatToStrF(Y,ffGeneral,6,6);
      StringGrid1.RowCount:=StringGrid1.RowCount+1;
      X:=X+H;
   End;
   // По концу цикла уничтожаем лишнюю строку.
   StringGrid1.RowCount:=StringGrid1.RowCount-1;
   // Выводим значения max и min.
   Edit1.Text:='Y='+FloatToStrF(Ymax,ffGeneral,6,6)+
           ' при '+'X='+FloatToStrF(Xmax,ffGeneral,6,6);
   Edit2.Text:='Y='+FloatToStrF(Ymin,ffGeneral,6,6)+
           ' при '+'X='+FloatToStrF(Xmin,ffGeneral,6,6);
end;
```

16.Нажав клавишу **F9**, запустить приложение на выполнение. Объяснить логику его функционирования. В случае необходимости выполнить отладку.

17.Сохранить отлаженный проект в папке размещения проекта лабораторной работы.

18.По результатам работы составить отчет.

Лабораторная работа № 4.

Элементы графики

Цель: Дальнейшее освоение работы с объектами ***Lazarus***. Использование специальных графических компонентов (элементов управления) для рисования графиков функций.

Рабочее задание: На основе проекта **Lab3** разработать новый проект, включив в него блок вывода на экран графика функции. Предусмотреть масштабирование графика, в зависимости от размеров окна.

Краткие теоретические сведения

Методы некоторых объектов, имеющих канву (точнее, методы самой канвы), например, форма, позволяют формировать графические изображения непосредственно, в том числе и рисовать графики. Однако среди многочисленных компонентов ***Lazarus*** имеется специальный объект типа ***TChart*** для графического представления числовых данных. Этот объект является панелью, на которой можно создавать диаграммы и графики различных типов. Компонент служит контейнером для объектов **Series** типа ***TLineSeries*** – серий данных, характеризующихся различными стилями отображения. Каждый компонент может включать несколько серий. Каждая серия имеет собственное имя, ее можно задавать программно как переменную. Многочисленные свойства самого компонента и его серий можно установить с помощью *Окна Свойств* или программно.

Компонент содержит большое количество специфических свойств, событий и методов.

Для программного создания графиков в лабораторной работе используются такие методы объектов **Chart** и **Series:**

- **ClearSeries** (очистка всех серий от занесенных ранее данных);
- **AddXY** (добавление в график новой точки).

Для вывода графика и таблицы значений можно было бы организовать отдельные формы, как в предыдущей работе, где была создана отдельная форма для вывода таблицы значений. В данной лабораторной работе, однако, применен другой прием: вся информация размещена в одной форме. А для того, чтобы иметь возможность попеременно переключаться с просмотра таблицы на просмотр графика, и на ввод данных использован компонент типа ***TPageControl*** (**Набор страниц с закладками**). Назначение этого компонента – создание нескольких, перекрывающих друг друга страниц (панелей) класса ***TTabSheet***. Каждая панель содержит свой набор компонентов. Доступ к конкретной странице осуществляется через связанный с ней корешок (закладку) – небольшой выступ над страницей, содержащий краткое название страницы.

Пример выполнения работы

1. Войти в среду ***Lazarus***. Создать новый проект.

2. Используя палитру компонентов (элементов управления), спроектировать форму **Form1,** изменить значение ее свойства **Name** в соответствии с требованиями, описанными в разделе 1 (**L4_<ФИО>_Form1**). Для свойства **BorderStyle** лучше оставить значение **bsSizeable** (устанавливается по умолчанию), что позволяет изменять размеры формы (окна) в процессе выполнения приложения**.**

3. В нижней части формы поместить элемент управления **BitBtn1** (командная кнопка) из страницы **Additional** с надписью «**Выход**». Растянуть его пошире, чтобы он имел ширину примерно такую, как форма. Чтобы при расширении формы этот компонент тоже растягивался, в сложном свойстве **Anchors** установит в ***true*** значения параметров **akLeft** и **akRight**. Установка в ***true*** значения параметра **akBottom** в этом же свойстве привяжет кнопку к нижнему краю формы: при изменении вертикального размера формы кнопка всегда будет находиться в нижней части формы.

4. Поместить на форму компонент **PageControl1** из палитры **Common Controls**. Расположить его так, чтобы он занимал всю оставшуюся после размещения командной кнопки часть формы.

5. Привязать созданный компонент **PageControl1** ко всем четырем сторонам формы, чтобы при изменении размеров формы соответственно изменялись и размеры компонента. Для этого в сложном свойстве **Anchors** значения всех параметров установить в ***true*****.**

6. Щелкая правой клавишей мыши на компоненте **PageControl1**, и выбирая пункт контекстного меню **Добавить страницу** (**New Page)**, создать три новых страницы.

7. Оставить названия полученных страниц **TabSheet1**… **TabSheet3** или присвоить новые по своему усмотрению. В полях свойств **Caption** написать: “**Ввод исходных данных”**, ”**Таблица значений”** и “**График функции”** соответственно.

8. Страница **TabSheet1** оформляется примерно как форма **Form1** в лабораторной работе № 3 (рис.11). С помощью буфера обмена можно скопировать все элементы этой формы на страницу **TabSheet1**:

9. Страница **TabSheet2** также оформляется аналогично форме **Form2** в лабораторной работе №3 (рис.12). На ней размещается компонент **StringGrid1** (текстовая таблица) из страницы **Additional** для размещения таблицы значений функции, два компонента **Label1** и **Label2**, а также два компонента **Edit1** и **Edit2** для размещения максимального и минимального значений функции. Определить свойства этих компонентов как в лабораторной работе №3. Для **StringGrid1** дать соответствующую привязку с помощью сложного свойства **Anchors.**

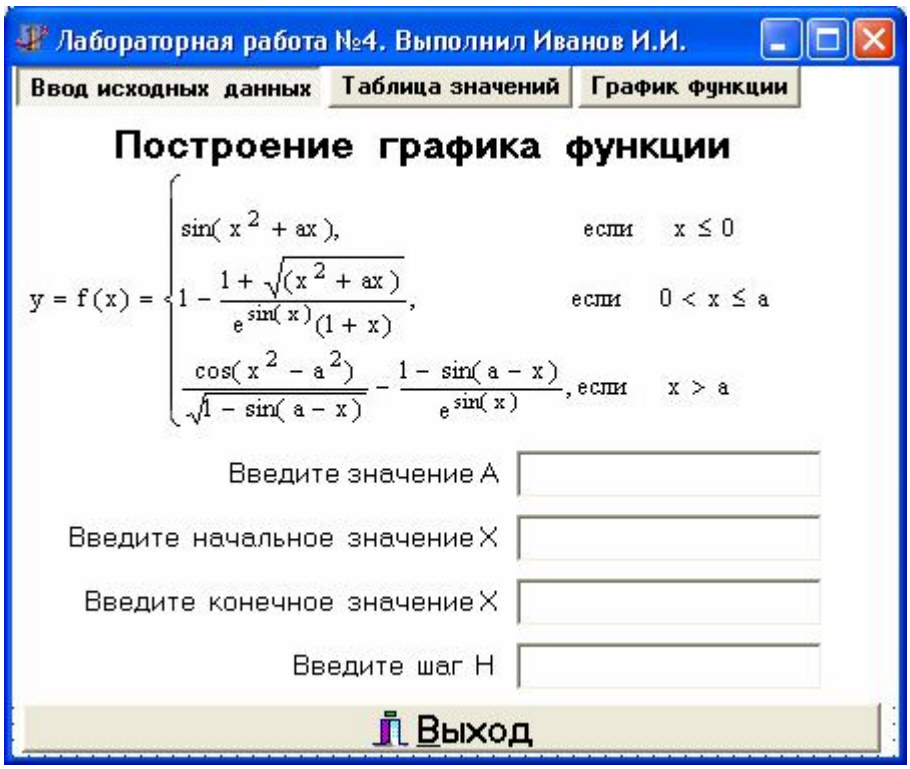

Рисунок 11 – Примерный вид первой страницы компонента **PageControl** проектируемой формы.

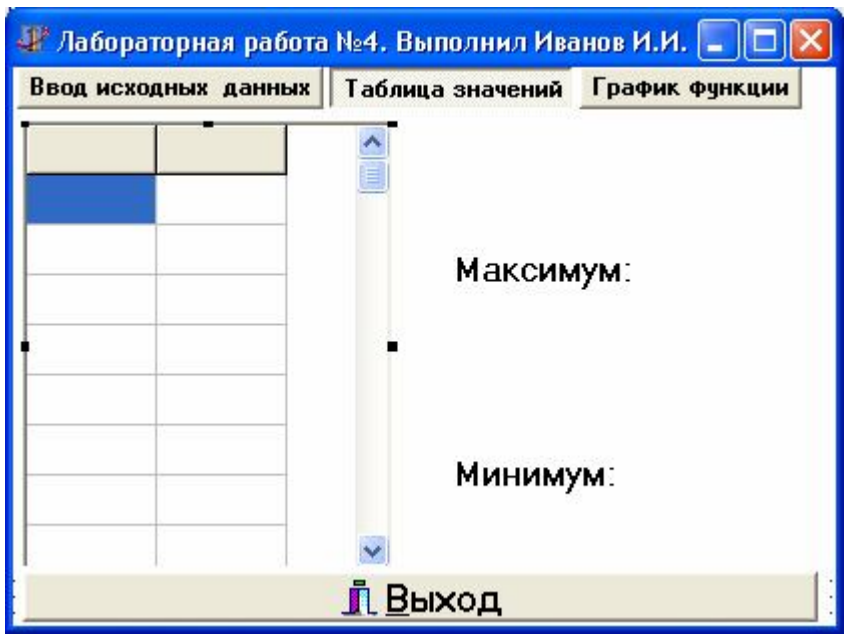

Рисунок 12 – Примерный вид второй страницы компонента **PageControl** проектируемой формы.

10. На третью страницу формы поместить компонент **Chart1** из палитры **Chart**, который осуществляет рисование графиков функций. Перейти на свойство **Titles** и ввести в подсвойстве **Text** заголовок: **“График функции, построенный по таблице значений”.** Сделать заголовок видимым:

Visible⇒true. Двойным щелчком на объекте **Chart1** открываем окно редактора графиков (**Edit series**), добавляем (**Add**) новый график (**Chart1LineSeries**). Для данного графика с помощью окна свойств избавляемся от легенды (**Legend⇒Visible⇒False**). Остальные параметры графика будут в дальнейшем задаваться программно. Привязать компонент **Chart1** ко всем четырем сторонам страницы компонента **PageControl** путем установки в ***true*** всех параметров свойства **Anchors**, установив предварительно необходимые размеры компонент **Chart1** (точнее расстояния от краев компонента **PageControl**). Удобнее в этом случае воспользоваться свойством **Align**, определив его значение равным ***alClient*** (растянуть на всю клиентскую область). Привязка компонента к границам контейнера (которым является компонент **PageControl**) означает, что при изменении размеров контейнера (страницы) будут соответственно изменяться и размеры компонента **Chart1**. А поскольку сам контейнер (компонент **PageControl**)**,** в свою очередь, привязан к границам формы, то при изменении размеров формы будут автоматически изменяться и размеры компонента **PageControl,** а значит и размеры компонента **Chart1**. При этом каждое изменение размеров компонента **Chart1** будет автоматически инициировать перерисовку содержащегося в этом компоненте графика. Страница **TabSheet3** будет иметь вид, примерно такой, как показано на рис.13.

Рисунок 13 – Примерный вид третьей страницы компонента **PageControl** проектируемой формы.

11. Создать или скопировать с предыдущих лабораторных работ функцию **f(x)** для вычисления значения функции.
12. Создать или скопировать с предыдущих лабораторных работ обработчики событий **OnExit** для однострочных окон редактирования.

13. Написать обработчик события **OnChange** для компонента **PageControl1.** В нем сначала необходимо выполнить проверку исходных данных. Эту часть кода можно скопировать из обработчика **Button1Click** предыдущей лабораторной работы, внеся в него небольшие коррективы: при передаче фокуса на компонент, в котором необходимо исправить (ввести) данные следует, кроме того, активизировать первую страницу компонента **PageControl1.** Вторую часть данного обработчика можно скопировать из обработчика **FormShow** предыдущей лабораторной работы. В любом случае обработчик события **OnChange** должен иметь примерно следующий вид:

```
procedure        TL4_Ivanov_Form1.PageControl1Change(Sender:
TObject);
   var H,Xn,Xk,A,X,Y,Ymax,Ymin,Xmax,Xmin:Real;
   var MySerie: TLineSeries;
begin
  If (Edit1.Text='') Then
     Begin
        ShowMessage('Отсутствует значение A');
        PageControl1.ActivePageIndex:=0;
        Edit1.SetFocus;
     End
  Else If (Edit2.Text='') Then
     Begin
        ShowMessage('Отсутствует начальное значение X');
        PageControl1.ActivePageIndex:=0;
        Edit2.SetFocus;
     End
  Else If (Edit3.Text='') Then
     Begin
        ShowMessage('Отсутствует конечное значение X');
        PageControl1.ActivePageIndex:=0;
        Edit3.SetFocus;
     End
  Else If (Edit4.Text='') Then
     Begin
        ShowMessage('Отсутствует шаг H');
        PageControl1.ActivePageIndex:=0;
        Edit4.SetFocus;
     End
  Else if (StrToFloat(Edit3.Text))<=
          (StrToFloat(Edit2.Text)) Then
     Begin
        ShowMessage('Начальное значение X должно быть' +
                    ' меньше конечного!');
        PageControl1.ActivePageIndex:=0;
        Edit3.SetFocus;
     End
  Else
```

```
    begin
       StringGrid1.RowCount:=1;
       StringGrid1.Cells[0,0]:='        X';
       StringGrid1.Cells[1,0]:='       f(X)';
       StringGrid1.GridLineWidth:=2;
       A:=StrToFloat(Edit1.Text);
       Xn:=StrToFloat(Edit2.Text);
       Xk:=StrToFloat(Edit3.Text);
       H:=StrToFloat(Edit4.Text);
       X:=Xn;Ymax:=-1e30;Ymin:=1e30;
       // Очищаем компонент Chart1. Там могут
       // остаться данные от предыдущего просмотра.
       // Создаем свою серию в компоненте Chart1
       // Определяем ее цвет (красный)
       Chart1.ClearSeries;
       MySerie:=TLineSeries.Create(Chart1);
       MySerie.LinePen.Color:=clRed;
       Chart1.AddSeries(MySerie);
       //Организуем цикл, вычисляющий f(x) в каждой точке
       // и строящий график функции.
       While X<=Xk Do
         Begin
           Y:=f(X,A);
           If Y>Ymax Then Begin Ymax:=Y;Xmax:=X; End;
           If Y<Ymin Then Begin Ymin:=Y;Xmin:=X; End;
           StringGrid1.RowCount:=StringGrid1.RowCount+1;
           StringGrid1.Cells[0,StringGrid1.RowCount-1]:=
                FloatToStrF(X,ffGeneral,6,6);
           StringGrid1.Cells[1,StringGrid1.RowCount-1]:=
                FloatToStrF(Y,ffGeneral,6,6);
           // Добавляем очередную точку в график функции.
           MySerie.AddXY(X,Y);
           X:=X+H;
          End;
       Edit6.Text:='Y='+FloatToStrF(Ymax,ffGeneral,6,6)+
          ' при '+'X='+FloatToStrF(Xmax,ffGeneral,6,6);
       Edit5.Text:='Y='+FloatToStrF(Ymin,ffGeneral,6,6)+
          ' при '+'X='+FloatToStrF(Xmin,ffGeneral,6,6);
       if (PageControl1.ActivePageIndex = 1) Then
          StringGrid1.SetFocus;
    End;
end;
```

14. При запуске проекта должна активироваться первая вкладка **PageControl1** и фокус передаваться однострочному текстовому полю для ввода A. Поэтому создаем обработчик **OnActivate** для формы:

```
procedure TL4_Ivanov_Form1.FormActivate(Sender: TObject);
begin
```

```
  PageControl1.ActivePageIndex:=0;
  Edit1.SetFocus;
end;
```

15. Нажав на клавиатуре клавишу **F9**, запустить приложение на выполнение. В случае необходимости выполнить отладку. Объяснить работу приложения.

16. Сохранить отлаженный проект в папке размещения проекта лабораторной работы.

17. По результатам выполнения работы составить отчет.