МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. И.А. БУНИНА»

И.И. Васильева

СИСТЕМНОЕ И ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Учебное пособие

Елец -2019

УДК
ББК

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А. Бунина
от _____, протокол №____

Рецензенты:
А.А. Самойлов, учитель информатики МБОУ Гимназия №97

И.И. Васильева
_____ Системное и прикладное программирование: учебное пособие. – Елец: Елецкий государственный университет им. И.А. Бунина, 2019. - ____ с.

В учебном пособии приводятся задачи, решаемые средствами систем программирования Lazarus и Microsoft Visual Studio, включающие в себя языки программирования Free Pascal, а также Visual Basic, Visual C# и Python. Данное учебное пособие предназначено для студентов СПО, обучающихся по специальностям 09.02.03 – «Программирование в компьютерных системах», 09.02.02 – «Компьютерные сети». Оно может быть использовано учителями информатики в рамках факультативных занятий в старших классах, а также преподавателями в колледжах и училищах.

УДК
ББК

Оглавление

Введение

Прикладные программы – это приложения, которые необходимы пользователям для решения конкретных задач. Они делятся на два вида – программы общего назначения и программы специального назначения. Например, текстовый редактор относится к программам общего назначения, а программа для бухгалтерского учета на предприятии – к специальному виду. Но так или иначе, их функционирование невозможно без системных программ. К системным в первую очередь относится операционная система. Прикладные программы обращаются к ней через системные вызовы. Даже в самых простых задачах необходимо реализовывать открытие и сохранение файла, обращаться к функциям вставки даты и времени.

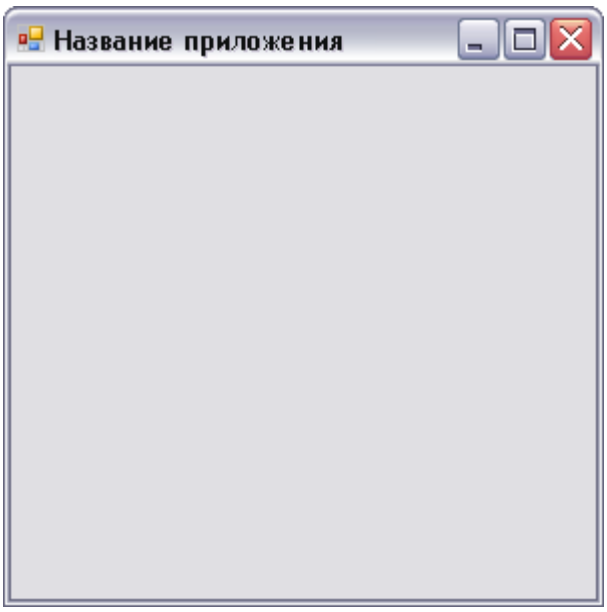
Графический пользовательский интерфейс – это взаимодействие между приложением и пользователем посредством произвольного доступа к объектам с помощью устройств ввода. Иными словами, кнопки, диалоговые окна, поля ввода информации, расположенные в окне любой программы.

Форма – это основной элемент приложения с графическим интерфейсом. Свойства формы (или окна приложения) зависят от использованной системы программирования, но большинство стандартных свойств должны поддерживать все среды. К форме можно применять различные методы, т.е. функции, которые можно выполнить после написания соответствующего кода. Условно формой называют окно во время его редактирования, т.е. когда приложение еще не сделано, то оно имеет форму, а когда оно уже готово и выполняется, то оно содержит окно.

Свойства формы можно настроить в окне настройки свойств, а можно определить программно. Например, заголовок формы:
Свойства – Text – вместо Form1 название программы.
Или:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.Text = "Название приложения"
    End Sub
```

В этом примере на Visual Basic при открытии данной формы меняется название на текст в кавычках.



В таком случае можно управлять данной функцией, например, менять название при нажатии на разные кнопки или дописывать имя открытого файла.

Аналогичные процедуры можно написать и для других систем – Visual C#, Delphi, Lazarus, PascalABC.Net. Некоторые среды программирования поддерживают создание оконного интерфейса, но не имеют визуального редактора. К ним относятся Visual C++, Python и др. Тогда все свойства всех элементов формы должны записываться вручную.

Глава 1. Элементарные оконные приложения

Задача 1.1

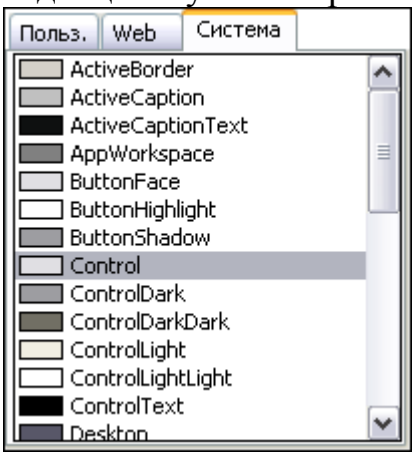
Рассмотрим простейшую задачу: изменить цвет основного окна формы на выбранный пользователем при нажатии на кнопку.

На форму наносятся компоненты – элементы управления. Часть из них являются видимыми, например, поле для ввода текста. Другие – скрытые, например, диалоговые окна. Такое окно появляется на экране только после совершения какого-либо действия.

Выберем в начальном окне Microsoft Visual Studio Создать проект - Visual Basic – Приложение Windows Forms. На панели элементов «Стандартная» выберем компонент Button (кнопка) и переместим его в форму двумя способами: 1) щелкнуть мышью по слову «Button» и, не отпуская кнопку, перетащить на форму; 2) щелкнуть мышью по слову «Button», щелкнуть еще раз мышью в нужном месте на форме. Название элемента по умолчанию – Button1. В свойствах оно встречается два раза – как имя объекта, и как выводимый текст на кнопке. Свойство «text» (текст) изменим, написав «ОК» вместо Button1, а свойство «(Name)» (имя) оставим без изменения. Объектам, как и переменным, можно давать собственные имена-идентификаторы, но выбор имени должен быть ответственным, чтобы не повлечь ошибки в программе.

Затем поместим еще один компонент с панели «Диалоговые окна» - ColorDialog . В VB он переместится под нашу форму, т.е. при открытии окна приложения этот элемент пока не заметен. Теперь рассмотрим свойства компонента «форма», а также метод элемента «ColorDialog» и «кнопка».

В настройках формы (окно «Свойства») есть параметр – BackColor (задний цвет). Он отвечает за смену фонового цвета. Для выбора списка с цветом надо щелкнуть по стрелке справа от BackColor:



В этом окне можно выбрать не так много стандартных системных и пользовательских цветов, и изменить это свойство можно один раз перед запуском программы. Для решения нашей проблемы мы и добавили элемент «ColorDialog». Как по-русски (или по-английски) указать то, что нужно открыть диалоговое окно? По смыслу подходят глаголы выполнить (execute)

или показать (show). В среде VB используется функция ShowDialog – показать диалог. А какая функция будет отвечать за выбор цвета из этого диалогового окна? Еще проще – Color (т.е. цвет). Осталось соединить элемент с его свойством или функцией воедино. Для этого используется разделительная точка.

Подытожим наши рассуждения: если элемент «диалог цвета» показан, то установить цвет формы такой же, как и выбранный цвет из диалогового окна.

Теперь запишем эту же фразу на Visual Basic:

If ColorDialog1.ShowDialog then Me.BackColor=ColorDialog1.Color

Единственный нюанс – вместо имени элемента Form1 (как в других системах) в VB надо писать Me (мой, моя), т.е. обращение к текущей форме. Осталось выяснить, куда поместить эту строчку кода. В условии задачи сказано: «при нажатии на кнопку». Это значит, что для элемента Button необходимо реализовать обработчик события. Событие – это одинарный щелчок левой кнопкой мыши, а обработчик – это строки кода, относящиеся к процедуре этого события. Т.к. нажатие – самое распространенное событие для кнопки, то вызвать редактор кода с готовым заголовком процедуры можно двойным щелчком мыши по кнопке «ОК». Для более редких событий (вообще для всех) применяется второй способ: в свойствах компонента «кнопка» нажать на значок в виде молнии (события), выбрать нужное и щелкнуть два раза мышью по названию. Осталось дописать вышеуказанную строку, чтобы получилось следующее:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        If ColorDialog1.ShowDialog Then Me.BackColor = ColorDialog1.Color
    End Sub
```

Разберем первую и третью строки:

Private – означает, что следующая процедура доступна только для данного окна программы, т.е. приватная.

Sub – ключевое слово, означающая «процедура».

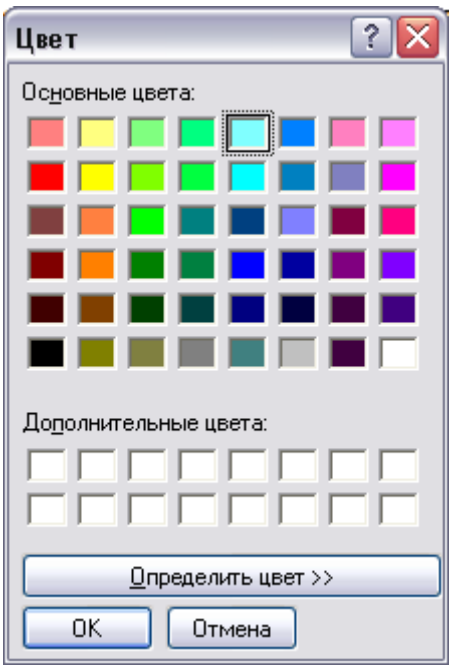
Button1_Click – имя процедуры. В данном случае оно состоит из имени элемента - «кнопка» (Button1) и имени события – щелчок (Click).

Параметры в скобках и вид управления VB формирует автоматически.

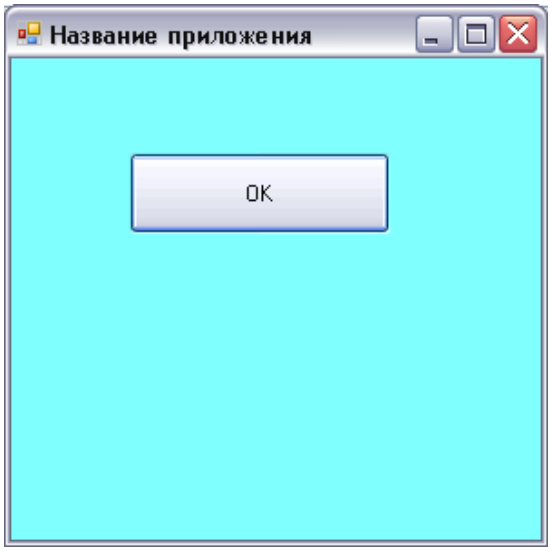
End Sub – конец процедуры, т.е. нашего фрагмента программы.

Для запуска нашей программы надо нажать кнопку «Пуск», расположенную выше формы.

После запуска приложения и нажатия на кнопку появится диалоговое окно:



Выберем голубой цвет и увидим изменения на форме:



Сама кнопка цвет не поменяла. Для этого нужно найти свойство «цвет» в настройках самой кнопки, а не формы. А цвет заголовка в данном случае зависит от системных настроек Windows.

Аналогично задача решается в среде Delphi (или бесплатном аналоге Lazarus), а также в C#

Например, процедура обработки той же кнопки в Lazarus имеет вид:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if colordialog1.Execute then form1.Color:=colordialog1.Color;
end;
```

Перед написанием кода необходимо изменить для заголовка формы и текста кнопки свойство «Caption» - заголовок.

Тот же пример в Visual C#

```
private void button1_Click(object sender, EventArgs e)
```

```
        {
if (colorDialog1.ShowDialog() == DialogResult.OK) this.BackColor = colorDialog1.Color;
        }
```

Теперь рассмотрим язык программирования Python.

Python – это интерпретируемый интерактивный объектно-ориентированный язык. Напрямую действия выполняются в командной строке, графический интерфейс не поддерживается. Но можно дополнительно подключить библиотеку Tkinter (Tk interface) или Qt5 (), а затем реализовать функции как графики, так и интерфейса, используя для последнего набор виджетов.

В онлайн-средах эти библиотеки не поддерживаются, т.к. виджеты опираются непосредственно на функции операционной системы.

```
from tkinter import *
from tkinter import colorchooser
root = Tk()
def onChoose():
        color = colorchooser.askcolor()
        color_name=color[1]
        root.configure(background=color_name)
root.title("Color")
root.geometry("300x150+300+300")
btn=Button(text="OK",  padx="20", pady="8", command=onChoose)
btn.pack(side=BOTTOM)
root.mainloop()
```

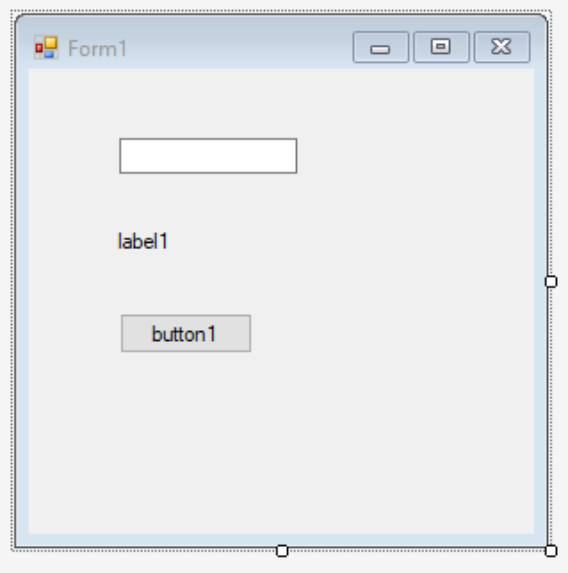
Рассмотрим каждую строку кода подробно:

1) подключение графической библиотеки;
2) дополнительное подключение диалогового окна выбора цвета;
3) инициализация графического окна;
4) заголовок функции пользователя;
5) присваивание переменной выбора цвета значения из диалогового окна;
6) присваивание полученного значения переменной для установки параметров цвета формы;
7) конфигурация (настройка) фонового цвета формы через полученное значение;
8) начало основной программы, установка заголовка формы;
9) установка размеров окна формы;
10) инициализация кнопки с параметрами относительного размера и определением действия по щелчку (т.е. вызов функции пользователя);

11) установка кнопки на форме снизу;
12) запуск окна приложения.
Для удобства работы с интерфейсом в Visual C++ и Python предусмотрено специальное приложение Qt Designer, позволяющее сохранить файл с формой, которое можно интегрировать в дальнейшую программу. Но все дальнейшие изменения придется выполнять в текстовом режиме.

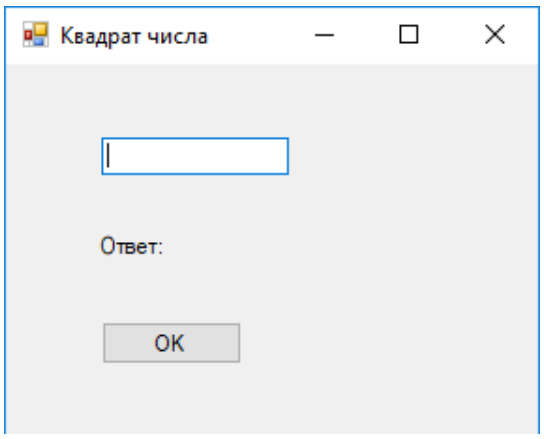
Задача 1.2

Усложним задачу. Пусть на форме заданы три элемента управления: поле для ввода текста, неизменяемое поле для вывода и кнопка. При нажатии на кнопку производится расчет по формуле $y=x^2$. Соответственно, x вводится в поле ввода, а результат y выводится в поле вывода.
1) Microsoft Visual C# (аналогичные манипуляции и в Visual Basic).



Поместим на пустую форму три компонента: TextBox, Label, Button.
Установим свойства для каждого из них, включая саму форму.
Зададим заголовок формы в свойстве Text – «Квадрат числа».
Для наглядности можно запустить обработчик события загрузки формы — FormLoad(), в качестве кода которого явно записать нужные методы:

```
private void Form1_Load(object sender,
EventArgs e)
        {
           this.Text = "Квадрат числа"
            label1.Text = "Ответ: "
            textBox1.Text = ""
            button1.Text = "OK"
        }
    }
```



Но при нажатии на кнопку «ОК» ничего не произойдет, т. к. у нас нет связанного с ней события.
Пока разберемся с первым обработчиком:
Будем учитывать, что C# чувствителен к регистру букв, поэтому не станем менять свойство Name (имя) у каждого компонента, оставив по умолчанию, чтобы не ошибиться в написании.
this.Text – текст заголовка формы. Ключевое слово «this» означает, что все свойства и методы реализуются только в этой (данной) форме.
label1.Text – содержимое неизменяемого поля вывода. Слово «label» дословно переводится как «метка, ярлык». С помощью этого элемента мы можем программно вывести любую надпись, но изменить ее пользователем в процессе работы нельзя. Этот компонент удобно использовать для подписей блоков ввода информации и вывода результатов.

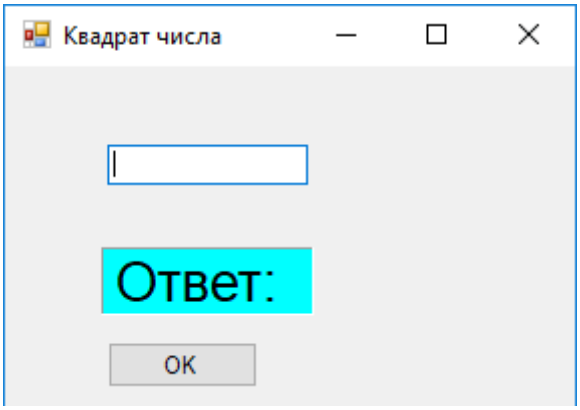
Некоторые другие свойства элемента label в C#:

свойство	описание
AutoSize	Включает или выключает изменение размера в соответствии с размером шрифта для однострочной надписи. Значение True увеличивает или уменьшает размер компонента в соответствии с текстом, а значение False оставляет рамку неизменной.
BackColor	Цвет фона за текстом. Вместе со свойством BorderStyle (стиль обрамления) позволяет выделить фрагменты интерфейса.
ContextMenuStrip	Контекстное меню, вызываемое нажатием правой кнопки мыши. Можно запрограммировать копирование в буфер обмена содержимого этого элемента.
Cursor	Выбор значка для курсора, по умолчанию — стрелка.
Enabled	Если значение этого свойства стоит в False, то этот компонент неактивен. Для элемента Label это не столь принципиально, а для кнопок и полей ввода можно разрешать или запрещать действия.
Font	Группа свойств, отвечающих за вид и размер шрифта.
Image	Вместо однотонного фона можно поместить изображение и отдельно задать его свойства
Location	Позиция элемента форме на основе координат верхнего левого угла.

Locked	Запрещает или разрешает перемещать элемент по экрану и менять его размеры.
Size	Определяет точный размер элемента. Существуют и другие свойства, позволяющие настроить границы размера компонента.
Visible	Показать (true) или скрыть (false) элемент на экране.

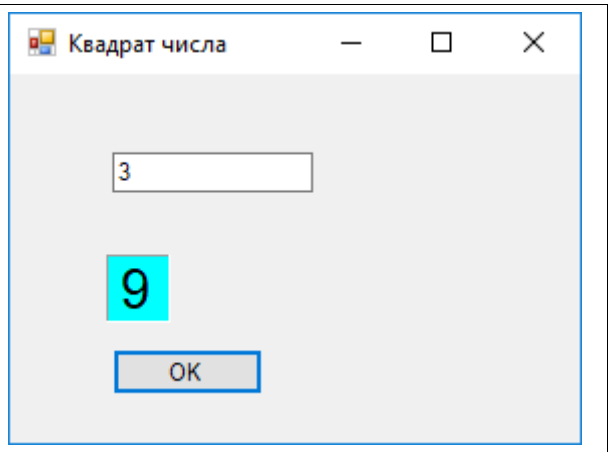
Остальные функции используются реже.
Добавим значения некоторых свойств программно в существующий код:

```
...
label1.BackColor = Color.Aqua;
label1.BorderStyle = BorderStyle.Fixed3D;
this.label1.Font = new Font("Arial", 20F);
...
```



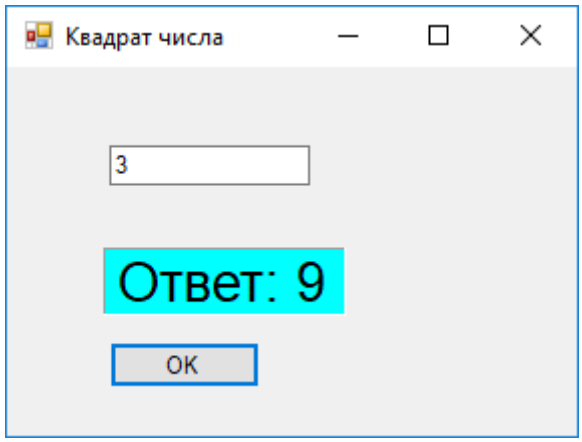
В первой строке мы поменяли свойство Color (цвет) на Aqua (бирюзовый), во второй изменили стиль рамки – fixed3D (фиксированная трехмерная граница). В языке C# нельзя использовать название шрифта в присваивании, поэтому для смены начертания и размера шрифта задается новый образец new Font. С помощью ключевого слова «new» можно добавить в коде любой элемент управления или переопределить набор параметров.
TextBox1.Text — изменение текста в поле ввода.
Большинство свойств этого компонента совпадает с Label, однако, добавлены параметры, отвечающие за ввод. Строго говоря, выводить результат можно тоже с помощью этого элемента, но для защиты выходной информации от доступа пользователя лучше применять разные компоненты.
Особенностью текстового поля является то, что обрабатывать можно только строковые значения. Для работы с числовыми данными необходимо использовать функции преобразования текста в число и обратно (при выводе).
Например, подхват значения x из TextBox, расчет значения функции и вывод его в Label при обработки события нажатия на кнопку в C# имеет вид:

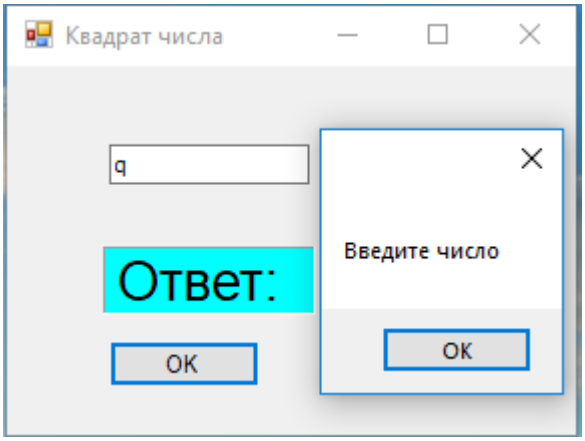
```
 private void button1_Click(object sender,
EventArgs e)
        {
float x = Convert.ToSingle(textBox1.Text);
float y = x * x;
 label1.Text = Convert.ToString(y);
        }
```



Разберем строки кода.
Определим типы переменных как вещественные float (т. е. на языке математики — округленные действительные числа). Для преобразования строки ввода в числовое значение используется метод Convert.ToSingle(). Для обратного перевода числового результата в строку вывода — метод Convert.ToString().
Наша программа работает, но некорректно.
1 недостаток: пропало слово «ответ»;
2 недостаток: при вводе неправильных данных, например, буквы вместо числа, программа прекращает свою работу.
Перепишем программу с учетом замечаний:

```
 private void button1_Click(object sender,
EventArgs e)
        {
            float y=0;
            float x;
if (float.TryParse(textBox1.Text.Trim(),
out x))
            {
                y = x * x;
label1.Text += Convert.ToString(y);
            }
            else
 MessageBox.Show("Введите число");


        }
```





Метод .Parse используется для преобразование любого значения, преимущественно строкового, в значение определенного типа, а метод .TryParse() возвращает логическое значение, обозначающее произошло ли преобразование, и возвращает преобразованное значение в параметре out. В данном примере сначала заранее определили тип переменных, выходному значению присвоили начальное значение ноль. В строке `if (float.TryParse(textBox1.Text.Trim(), out x))` параметром метода .TryParse() является текст, находящийся в поле ввода, но для точности преобразования из него удалили крайние пробелы (с помощью метода .Trim()). Если содержимое текстового поля можно преобразовать в вещественное число, то результат поместим в переменную x. Тогда строка

```
x = Convert.ToSingle(textBox1.Text);
```

не нужна. Тогда после расчета к тексту в поле вывода добавится преобразованный в строковый тип результат, т.е.:

```
label1.Text = label1.Text + Convert.ToString(y);
```

В коде программы использовался сокращенный оператор присваивания, как в языке C++.

В противном случае, когда в textbox введено не число, на экране с программой появляется диалоговое окно MessageBox с информацией о неверных данных. Этот элемент вместе с методом Show() – показать – имеет много дополнительных параметров, например, настройку заголовка окна, иллюстрации информационного сообщения и наименования кнопок. В

простейшем случае достаточно одного параметра – текста сообщения, а кнопка «ОК» с функцией закрытия окна реализована по умолчанию автоматически.

Аналогичные рассуждения для программы, составленной на Visual Basic:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.Text = "Квадрат числа"
        Label1.Text = "Ответ: "
        TextBox1.Text = ""
        Button1.Text = "OK"
        Label1.BackColor = Color.Aqua
        Label1.BorderStyle = BorderStyle.Fixed3D
        Me.Label1.Font = New Font("Arial", 20.0F)
    End Sub

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim x, y As Single
        If IsNumeric(TextBox1.Text) Then
            x = Val(TextBox1.Text)
            y = x ^ 2
            Label1.Text = Label1.Text + Str(y)
        Else
            MsgBox("Введите число")
        End If
    End Sub
End Class
```

В этом листинге приведены две процедуры – FormLoad() для установки параметров элементов управления и Button_Click() для проведения расчета при нажатии на кнопку.

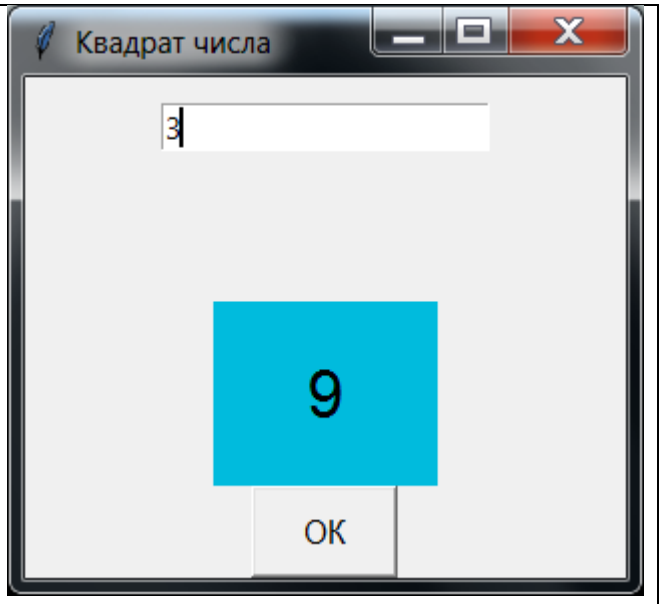
В этой системе программирования есть удобная функция IsNumeric(), позволяющая выяснить, является ли аргумент этой функции числом или нет.

Сравните этот код с подобным в Lazarus:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  form1.caption := 'Квадрат числа';
       Label1.caption := 'Ответ: ';
       edit1.Text := '';
       Button1.caption := 'OK';
       Label1.Color := clAqua;
       Label1.BorderSpacing.InnerBorder := 1;
       Label1.Font.Name:='Arial';
       label1.Font.Size:=20;
end;
procedure TForm1.Button1Click(Sender: TObject);
var x, y: real;
begin
         if TryStrToFloat(edit1.Text,x) then
          begin
          y := x *x;
          Label1.caption := Label1.caption + FloatToStr(y);
          end
         else
          showmessage('Введите число');
end;
```

Эта же программа на языке Python

```
from tkinter import *
import math
y = 0
def click_button():
    global y
    y =pow(int(x.get()),2)
    label1=Label(text=str(y),
padx="35", pady="25", height="1",
width="2",font = "Arial 20",
background="#0bd")
    label1.pack(side=BOTTOM)
root = Tk()
root.title("Квадрат числа")
root.geometry("300x250")
btn = Button(text="OK", padx="20",
pady="8", font="16",
command=click_button)
btn.pack(side=BOTTOM)
x = StringVar()
message_entry = Entry(textvariable=x)
message_entry.place(relx=.5, rely=.1,
anchor="c")
root.mainloop()
```



Т.к. Python – интерпретатор, то даже окно приложения запускается вместе с командной строкой, поэтому не обязательно добавлять в код условие правильности набора числа. При ошибочном вводе данных в командной строке можно увидеть подробное сообщение об ошибке.

Рассмотрим данный код построчно:

1) загрузка функций из графической библиотеки;
2) загрузка математической библиотеки;
3) установка начального значения вычисляемой функции;
4) заголовок функции пользователя, вычисляемой при нажатии на кнопку;
5) объявление глобальной переменной;
6) вычисление квадрата числа при выборе аргумента из текстового блока;
7) размещение результата в блоке вывода label с одновременной установкой параметров этого блока (преобразованный в текст результат, расстояния от границ окна, высота и ширина, шрифт и размер шрифта, цвет фона);
8) расположение элемента вывода на форме снизу;
9) основная программа: инициализация графического окна;
10) текст заголовка окна формы;
11) размеры формы;
12) параметры кнопки (нанесенный текст, относительное расположение, размер шрифта, привязка к процедуре – обработчику события);
13) расположение кнопки на форме снизу;
14) объявление строковой переменной;
15) связь переменной с содержимым текстового блока ввода (Entry);
16) относительное расположение текстового блока на форме;
17) открытие приложения.

Глава 2. Математические и физические задачи

Одними из самых популярных задач, решаемых на компьютере, являются задачи из области математики и физики, в которых требуется производить вычисления по формулам. Разберем классические задания по геометрии, алгебре и физике в разных системах программирования.

Задача 2.1

Постановка задачи. Собрать типовые формулы школьного курса планиметрии и стереометрии, описывающие свойства геометрических фигур, в единое приложение.

Внешнее описание: выделить 5 фигур на плоскости и 5 фигур в пространстве, установить основные свойства и измерения (площади, высоты, объемы и т.д.).

Функциональная спецификация: решение по каждой фигуре производится в отдельном окне, доступ к каждому из которых осуществляется с помощью основного окна. Итого в проекте задействовано 11 форм. В файле ресурсов хранятся изображения фигур, которые используются как на главной форме (в виде изображения на кнопках), так и на второстепенных формах.

Геометрические фигуры:

1) Треугольник: ввести – 3 стороны; рассчитать - площадь, высоты, медианы, биссектрисы, углы, радиусы вписанной и описанной окружностей; изобразить – разные типы треугольника в зависимости от вычисленных углов (произвольный или прямоугольный).

2) Окружность: ввести – радиус; рассчитать – длину окружности, площадь круга; изобразить – одну окружность схематично.

3) Трапеция: ввести – 4 стороны; рассчитать – площадь, углы; изобразить – трапецию схематично.

4) Параллелограмм: ввести – 2 смежные стороны и угол (в градусах) между ними; рассчитать – площадь, диагонали; изобразить – фигуру в зависимости от равенства сторон и углов (произвольный параллелограмм, прямоугольник, квадрат, ромб).

5) Правильный многоугольник: ввести – количество сторон, длину стороны; рассчитать – площадь, углы, радиусы вписанной и описанной окружностей; изобразить – фигуру по числу сторон от треугольника до октаэдра.

6) Цилиндр: ввести – радиус и высоту; рассчитать – объем и площадь боковой поверхности; изобразить – цилиндр с нанесенными значениями радиуса и высоты.

7) Шар: ввести – радиус; рассчитать – объем и площадь сферической поверхности; изобразить – шар схематично.

8) Конус: ввести – радиус основания и высоту; рассчитать – объем и площадь боковой поверхности; изобразить – конус схематично.

9) Прямоугольный параллелепипед: ввести – длину, ширину, высоту; рассчитать – объем и площадь боковой поверхности; изобразить – параллелепипед схематично.

10) Четырехугольная пирамида: ввести – 2 стороны основания, высоту; рассчитать – объем и площадь боковой поверхности; изобразить – пирамиду схематично.
Система программирования: Microsoft Visual Studio C#.
Особенности реализации и возникшие трудности: доступ к каждой фигуре осуществляется 2 способами – через пункты меню и через кнопки на главной форме. На каждой вспомогательной форме есть кнопка «Вернуться». При переключении между формами может быть не корректная работа: закрытие основной формы, оставление приложения в «Диспетчере задач» системы. Для решения этих проблем в C# имеются методы управления появлением и скрытием форм show(), close() и hide(), собственный обработчик события системной кнопки окна «Закрыть», пункт меню «Выход» с кодом закрытия всех форм принудительно.
Достоинства приложения: проверка некорректного ввода данных для треугольника, многоугольника и др.; отсутствие лишних окон на экране.
Недостатки приложения: дублирование частей кода; минимальная справочная информация.
Разработка интерфейса. В приложении использовались следующие компоненты: меню (MenuStrip), кнопки (Button), всплывающие подсказки (ToolTip), текстовые поля (TextBox), статический текст (Label), окно графического вывода (PictureBox).
В среде Microsoft Visual Studio существует редактор меню. С одной стороны, нужные пункты можно добавлять сразу на форме. С другой стороны, в контекстном меню элемента menuStrip1 (по умолчанию) можно вызвать диалоговое окно с возможностью редактирования.

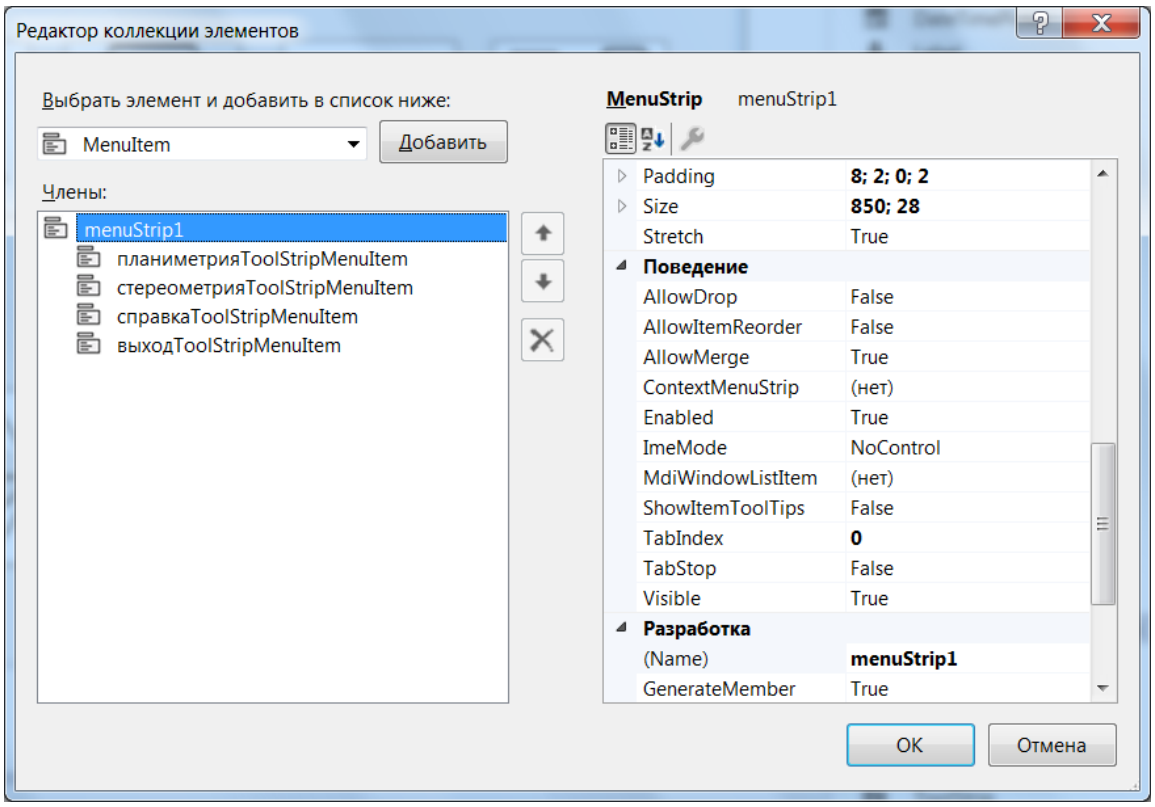


Рис. 2.1. Редактор меню C#

В этом редакторе можно менять пункты меню местами, настраивать клавиши быстрого доступа к пунктам, управлять размерами пунктов и расстоянием между ними.

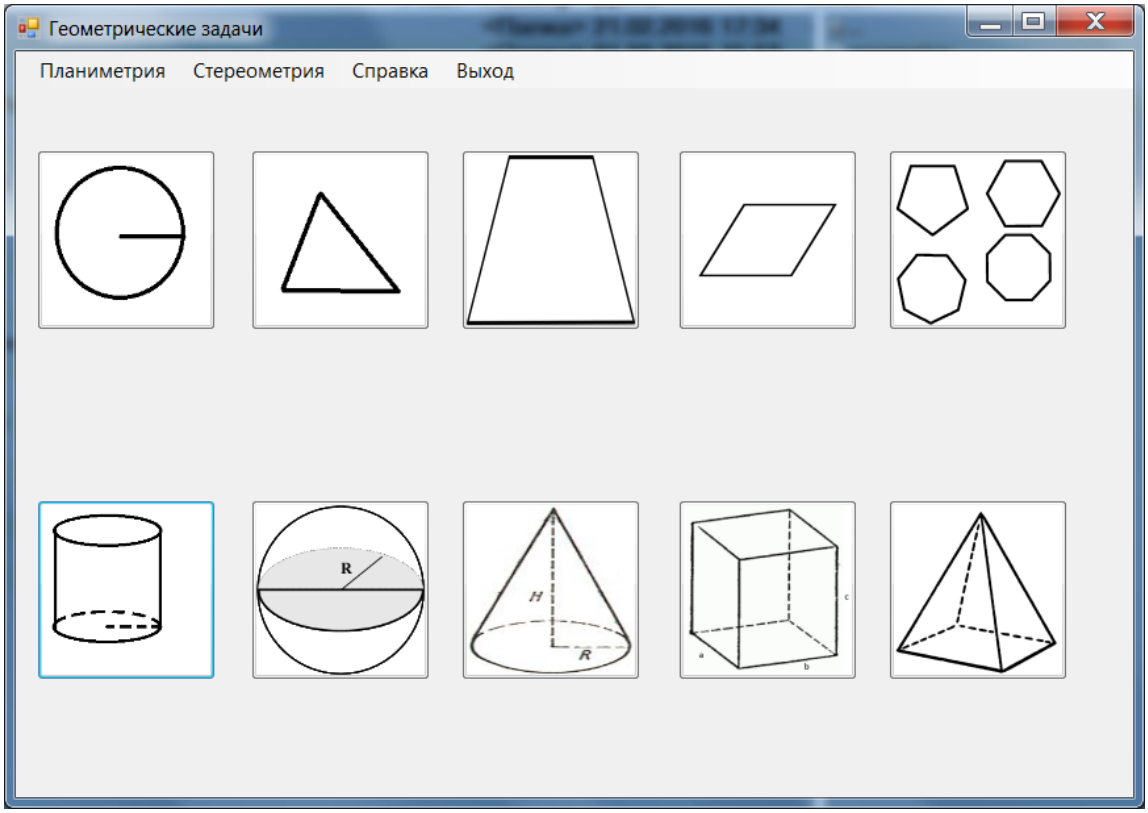
Для дополнительного выбора окна с вычисляемыми объектами на основной форме располагаются 10 кнопок. Изображение на кнопке выбирается из файла ресурсов с помощью свойства BackgroundImage. Для этого свойства можно установить дополнительный атрибут расположения рисунка BackgroundImageLayout – растянуть изображение по размеру кнопки (Stretch), замостить плиткой (Tile), разместить по центру (Center), масштабировать (Zoom). Нужный параметр выбирается в зависимости от размера изображения и кнопки, при равных значениях можно оставить этот атрибут без изменений (None).

Всплывающие подсказки так же можно оформить с помощью элемента управления, но в данном приложении они устанавливаются программно в процедуре загрузки формы:

```
private void Form1_Load(object sender, EventArgs e)
        {
            ToolTip t = new ToolTip();
            t.SetToolTip(button1, "Цилиндр");
              …
```

Добавить оставшиеся описания для остальных кнопок.

Окончательно основная форма с выбором фигуры имеет вид:



В формах 2-11 для ввода данных пользователем применяются текстовые поля TextBox. Для того, чтобы осуществить числовую обработку данных из текстового поля, необходимо перевести тип данных из строкового в

вещественный (или целочисленный) формат. В C# для этого используется команда:

```
r = Convert.ToDouble(textBox1.Text);
```

Вывод результата производится в статическое текстовое поле. Для этого сначала необходимо переконвертировать значение обратно из числового типа данных в строковый:

```
label3.Text = label3.Text+" "+Convert.ToString(v);
```

Т.к. в поле Label уже стояло значение «Объем равен», то к этому значению добавляем через пробел второе значение с результатом.

С помощью компонента PictureBox на форму можно добавить любое изображение. Если картинки будут меняться в зависимости от условия, то они устанавливаются не через окно свойств компонента, а в программе, например:

```
if (alpha == 90 || beta == 90 || gamma == 90)
                {
                    pictureBox1.Image = Properties.Resources.pryamtr;
                    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
                }
                else
                {
                    pictureBox1.Image = Properties.Resources.triangle;
                    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
                }
```

В этом фрагменте программы записано условие: если один из углов равен 90 градусов, то вывести изображение прямоугольного треугольника из ресурсов, в противном случае изобразить обычный треугольник. Сразу в коде устанавливается атрибут положения изображения внутри компонента (PictureBoxSizeMode.StretchImage) – «растянуть по размеру».

Полный код с расчетом по формуле и выводом результатов выполняется при нажатии на кнопку «Рассчитать». На каждой из вспомогательных форм располагается вторая кнопка «Вернуться на главную» со следующим кодом:

```
private void button2_Click(object sender, EventArgs e)
        {
            Form1 form1 = new Form1();
            form1.Show(this);
            Hide();
        }
```

Алгоритм этого блока следующий:

1) инициализировать форму Form1;
2) показать эту форму на экране;
3) спрятать с экрана предыдущую форму (но не закрывать).

Для принудительного закрытия формы надо изменить код стандартной кнопки «Закрыть» (красный крестик). В противном случае главное окно не появится на экране.

```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
        {
            Form form1 = Application.OpenForms[0];
            form1.Show();
        }
```

Для инициализации процедуры «FormClosed» надо в свойствах элемента «Форма» (например, Form2) выбрать пункт «События» и найти событие «FormClosed».
Все добавленные в проект формы автоматически нумеруются, начиная с нуля, поэтому Form1 можно не инициализировать заново, а выбрать из списка Application.OpenForms[0].
В главной форме для корректного закрытия приложения добавлен пункт меню «Выход» с кодом:

```
private void выходToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();

        }
```

Пункт «Справка» реализован упрощенно. Это обычное диалоговое окно вывода MessageBox:

```
private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Текст справки", "Справка", MessageBoxButtons.OK,
MessageBoxIcon.Question);
        }
```

Полный код программы представлен в электронном приложении

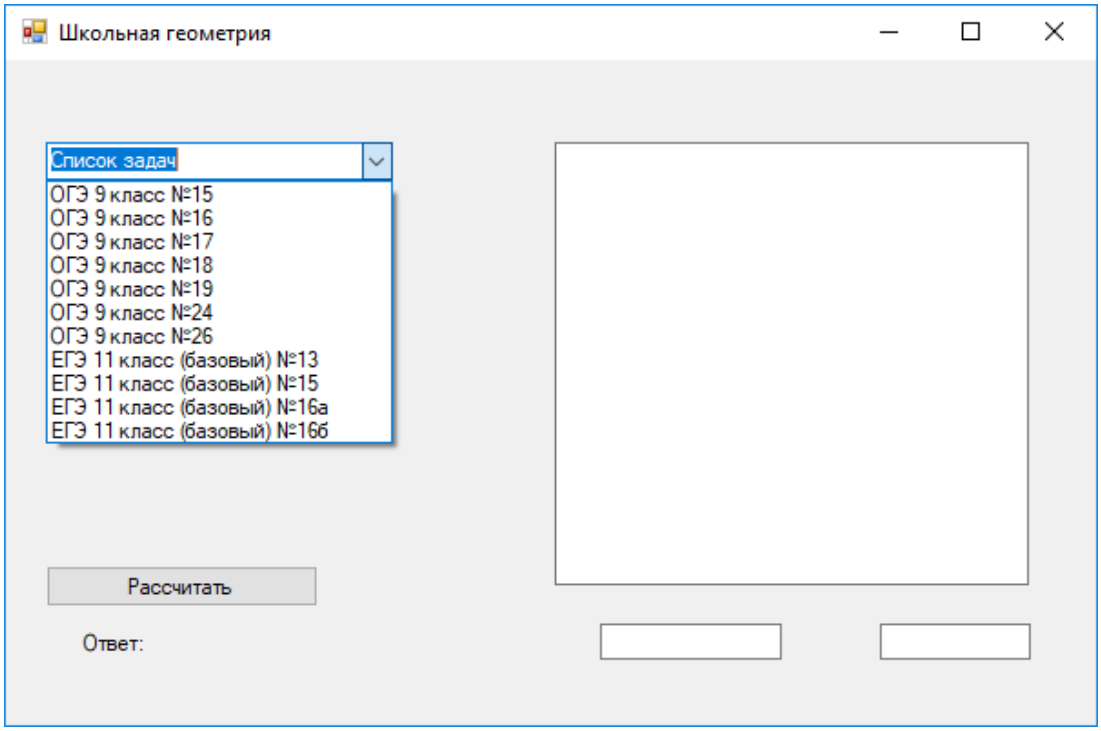
Задача 2.2

Постановка задачи. Решить типовые задачи из школьного курса геометрии, используя контрольно-измерительные материалы ОГЭ 9 класс и ЕГЭ 11 класс (базовый уровень).
Внешнее описание: условие задачи выбрать из списка, ввести необходимые по смыслу данные и получить числовой результат.
Функциональная спецификация: список содержит 10 пунктов. Условие задачи отображается в текстовом поле. В отдельных текстовых полях пользователем вводятся входные данные. Расчет производится в отдельном модуле. Вывод осуществляется в том же окне при нажатии на кнопку «Рассчитать». Количество текстовых полей с подписями для ввода данных регулируется программно в зависимости от поставленной задачи.
Система программирования: Microsoft Visual Studio Visual Basic.
Особенности реализации и возникшие трудности: для каждого условия изображается чертеж. В тексте числовые параметры обозначаются в виде переменных (a, x, h…), а подписи к полям ввода содержат эти обозначения. Выбор задачи, чертежа и полей ввода осуществляется в процедуре ComboBox1_SelectedIndexChanged, а выбор модуля решения и вывода результата — в процедуре Button1_Click.
Достоинства приложения: корректная обработка десятичной точки; отсутствие лишних окон на экране.
Недостатки приложения: дублирование частей кода; отсутствие справочной информации.
Разработка интерфейса и программная реализация. В приложении использовались следующие компоненты: редактируемый раскрывающийся

список (ComboBox), кнопки (Button), текстовые поля (TextBox), статический текст (Label), окно графического вывода (PictureBox).
Значения (строки) в список фиксированы и вводятся заранее с помощью редактора свойств этого элемента управления. Основной функцией этого компонента является ComboBox1.SelectedIndex — номер выбранной строки (на которой стоит курсор). Строки нумеруются с нуля, поэтому условие If ComboBox1.SelectedIndex = 0 Then TextBox1.Text = "задача1" означает: если выбрана первая строка из списка, то в текстовое поле поместить условие первой задачи.
Если в следующей задаче требуется меньше входных данных, то лишние текстовые поля ввода скрываются с помощью метода TextBox3.Visible = False, а в статических комментариях просто удаляем текст. Оставляя сам компонент на экране: Label3.Text = "".
Если в последующей задаче больше входных данных, то поля ввода и подписи можно добавить программно.
Аналогично предыдущему проекту 2.1 данного пособия, в Visual Basic так же существует возможность программного добавления изображений из ресурсов и установка свойства растягивания до нужных размеров. Для этого в коде устанавливается атрибут положения изображения внутри компонента (PictureBoxSizeMode.StretchImage):

```
PictureBox1.Image = My.Resources.f16_9
PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
```

Окончательно основная форма с выбором условия задачи имеет вид:



Подобное условие выбора индекса строки записывается в обработчике нажатия на кнопку. Фрагмент кода:

```
If ComboBox1.SelectedIndex = 0 Then
            TextBox2.Text = Replace(TextBox2.Text, ",", ".")
            TextBox3.Text = Replace(TextBox3.Text, ",", ".")
```

```
            Module1.h1 = Val(TextBox2.Text)
            Module1.h2 = Val(TextBox3.Text)
            Label1.Text = "Ответ:  " + Str(Module1.f15_9)
        End If
```

В первых двух строках ошибочно введенная десятичная запятая заменяется на точку, в противном случае Visual Basic при конвертации в числовой формат отбрасывает неверный знак, сохраняя целое число. В строках процедуры 3-4 значения из полей ввода заносятся в переменные соответствующего модуля.
Новый модуль создается с помощью пункта Проект — Добавить модуль в окне редактора Visual Studio. После написания кода модуля в предыдущую процедуру добавляется пятая строка с выводом результата по имени модуля.

Рассмотрим код модуля №1:

```
Module Module1
    Public h1, h2 As Single
    Dim h3 As Single
    Public Function f15_9()
        h3 = 2 * h2 - h1
        f15_9 = (h3)
    End Function
End Module
```

Префиксом Public помечаются переменные, которые участвуют в основной программе в качестве входной информации. Префикс Dim используется при объявлении внутренней переменной. Вычисления производятся внутри функции пользователя. После расчета значение результата присваивается значению функции. Внутри модуля может быть несколько функций, но каждая из них должна возвращать только одно значение.

Полный код программы представлен в электронном приложении

Задача 2.3

Постановка задачи. Решение полиномиальных уравнений, т. е. уравнений от одной переменной в 1, 2, 3 и 4 степени. Построить график функции - левой части данного уравнения.
Внешнее описание: в одном общем окне вводятся в текстовые поля коэффициенты уравнения и выбирается цвет графика.
Функциональная спецификация. С помощью кнопки «Решить» производится расчет корней. По кнопке «Построить» изображается график. Нажатием кнопки «Очистить» значения коэффициентов обнуляются, а поле с графиком очищается.
Система программирования: Lazarus.
Особенности реализации и возникшие трудности: поля ввода приводятся для всех пяти коэффициентов уравнения $ax^4+bx^3+cx^2+dx+e=0$. Если необходимо решить уравнение степени ниже, то коэффициенты при высших степенях приравниваются к нулю. Переменные для входных коэффициентов и выходных результатов объявляются как глобальные. Действия производятся в трех основных процедурах — обработчиках кнопок, а решение уравнений — в четырех вспомогательных процедурах на каждый вид уравнения.

В обработке события «Решить» проверяются условия на равенство коэффициентов нулю, в результате которых вызывается соответствующая процедура:

```
if a<>0 then ferrari(a,b,c,d,e); {уравнение четвертой степени}
 if (a=0) and (b<>0) then cubic(b,c,d,e); {кубическое уравнение}
 if (a=0) and (b=0) and (c<>0) then kvadrur(c,d,e); {квадратное уравнение}
 if (a=0) and (b=0) and (c=0) then lin(d,e); {линейное уравнение}
```

По умолчанию Lazarus (Free Pascal) не поддерживает операции с комплексными числами, поэтому при решении данной задачи либо ограничимся действительными корнями уравнения, либо надо написать специальный класс (или запись). Остановимся на первом варианте для упрощения.

Уравнение четвертой степени решается методом Феррари. После подстановки получается уравнение третьей степени — кубическая резольвента, которое имеет целочисленный корень, являющийся делителем свободного члена. В нашей программе этот корень находится методом подбора среди делителей внутри тела цикла. После этого уравнение распадается на два квадратных уравнения. Для их решения вызывается процедура kvadrur() с нужными параметрами.

Уравнение третьей степени решается методом Кардано. При извлечении квадратных корней по формуле могут получиться комплексные значения, а окончательный результат — действительные корни. Для каждого случая «дискриминанта» записываются готовые формулы расчета, в том числе и для действительных частей комплексной области. Фрагмент кода:

```
if s<0
   then
     begin
       if q<0 then f:=Arctan(-2*Sqrt(-s)/q);
…
```

Во всех процедурах расчета результат выводится в той же процедуре с помощью диалогового окна вывода, например:

```
showmessage('X1='+floattostr(x1));
```

Для уравнения второй степени при отрицательном дискриминанте выводится сообщение «Решений нет». Т.к. эта процедура вызывалась и при решении уравнения четвертой степени, то после проверки условия неравенства первого коэффициента нулю производится дополнительный расчет корней:

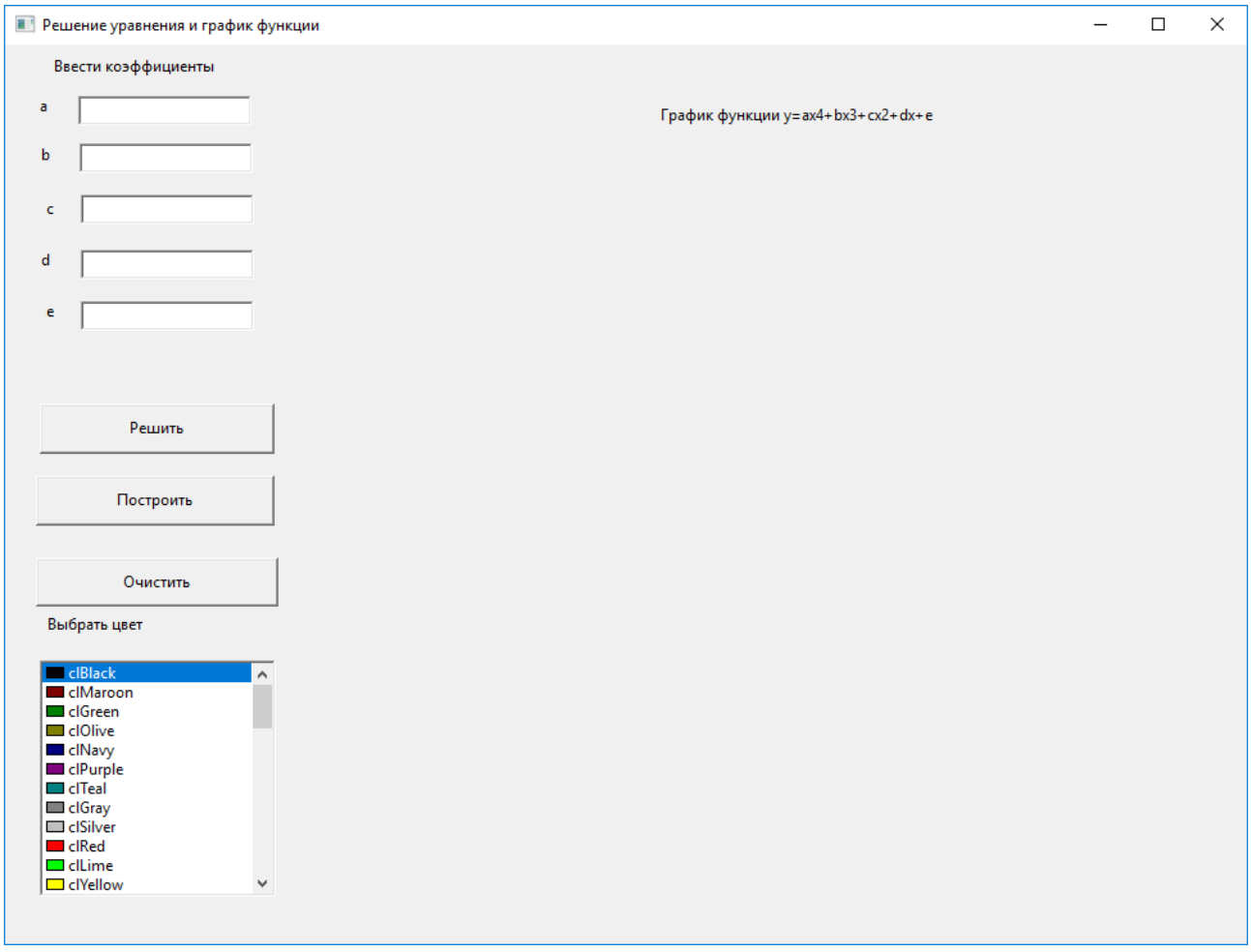
```
if strtofloat(edit1.text)<>0 then begin
x1:=x1-strtofloat(edit2.text)/strtofloat(edit1.text)/4;
x2:=x2-strtofloat(edit2.text)/strtofloat(edit1.text)/4;
```

При решении уравнения первой степени $dx+e=0$ возможны случаи: оба коэффициента равны нулю, только коэффициент при x равен нулю или оба коэффициента имеют ненулевые значения. Для каждого случая приводится свой результат.

При построении графика функции может возникнуть ситуация, когда значения коэффициентов (в пикселях) настолько малы или велики, что график некорректно отображается. Для исправления этого недостатка части функции

делятся или умножаются на некоторые числа для более красивого отображения:
a:=StrToFloat(Edit1.Text)/1000;

Достоинства приложения: простой интуитивный интерфейс.
Недостатки приложения: некорректное решение некоторых видов уравнений, нет поддержки комплексных чисел.
Разработка интерфейса и программная реализация. В приложении использовались следующие компоненты: нередактируемый список цвета (ColorListBox), кнопки (Button), текстовые поля (Edit), статический текст (Label), окно графического вывода (PaintBox).
Список цвета ColorListBox используется для выбора цвета из палитры линий графика. В других системах программирования такой компонент может отсутствовать, т. к. целесообразнее использовать стандартное диалоговое окно ColorDialog (как в проекте 1.1 настоящего пособия).
Окончательно форма задачи имеет вид:



Приведем фрагменты кода с построением графика:

```
with PaintBox1.Canvas do
 begin
 MoveTo(0,260); LineTo(635,260);
…
```

Далее приводятся аналогичные процедуры для вычерчивания линий. Здесь метод PaintBox1.Canvas.MoveTo(x,y) ставит курсор в позицию с заданными координатами, а соответственно .LineTo(x,y) – вычерчивает линию до точки с заданными координатами. В результате получим оси координат со стрелками на концах.

TextOut(320,262, FloatToStr(0)); - изображение цифры «0» в графическом режиме на пересечении осей координат.

```
x1:=-300;
repeat
y1:=a*x1*x1*x1*x1+b*x1*x1*x1+c*x1*x1+d*x1+e;
x:=round(x1);
y:=round(-y1);
Pixels[x+318,y+260]:=ColorListbox1.Selected;
x1:=x1+0.1
until (x1>300);
```

Отрезок изменения оси Ox [-300,300]. Начальное значение присваивается, а конечное рассчитывается в цикле с постусловием, прибавляя шаг 0,1 пикселей. Переменной y1 присваивается значение функции (здесь может стоять любая необходимая функция).

Процедура вычерчивания точки имеет целочисленные параметры, поэтому значения x1 и y1 сначала округляются до целого. Координатная ось графика и сетка изображения не совпадают, поэтому график может отобразиться в перевернутом виде. Во избежание этого функцию заменяем противоположной: -y1. Сам график смещаем на необходимое количество пикселей для центрирования и присваиваем значение выбранного цвета из ColorListbox1.

Полный код программы представлен в электронном приложении

Задача 2.3 а

Изменим интерфейс и процедуры расчета корней уравнения. Пусть график выводится в отдельном окне, а в результате могут присутствовать комплексные корни. Реализуем эту подзадачу с помощью системы программирования Microsoft Visual Studio Python (или приложения IDLE Python 3.6).

Разработка интерфейса: графический пользовательский интерфейс был разработан с помощью библиотеки tkinter. На форме помещаются 5 полей ввода коэффициентов (Entry) с сопровождающим текстом, статическое поле вывода результата (Label) и две кнопки – «Решить» и «Построить». При нажатии на последнюю кнопку открывается дочернее окно:
(win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")), в котором происходит отрисовка графика с помощью метода canvas.

Особенности реализации и возникшие трудности: по аналогии с предыдущей реализацией для решения уравнений используются четыре функции пользователя для уравнения степеней 1, 2, 3 и 4.

Для упрощения решения уравнения четвертой степени вместо метода Феррари применяется метод подбора целочисленных корней среди делителей свободного члена (внутри цикла):

```
k=abs(round(e))
for i in range(-k,k+1):
     if int(a*i**4+b*i**3+c*i**2+d*i+e)==0:
         u=u+1
         text=text+"x%s= %s\n" %(u,i)
```

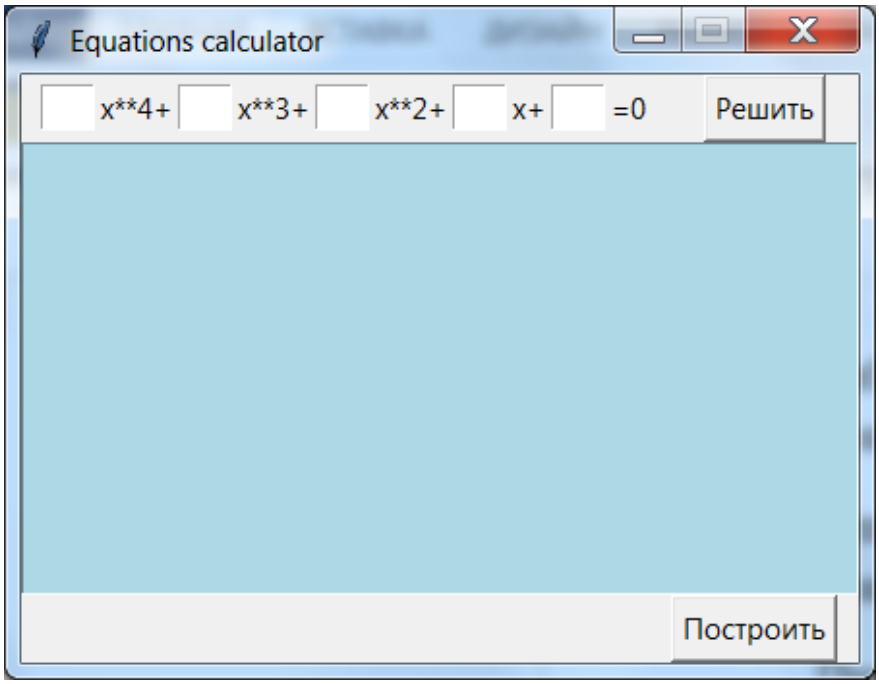
В этом фрагменте *e* – свободный член, *u* – номер корня.

Для работы с комплексными числами в языке Python достаточно подключить библиотеку cmath, поэтому алгоритмы решения кубического и квадратного уравнений упрощены.

Решение линейного уравнения аналогично тому, что было в предыдущей реализации на Lazarus.

При вводе коэффициентов уравнения записывается функция handler(), в которой происходит проверка чисел на соответствие шаблону «0.0». Таким образом, функция расчета «оборачивается» в функцию соответствия параметров inserter(). В случае несовпадения в текстовое поле выводится сообщение о неправильном вводе.

Окончательно основная форма решения задачи имеет вид:



Приведем фрагмент кода с построением графика:

```
def click_button():
    a_val = float(a.get())
    …
    def fu(x):
        y=a_val*x**4/1000+b_val*x**3/100+c_val*x**2/10+d_val*x+e_val*10
        return y
    win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")
    win.title("График")
    win.minsize(width=400,height=200)
    points=[]
```

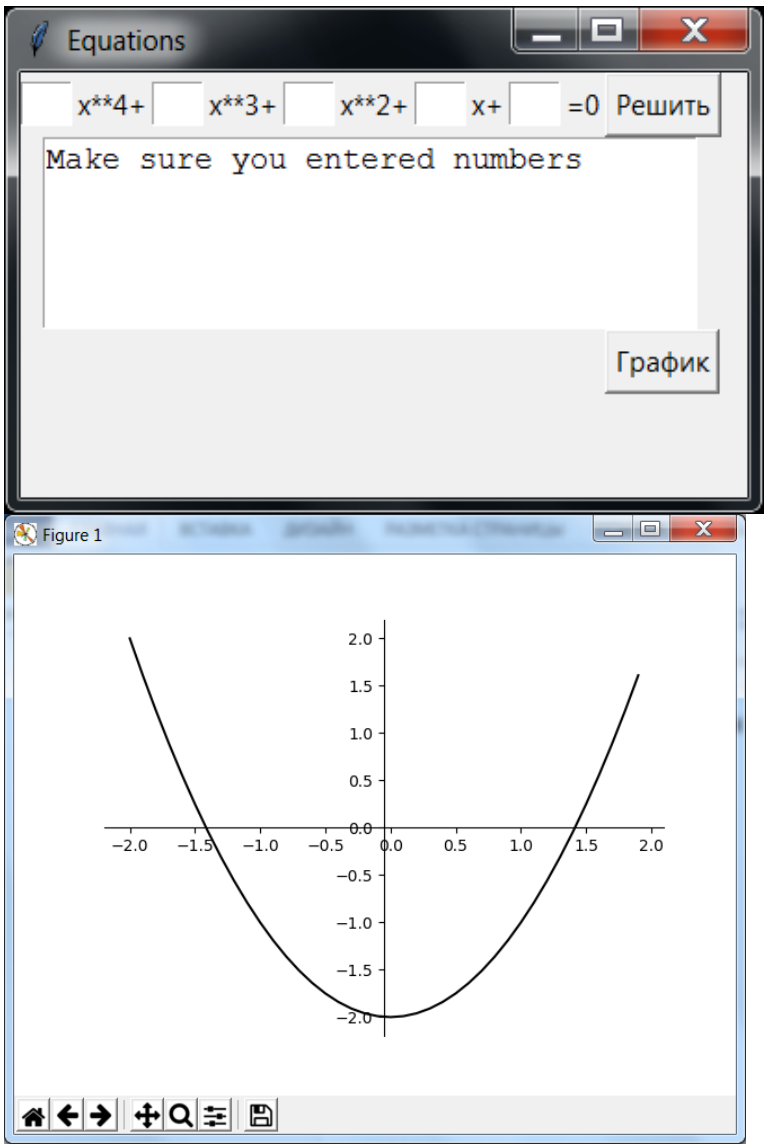
```
    for x in range(-480,480):
        y=int(fu(x))
        x=x+180
        y=240-y
        pp=(x,y)
        points.append(pp)
    canvas=Canvas(win)
    canvas["height"]=360
    canvas["width"]=480
    canvas["background"]="#eeeeff"
    canvas["borderwidth"]=2
    canvas.pack({"side":"bottom"})
    canvas.create_line(points,fill="blue",smooth=1, width=2)
    canvas.create_line(180,360,180,0,width=2,arrow=LAST) #ось y
    canvas.create_line(0,240,480,240,width=2,arrow=LAST) #ось x
```

1) Функция обработки кнопки «Построить».
2) Связь значений из полей ввода с числовыми переменными. Эти строки не совсем корректны, т.к. в предыдущей функции handler() уже использовался ввод переменных с их проверкой. В этом фрагменте проверка не осуществляется. Целесообразно оформить эти действия в общий класс, но в этом коде ограничимся дублированием.
3) Отдельная функция *y=fu(x)*, которую можно редактировать внутри кода.
4) Инициализация нового окна и установка свойств канвы рисования.
5) Создание массива точек и построение графика по этим точкам.
6) Вычерчивание осей координат.

Полный код программы представлен в электронном приложении

Задача 2.3 б

Изменим код задачи, используя понятие класса. Для построения графика воспользуемся специальной библиотекой matplotlib, позволяющий выводить графики функций в отдельном окне, используя синтаксис системы компьютерной математики Matlab.
Для этого необходимо:
1) Добавить библиотеку: import matplotlib.pyplot as plt
2) Описать класс: class Application(Frame)
3) Инициализировать основную функцию класса __init__(self,master).
4) Добавить функцию super(), которая позволяет ссылаться на родительский класс без явного присвоения имен.
5) Описать основные функции класса, влияющие на реализацию графического интерфейса.
6) Функция создания интерфейса create_widgets(self): подписи, текстовые поля ввода, кнопки, выровненные по сетке; текстовая область вывода.
7) Основная рабочая функция для связывания числовых переменных со значениями из текстовых полей, вызов функций решения уравнений.
8) Функция построения графика средствами библиотеки matplotlib.pyplot.
Окна формы:



Полный код программы представлен в электронном приложении

Задача 2.4

Постановка задачи. Калькулятор, который выполняет основные арифметические действия.

Внешнее описание: для простоты введем два текстовых поля для ввода чисел (аргументов), поле вывода результата и кнопки для операций.

Функциональная спецификация. С помощью этой программы выполняются следующие действия: сложение, умножение, вычитание, деление, целочисленное деление, возведение в степень и остаток от деления.

Вычисления для каждой из операции производятся в отдельной процедуре – обработчике события нажатия на кнопку. Данные в текстовые поля вводятся с клавиатуры.

Система программирования: Python.

Особенности реализации и возникшие трудности: для каждой из операций необходимо определить тип аргументов (целочисленный или вещественный) и тип исключений (например, деление на ноль).

Приведем процедуру для сложения двух чисел:

```
def btn1_click():
    try:
        a1 = float(v1.get())
        a2 = float(v2.get())
        cash = a1+a2
        v3.set(cash)
    except:
        v3.set ("Должны быть введены числа")
```

В блоке try производится попытка преобразования значения из текстового поля в числовой тип, подсчет суммы и вывод его значения в текстовый блок результата. В случае неудачи выполнения этих операций (блок except) в поле результата должно появиться соответствующее сообщение.

Для некоторых операций все аргументы или только второй (в случае возведения в степень) должны быть целочисленными. Проверка правильности ввода регулируется аналогичным блоком исключения.

Рассмотрим на примере нахождения остатка от деления:

```
def btn7_click():
    try:
        a1 = int(v1.get())
        a2 = int(v2.get())
        cash = a1%a2
        v3.set(cash)
    except ZeroDivisionError:
        v3.set("Деление на ноль, попробуйте ещё ")
    except Exception:
        v3.set("Должны быть введены целые числа ")
```

В этой процедуре два исключения – деление на ноль и неправильный ввод чисел.

Достоинства приложения: простота реализации.

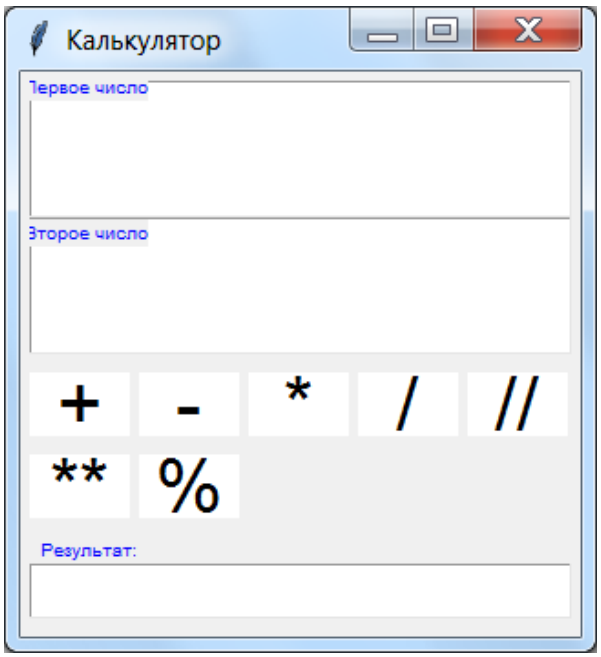
Недостатки приложения: перегруженность текстовыми блоками, не все функции реализованы.

Разработка интерфейса и программная реализация. В приложении использовались следующие компоненты: кнопки (Button), текстовые поля (Entry), статический текст (Label). Для реализации графического интерфейса применяется библиотека tkinter с вышеперечисленными виджетами.

Порядок следования блоков программы:

1) Установка свойств (размера и имени) формы.
2) Размещение текстовых полей для ввода чисел.
3) Семь функций расчета – обработчиков нажатия на кнопку с выбором операции. После каждой процедуры размещается соответствующая кнопка со знаком операции.
4) Размещение текстового поля для вывода результата.
5) Отображение формы с виджетами на экране.

Окончательно основная форма решения задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.5

Постановка задачи. Создать аналог классического калькулятора Windows. В классическом Калькуляторе существует только одно текстовое поле – для ввода каждого из аргументов и для вывода результата вычислений. Данные вводятся двумя способами – как при нажатии на кнопку с соответствующей цифрой, так и с помощью клавиатуры.

Внешнее описание: обработка нажатия каждой из 24 кнопок с цифрами, знаками арифметических операций и математических функций, удаление информации из текстового поля, отображение знака операции, выбор меры угла для тригонометрических функций.

Функциональная спецификация. С помощью этой программы выполняются следующие действия: сложение, умножение, вычитание, деление, возведение в степень, а также следующие математические функции: синус, косинус, тангенс, натуральный логарифм, квадратный корень из числа, модуль числа и обратная величина числа (*1/x*). Обработка каждого действия реализуется в отдельной процедуре – обработчике события нажатия на кнопку.

Вычисления для каждой из операций производятся в общей процедуре calculate() с помощью оператора-переключателя switch.

Система программирования: Microsoft Visual Studio C#.

Особенности реализации и возникшие трудности: часть переменных (операнды, знак числа и номер выбора функции) лучше описать как глобальные. В языке C# они помещаются в раздел описания класса Form1. Если добавить переменную строкового типа (например, string s) для работы с текстовым полем, то часть действий может осуществляться некорректно. Лучше оставить непосредственное присваивание для textBox1.Text.

Например, код для обработки вычислений по математической функции имеет вид: функция(Convert.ToString(textBox1.Text)).
Если в текстовом поле содержимое пусто, то при нажатии на кнопки с расчетом функций или операций (например, «+») программа некорректно завершает работу. Существует 3 способа решения этой проблемы:
1) Защита в форме окна сообщения, например, MessageBox.Show(“Введите число”).
2) Поставить значение в текстовом поле по умолчанию «0». Тогда в процедурах обработки кнопок с числами (1,2,…) лишний ноль должен стираться.
3) В процедурах обработки кнопок с функциями и операциями определить условие: если в текстовом поле пусто, то добавить значение 0», например: if (textBox1.Text==””) textBox1.Text=”0”.
Во всех трех случаях фрагменты кода с проверкой будут повторяться для каждой из процедур.
Т.к. изначально в текстовом поле нашего приложения значение пусто, то необходимо реализовать корректный обработчик нажатия на кнопку с функцией. Добавим условие, реализованное в третьем пункте, когда отсутствующее значение приравнивается к нулю:

```
if (textBox1.Text == "") a = 0;
```

Отдельно необходимо выполнить смену значения логической переменной (true/false) для знака числа – положительное или отрицательное.
Допустимые значения и исключения:
1) При делении числа на ноль или значения логарифма при нулевом или отрицательном значении аргумента нет дополнительного условия обработки ошибки, т.к. C# в этом случае возвращает в качестве результата слово «бесконечность».
2) При вычислении квадратного корня из отрицательного числа в поле результата появляется сообщение «Нельзя вычислить корень».
3) Исключения можно реализовать другим способом, например, для функции $y=1/x$ при нулевом значении аргумента выводится диалоговое окно с сообщением «Введите число, не равное нулю».
Для нанесения знака операции в качестве текста на кнопке, который нельзя ввести с клавиатуры, можно применить один из двух способов:
1) изобразить в графическом редакторе и выбрать в свойствах кнопки изменение фонового рисунка, например: button1.BackgroundImage…
2) найти нужный символ в Таблице символов Windows и скопировать символ или указать его код.

Достоинства приложения: альтернатива стандартному калькулятору в режиме «Инженерный».
Недостатки приложения: не все функции реализованы. Нет поддержки добавления в память и сохранения результата.

Разработка интерфейса и программная реализация. В приложении использовались следующие компоненты: кнопки (Button), текстовое поле (textBox), статический текст для отображения знака операции (Label), переключатели (RadioButton) для изменения меры угла.
Все компоненты размещены на одной форме, причем, компонент для надписи Label помещается сверху на компонент ввода значения textbox.

Десять кнопок с цифрами от 0 до 9 имеют следующий код обработчика (показан на примере кнопки с цифрой «1»):

```
private void button1_Click(object sender, EventArgs e)
        {
           textBox1.Text += 1;
        }
```

В текстовое поле добавляется значение с цифрой всякий раз, когда нажата соответствующая кнопка.
В обработчике кнопки знака «=» записано две команды: вызов функции calculate() с реализацией бинарных арифметических операций (с двумя операндами) и стирание промежуточного значения с выбором знака из статического поля.
Четыре кнопки со знаками (плюс, минус, умножить, разделить) обрабатываются по следующей схеме:

```
private void button11_Click(object sender, EventArgs e)
        {if (textBox1.Text == "") a = 0;
        else
            a = float.Parse(textBox1.Text);

            textBox1.Clear();
            count = 1;
            label1.Text = a.ToString() + "+";
            znak = true;
        }
```

1) Если в текстовом поле написано число, то его необходимо преобразовать из строкового типа в вещественный и присвоить переменной первого аргумента. Если пустое значение, то первый аргумент сделать равным нулю.
2) Очистить текстовое поле для подготовки ввода второго аргумента.
3) Указать номер операции для переключателя в функции calculate().
4) Поместить в статическое поле значение первого аргумента и знак операции.
5) Определить логическое значение «истина» для знака числа.
Кнопка «C» - очистить всё. Удаляются все данные из полей ввода и вывода.
Кнопка «<-» - очистить один символ. Очищение символов по одному происходит внутри цикла. Сначала удаляются все значения из текстового поля, а потом добавляются по одному все предыдущие значения, кроме последнего, из заранее сохраненной текстовой переменной.
Кнопка «,» - десятичная запятая. С помощью условия if (textBox1.Text.IndexOf(',') == -1) десятичная запятая добавляется к числу только один раз.

Кнопка «+/-» - знак числа. При нажатии добавляет или стирает «-» перед числом, меняет значение логической переменной для дальнейшей обработки. Например, операция сложения *a+ (-b)* равносильна операции вычитания *a-b*.
Кнопки функций реализуют вычисление соответствующей функции. Для тригонометрических функций (sin, cos, tan) используется переключатель между радианной и градусной мерами угла.
1) Градусы:

```
if (radioButton2.Checked)
            {
                if (textBox1.Text == "") textBox1.Text = "0";
                else
                {
           double result = Math.Sin(Convert.ToDouble(textBox1.Text) * Math.PI / 180);
                    textBox1.Text = result.ToString();
                }
}
```

2) Радианы:

```
if (radioButton1.Checked)
            {
                if (textBox1.Text == "") textBox1.Text = "0";
                else
                {
                    double result = Math.Sin(Convert.ToDouble(textBox1.Text));
                    textBox1.Text = result.ToString();
                }
            }
```

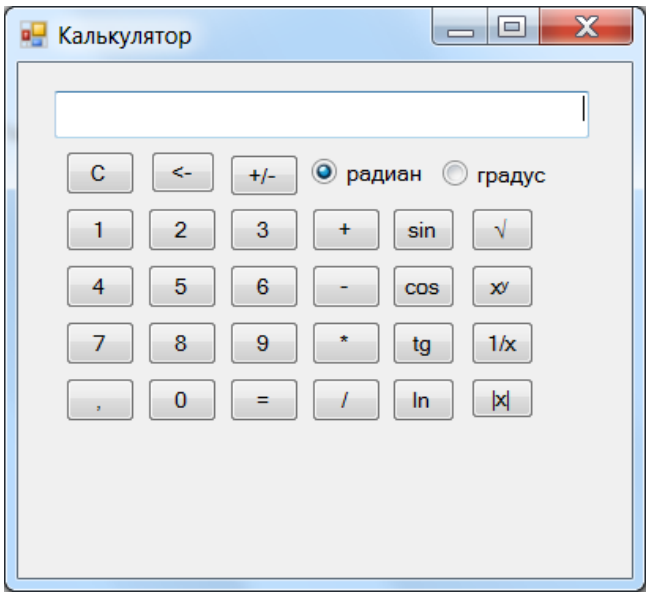
Эти блоки реализованы без учитывания особенностей языка C# и могут использоваться и в других системах программирования.
Фрагмент функции calculate() для реализации сложения двух чисел:

```
private void calculate()
        {
            switch (count)
            {
                case 1:
                    b = a + float.Parse(textBox1.Text);
                    textBox1.Text = b.ToString();
                    break;
…
```

Значение первого аргумента *a* берется из соответствующей процедуры-обработчика знака операции. Второй аргумент конвертируется в число из повторно заполненного текстового поля. Результат помещается в переменную *b*, значение которой сразу выводится в то же текстовое поле.
В языке C# функция подсчета степени числа Pow из математической библиотеки Math имеет тип double. Для работы над данными других типов используется явное приведение типов, т. е.:

```
b =  (float) Math.Pow((double) a, double.Parse(textBox1.Text));
```

Окончательно основная форма решения задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.5 а

Постановка задачи. Простой калькулятор с памятью и озвучкой кнопок.
Внешнее описание. Изменим код предыдущего приложения, используя классы. Ввод данных и вывод результата осуществляется только с помощью кнопок в неизменяемом статическом поле.
Функциональная спецификация. С помощью этой программы выполняются следующие действия: сложение, вычитание, умножение, деление, квадратный корень и корень любой степени, возведение числа в квадрат и в любую степень, факториал, операции с памятью.
Для краткости функции, не возвращающие значение, назовем процедурами, а возвращающие – просто функциями. Функции, записанные внутри класса, назовем методами.
Система программирования: Microsoft Visual C#.
Особенности реализации и возникшие трудности: сначала создается интерфейс класса InterfaceCalc.cs в пространстве имен calculator. В этом классе перечисляются все необходимые функции (методы), причем для операций с двумя операндами a и b первый из них запоминается в процедуре void Put_A(double a), а второй является аргументом соответствующей функции, например, double Sum(double b) – суммирование двух чисел. Функции с одним реальным аргументом записываются без формальных параметров, например, double Sqrt() – квадратный корень из числа. В описании также добавлены функции работы с памятью:

double MemoryShow(); - показать содержимое регистра памяти
void Memory_Clear(); - стереть содержимое регистра памяти
void M_Multiplication(double b); - умножение
void M_Division(double b); - деление
void M_Sum(double b); - сложение
void M_Subtraction(double b); - вычитание

Затем создается класс Calc, реализующий интерфейс InterfaceCalc:

```
    public class Calc : InterfaceCalc
```

В этом классе определяется переменная класса *a* (глобальная переменная с областью видимости для всех функций класса и только для них):

```
private double a = 0;
```

В методе void Put_A(double a) запоминается значение этой переменной:

```
public void Put_A(double a)
        {
            this.a = a;
        }
```

В остальных методах класса производится непосредственная работа с этой переменной, например, очистка. Арифметические операции, математические функции и т.д. Для сложения чисел соответствующий метод имеет вид:

```
public double Sum(double b)
        {
            return a + b;
        }
```

Для операций с памятью к переменным класса добавляется переменная memory, которая аналогичным образом используется в данных методах. Например:

```
//показать содержимое регистра памяти
        public double MemoryShow()
        {
            return memory;
        }

        //стереть содержимое регистра памяти
        public void Memory_Clear()
        {
            memory = 0.0;
        }
```

Наконец, создается форма и класс формы, в файл которого записывается непосредственный код программы:

```
public partial class Form1 : Form
    {
        Calc C;
…
// описываются дополнительные переменные класса и компоненты
public Form1()        {
C = new Calc();
...
```

По аналогии с предыдущей реализацией калькулятора записываются функции смены знака числа, ввода цифр и десятичной запятой с помощью кнопок, очистка текстового поля, например:

```
//кнопка изменения знака у числа
        private void buttonChangeSign_Click(object sender, EventArgs e)
        {
            if (labelNumber.Text[0] == '-')
                labelNumber.Text = labelNumber.Text.Remove(0, 1);
            else
                labelNumber.Text = "-" + labelNumber.Text;
        }
```

Операнды вводятся только нажатием кнопок (не с клавиатуры) в статическое поле Label, поэтому в указанной процедуре используется такое условие: если

первым символом текста в поле является знак «-», то этот символ стирается, в противном случае - знак «-» добавляется.
Процедура кнопки с вводом цифры, например, цифры «1»:

```
private void button1_Click(object sender, EventArgs e)
        {
            labelNumber.Text += "1";

            CorrectNumber();
            sp1.Play();
        }
```

После добавления к тексту очередной цифры вызывается функция проверки цифры на правильность ввода и проигрывается звук ввода единицы из файла, заранее записанного в ресурсах, и инициализированного в компонентах.
Приведем фрагмент процедуры CorrectNumber(), проверяющей правильность вводимого значения:

```
//если слева от положительного числа ноль, а после него не запятая, тогда ноль можно
удалить
            if (labelNumber.Text[0] == '0' && (labelNumber.Text.IndexOf(",") != 1))
                labelNumber.Text = labelNumber.Text.Remove(0, 1);

...
```

В обработчике кнопок с операциями для двух операндов используется следующий код (на примере сложения):

```
private void buttonPlus_Click(object sender, EventArgs e)
        {
            if (CanPress())
            {
                C.Put_A(Convert.ToDouble(labelNumber.Text));

                buttonPlus.Enabled = false;

                labelNumber.Text = "0";
            }
        }
```

Если эта кнопка нажата, то значение из текстового поля сохраняется в переменной, кнопка становится неактивной, а значение поля обнуляется для ввода следующего.
В обработчике кнопки «равно» собираются вызовы непосредственных функций для расчета. При этом значение первой переменной обнуляется, все кнопки становятся доступными для нажатия (при вызове процедуры FreeButtons()) и обнуляется количество нажатий на кнопку вызова из памяти.
Покажем фрагмент обработчика на примере деления, в котором накладывается дополнительное условие – звук со словами «на ноль делить нельзя» при недопустимости данной операции:

```
if (!buttonDiv.Enabled)
            {if (Convert.ToDouble(labelNumber.Text) != 0) labelNumber.Text =
C.Division(Convert.ToDouble(labelNumber.Text)).ToString();
                else spz.Play();
            }
```

Среди методов с реализацией остальных математических функций рассмотрим метод обращения к памяти:

```
//кнопка MRC
        private void buttonMRC_Click(object sender, EventArgs e)
        {
            if (CanPress())
```

```
        {
            k++;

            if (k == 1)
                labelNumber.Text = C.MemoryShow().ToString();

            if (k == 2)
            {
                C.Memory_Clear();
                labelNumber.Text = "0";

                k = 0;
            }
        }
    }
```

Если на кнопку «MRC» нажали один раз (k=1), то выводится в текстовое поле значение из памяти. Если нажали два раза (k=2), то происходит очистка переменной памяти.

Фрагмент логической функции, которая проверяет не нажата ли еще какая-либо из кнопок математических операций, имеет вид:

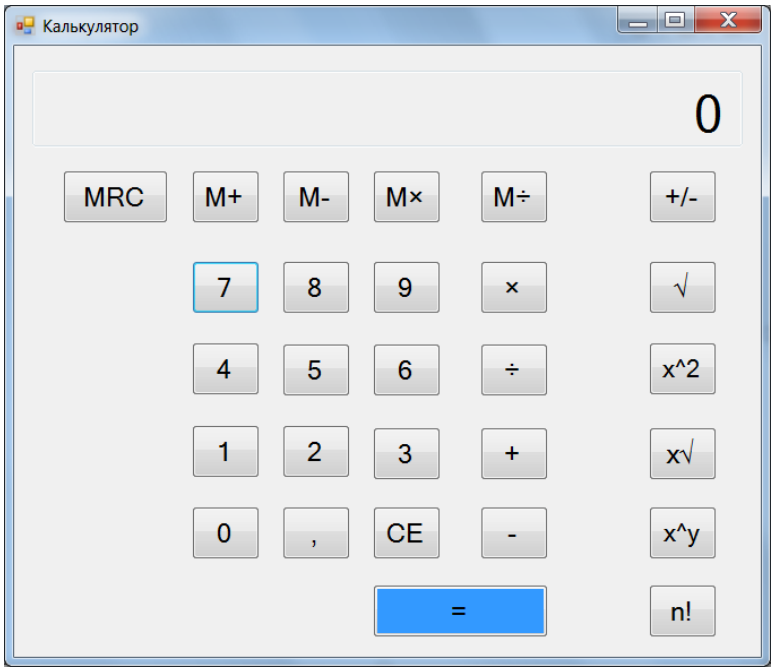
```
    private bool CanPress()
    {
        if (!buttonMult.Enabled)
            return false;
```

…

Фрагмент функции снятия нажатия всех кнопок операций:

```
    private void FreeButtons()
    {
        buttonMult.Enabled = true;
```

Окончательно основная форма решения задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.5 б

Постановка задачи. Модифицируем предыдущий калькулятор. Для этого добавим пункты меню, с помощью которых:
а) в отдельных окнах будут простая и расширенная версии;
б) появится возможность копирования и вставки в текстовом поле;
в) в отдельном окне появится режим построения графиков функций.
Внешнее описание: обработка нажатия каждой из кнопок с цифрами, знаками арифметических операций и математических функций (в расширенной версии кнопки и их код дублируются), копирование информации из текстового поля. В отличии от предыдущего примера не отображается знак операции, нет выбора меры угла для тригонометрических функций (только радианы).
Функциональная спецификация. С помощью этой программы выполняются следующие действия:
Обычный режим: сложение, умножение, вычитание, деление, процент числа, добавление трех нулей к числу при вводе.
Расширенный режим: возведение в степень (квадрат, куб и произвольную), синус, косинус, тангенс, натуральный и десятичный логарифм, квадратный корень из числа (и корень *n*-ой степени из числа), модуль числа и 10^x), экспонента, факториал числа, вывод в текстовое поле констант π и *e*.
Обработка каждого действия реализуется в отдельной процедуре – обработчике события нажатия на кнопку. Вычисления для каждой из операций производятся в процедуре кнопки со знаком равенства с помощью условного оператора.
Режим графиков: выбор количества графиков на экране (от 1 до 3), выбор функции из каждой категории (степенная функция, линейная с модулем, тригонометрические функции и логарифм). Настройка опций графика: показ осей, масштаб, сетка, надписи, сохранение графика в графический файл (изменяется соответствующий пункт меню).
Система программирования: Microsoft Visual Studio Basic.
Особенности реализации и возникшие трудности: кодирование обработчика кнопок калькулятора во многом повторяет предыдущий пример за некоторым исключением. Во-первых, команда изменения знака числа реализуется не через логическую переменную, а простым умножением на (-1) хранящегося в текстовом поле числа. Во-вторых, представлено несколько вариаций возведения в степень и извлечения корня числа. Факториал числа реализован в виде умножения чисел в цикле. Процент от числа вычисляется по формуле arg1 = arg1 * (arg2 * 0.01).
В программу добавлена функция, позволяющая вводить с клавиатуры только цифры (исключая остальные символы):

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
        If Asc(e.KeyChar) <> 8 Then
            If Asc(e.KeyChar) >= 48 And Asc(e.KeyChar) <= 57 Then
            Else
                e.Handled = True
            End If
        End If
```

```
    End Sub
```
В режиме графиков обнаружен один недостаток – при подстановке в качестве показателя степени $n \geq 4$ программа выдает ошибку.
Допустимые значения и исключения:
1) При значении логарифма при нулевом или отрицательном значении аргумента нет дополнительного условия обработки ошибки, т.к. VB в этом случае возвращает в качестве результата слово «-бесконечность».
2) При делении на ноль происходит проверка условия и вывод в текстовое поле ошибки «ERR».
Достоинства приложения: альтернатива стандартному калькулятору.
Недостатки приложения: не все функции реализованы, дублирование части формы и кода.
Разработка интерфейса и программная реализация. В приложении использовались следующие компоненты: кнопки (Button), текстовое поле (textBox), пункты меню (MenuStrip), открывающийся список с выбором типа графика и масштаба (ComboBox), флажок для выбора настроек (CheckBox), графическое окно для вывода графика (PictureBox).
Все компоненты размещены на трех формах.
11 кнопок с цифрами от 0 до 9 и «000» имеют следующий код обработчика (показан на примере кнопки с цифрой «1»):
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
        TextBox1.Text = TextBox1.Text & "1"
    End Sub
```
В текстовое поле добавляется значение с цифрой всякий раз, когда нажата соответствующая кнопка.
В обработчике кнопки знака «=» записан код с реализацией бинарных арифметических операций (с двумя операндами). Выбор действия зависит от значения глобальной переменной work:
0 – очистка текстового поля;
1 – сложение;
2 – вычитание;
3 – умножение;
4 – деление;
5 – процент числа.
В обработчиках нажатия на кнопки со знаками операций или функций указывается дополнительное условие, проверяющее, что текстовое поле не пусто. Например, для знака «плюс»:
```
Private Sub ButtonPlus_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles ButtonPlus.Click
        If TextBox1.Text <> "" Then
            arg1 = TextBox1.Text
            work = 1
            TextBox1.Text = ""
            TextBox1.Focus()
        End If
    End Sub
```

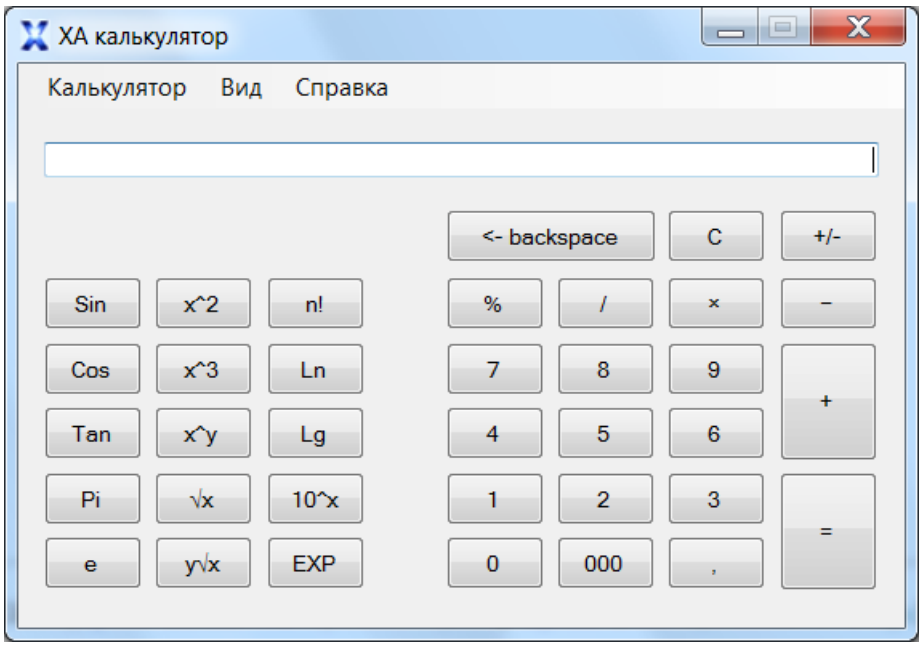
После сохранения в переменную значения первого аргумента текстовое поле очищается, и в это поле устанавливается курсор для дальнейшего ввода второго аргумента.
Для рисования графиков функций используется класс поверхности рисования System.Drawing.Graphics.
В обработчике нажатия на кнопку происходит следующее:
1) установка кисти для рисования (перо) и шрифта для надписей;
2) выбор значения переменной для масштабирования;
3) создание надписи в графическом режиме;
4) переворачивание оси для связи виртуальных координат с декартовыми;
5) рисование осей;
6) рисование сетки (линии в цикле);
7) изображение каждого вида графика, например:

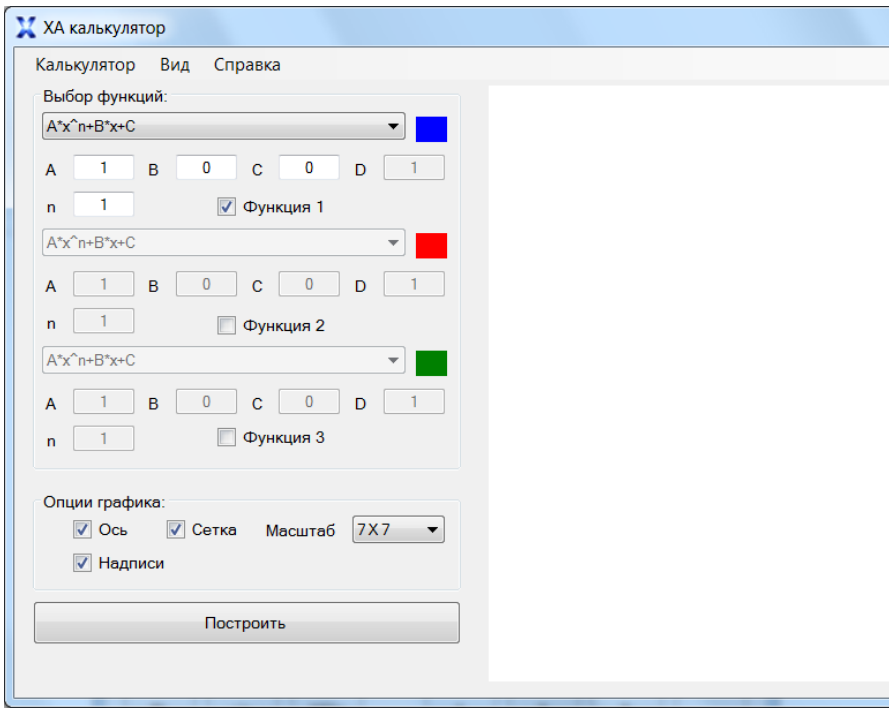
```
If FX = "A*x^n+B*x+C" Then

                For i = -XP To XP Step k
                    x = i
                    y = A * x ^ n + B * x + C
                    g.DrawEllipse(pero, x * m, y * m, 1, 1)
                Next i
```

Затем код дублируется для каждой из трех выбранной функции в ComboBox.
Кроме того, используются процедуры установки начальных значений (FormLoad), выбора функции и настроек (SelectedIndexChanged).
Также в программе реализован пункт меню «Справка». Для открытия файла помощи используется одна команда:

```
Process.Start("ReadMe.htm")
```

Для информации «О программе» открывается дополнительная форма.
Окончательно основная форма решения задачи имеет вид:

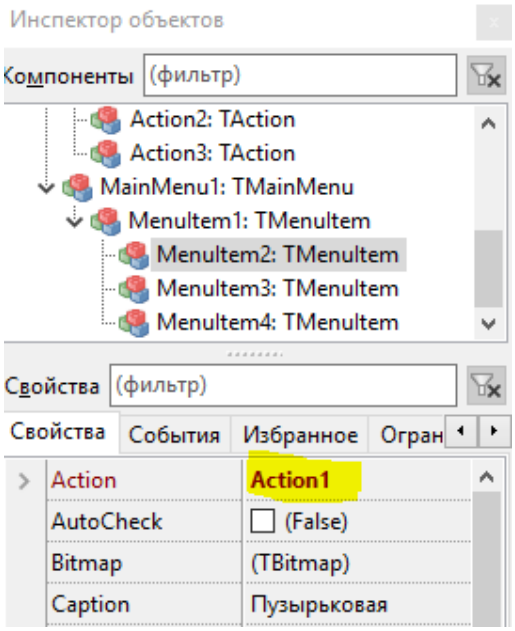




Полный код программы представлен в электронном приложении

Задача 2.6

Постановка задачи. Рассмотреть различные алгоритмы сортировки одномерных массивов.
Внешнее описание: исходный массив вводится в первую строку компонента StringGrid, метод сортировки выбирается либо с помощью меню, либо нажатием на радио-кнопку, отсортированный массив выводится во вторую строку того же элемента StringGrid.
Функциональная спецификация: в качестве примера приводятся алгоритмы трех методов сортировок: пузырьковой, выбором и вставками.
Система программирования: Lazarus
Особенности реализации и возникшие трудности: При обработчике события выбора пункта меню и нажатия на кнопку RadioButton содержимое процедур должно дублироваться. В среде Lazarus существует невизуальный компонент TActionList, который применяют, когда одна и та же команда должна выполняться различными действиями пользователя (например, из меню и кнопкой). В этом случае создается один обработчик, а запускается он от нескольких разных методов. В обработчике вызова метода содержится одна команда, например:
if radiogroup1.ItemIndex=0 then action1.Execute;
Во многих случаях процедуру Action1 можно указать прямо в свойствах элемента управления (Инспекторе объектов):



Для работы приложения используется одномерный динамический массив, задаваемый в разделе глобальных переменных. В этом же разделе задается целочисленная переменная — размер массива:

```
var
  Form1: TForm1;
   arr:array of real;
   m:integer;
```

В процедуре обработки кнопки выбора длины массива используется диалоговое окно InputBox и соотнесение полученного результата с длиной массива и размером компонента StringGrid:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  m:=strtoint(inputbox('m=','m=','0'));
 stringgrid1.ColCount:=m;
 SetLength(arr, m);
end;
```

Достоинства приложения: лаконичный дизайн и не избыточный код.

Недостатки приложения: необходимость вручную прописывать количество элементов массива, в случае отсутствия или ошибочного ввода данных программа выдает системную ошибку и закрывается.

Чтобы избавиться от первого недостатка вместо кнопки с обработчиком события (ввод длины массива) используется процедура обработки введенного текста:

```
procedure TForm1.StringGrid1SetEditText(Sender: TObject; ACol, ARow: Integer; const Value:
string);
begin
  if ACol = StringGrid1.ColCount - 1 then
    StringGrid1.ColCount:= StringGrid1.ColCount + 1;
  m:=stringgrid1.colcount-1;
 SetLength(arr, m);
end;
```

В этой процедуре при вводе каждого значения количество столбцов увеличивается на единицу. Т.к. в последнем столбце значение не нужно, то длина массива на единицу меньше количества всех столбцов.
Для избавления от второго недостатка достаточно в команде присваивания элементам массива значений из StringGrid вместо
arr[i]:=StrToFloat(StringGrid1.Cells[i-1,0]);
указать: TryStrToFloat(StringGrid1.Cells[i-1,0],arr[i]);

Разработка интерфейса. В приложении использовались следующие компоненты: меню (MainMenu), контейнер для действий (ActionList), кнопка (Button), сетка таблицы для ввода/вывода массива (StringGrid), группа переключателей (RadioGroup). В результате модификации программы кнопку можно удалить с формы.

Рассмотрим алгоритмы сортировки одномерного массива:
1) Пузырьковая сортировка.
Алгоритм сортировки простыми обменами (пузырьком) состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются *N-1* раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.
Фрагмент кода:

```
for i := 1 to m-1 do
      for j := 1 to m-i do
         if arr[j] > arr[j+1] then begin
            k := arr[j];
            arr[j] := arr[j+1];
            arr[j+1] := k
         end;
```

2) Сортировка выбором.
Алгоритм состоит из следующих шагов: найти номер минимального значения в текущем массиве; произвести обмен этого значения со значением первой не отсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции); сортировать оставшиеся элементы массива, исключив из рассмотрения уже отсортированные элементы.
Фрагмент кода:

```
for i := 1 to m - 1 do begin
     nmax := i;
     for j := i + 1 to m do
       if arr[j] < arr[nmax] then nmax := j;
     buf := arr[i];
     arr[i]:= arr[nmax];
     arr[nmax]:= buf;
```

```
    end;
```

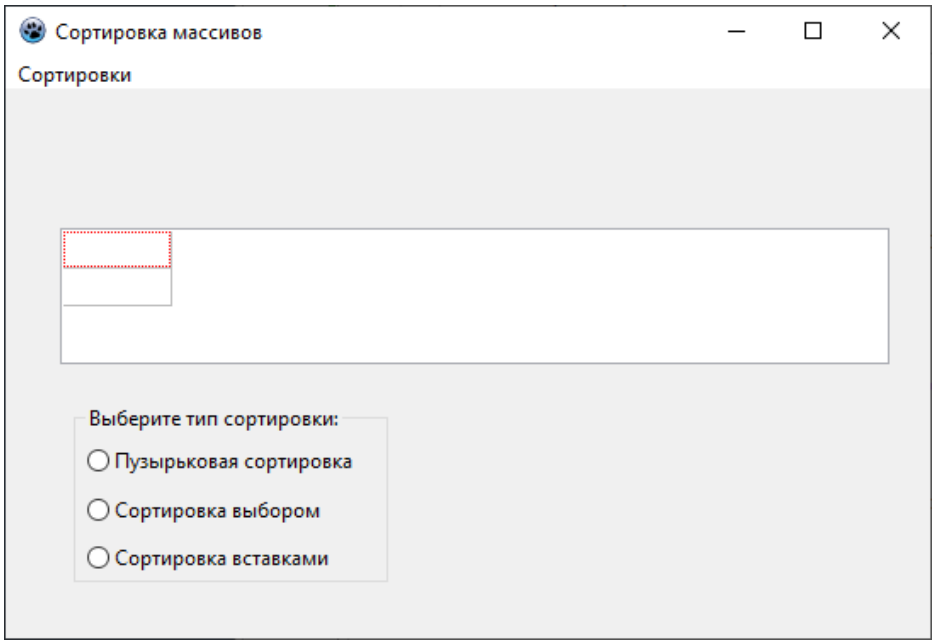
3) Сортировка вставками.

Алгоритм, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов до тех пор, пока набор входных данных не будет исчерпан.

Фрагмент кода:

```
for i:=2 to m do
 begin
 buf:=arr[i];
j:=i-1;
while (j>=1) and (arr[j]>buf) do
begin
arr[j+1]:=arr[j];
j:=j-1;
end;
 arr[j+1]:=buf
 end;
```

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.6 а

Постановка задачи. Рассмотреть различные алгоритмы сортировки одномерных массивов с текстовым описанием метода сортировки.

Внешнее описание: выбор метода сортировки осуществляется с помощью раскрывающегося дерева, в текстовом окне отображается описание алгоритма, в таблице DataGridView вводятся элементы массива в количестве десяти. Расчет производится в отдельной процедуре, а результат собирается и выводится в текстовом виде по нажатию на кнопку.

Функциональная спецификация: в качестве примера приводятся алгоритмы десяти методов сортировок: пузырьковая, шейкерная, гномья, вставками, слиянием, выбором, Шелла, пирамидальная, быстрая (Хоара) и подсчетом.
Система программирования: Microsoft Visual Basic.
Особенности реализации и возникшие трудности: в системе программирования VB (и других системах, основанных на технологии .NET) существует множество функций для работы с массивами: определение нижней и верхней границ массива, нахождение длины массива, поиск максимального и минимального элементов, поэтому можно сосредоточиться только на непосредственных алгоритмах сортировки, исключив вспомогательные действия.
Каждая из десяти процедур сортировок построена по следующему принципу (на примере пузырьковой):

```
Sub Bubble(ByRef a() As Integer)
```

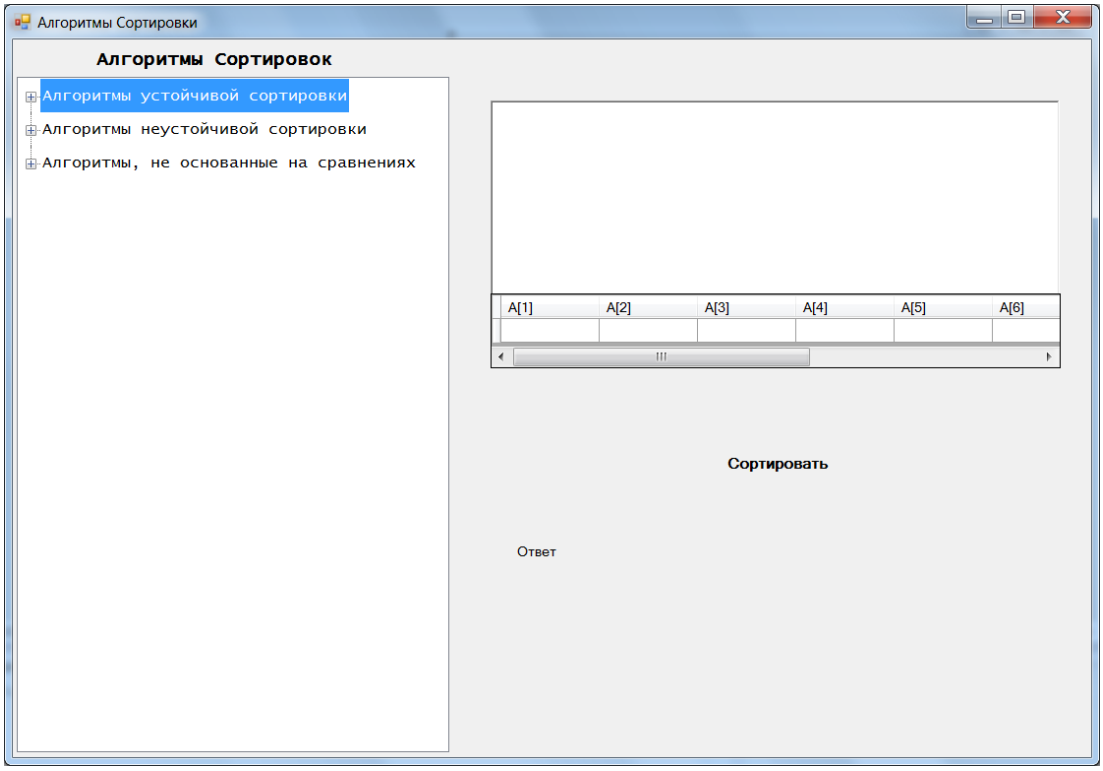
Параметр-массив передается по ссылке. В начале программы (в разделе переменных класса формы) массив объявляется как статический (с конечным числом элементов), но VB в любом случае работает с динамическим массивом, переданным в качестве параметра процедуры.
В процедуре PrintArr() осуществляется вывод отсортированного массива в строку статического текста и обнуление числового массива.
В процедуре SortA() осуществляется выбор алгоритма сортировки в зависимости от переменной, сопоставленной соответствующему узлу дерева. После чего осуществляется вызов предыдущей процедуры вывода результата.
Дерево выбора заполняется программно. Сначала записывается код процедуры InitializeTreeView(), в которой каждому узлу сопоставляется его значение. Затем эта процедура вызывается при загрузке формы.
В процедуре TreeView1_AfterSelect_1 происходит сопоставление выбранного узла переменной, используемых в операторе выбора процедуры SortA().
В последней процедуре – обработчике кнопки – инициализируется массив и вызывается процедура выбора сортировки:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        Dim i As Integer
        For i = a.GetLowerBound(0) To a.GetUpperBound(0) - 1 Step 1
            a(i) = Convert.ToInt16(DataGridView1.Rows(0).Cells(i).Value)
        Next i
        SortA()
    End Sub
```

Метод a.GetLowerBound(0) определяет нижнюю границу массива, а a.GetUpperBound(0) – верхнюю. Так как элементы нумеруются с нуля, то из верхней границы вычитается единица.

Достоинства приложения: дружественный интерфейс.
Недостатки приложения: ограниченность длины массива (количества элементов).
Разработка интерфейса. В приложении использовались следующие компоненты: иерархическое дерево выбора элементов (TreeView), текстовое

окно для вывода информации о сортировке (RichTextBox), таблица для ввода элементов массива (DataGridView), кнопка для запуска сортировки (Button), статическое поле для вывода результата (Label).
Окончательно форма задачи имеет вид:



Перечислим некоторые методы приложения:
1) a.Length() – возвращает длину массива;
2) a.Min() – возвращает минимальное значение;
3) a.Max() – возвращает максимальное значение;
4) TreeView1.BeginUpdate() – отключает перерисовку иерархического представления;
5) TreeView1.Nodes.Add("Алгоритмы устойчивой сортировки").Tag = 1 – формирование центрального узла иерархии;
6) TreeView1.Nodes(0).Nodes.Add("Сортировка пузырьком").Tag = 11 – формирование текущего узла дерева.

Полный код программы представлен в электронном приложении

Задача 2.6 б

Постановка задачи. Рассмотреть алгоритм сортировки TimSort одномерного массива.
Внешнее описание: сначала указывается количество и тип элементов массива, затем исходный массив вводится в первый компонент DataGridView1, отсортированный массив выводится во второй компонент DataGridView2.

Функциональная спецификация: элементы массива можно вводить как вручную, так и с помощью датчика случайных чисел.
Система программирования: Microsoft Visual C#.
Особенности реализации и возникшие трудности: Timsort – гибридный алгоритм стабильной сортировки, сочетающий сортировку вставками и сортировку слиянием. Предназначен для работы с большим количество различных данных. Основная идея алгоритма в том, что в реальном мире сортируемые массивы данных часто содержат в себе упорядоченные подмассивы. Основная идея алгоритма:

1) По специальному алгоритму входной массив разделяется на подмассивы.
2) Каждый подмассив упорядочивается сортировкой вставками.
3) Отсортированные подмассивы собираются в единый массив с помощью модифицированной сортировки слиянием.

Перед непосредственной реализацией алгоритма создаются интерфейс класса и дочерние классы TimSort, используя конструктор, для каждого типа данных – целочисленный и вещественный. Затем все методы алгоритма для каждого из двух типов данных (разделение на подмассивы, сортировка и слияние в массив) объединяются в базовый класс TimSort:

```
public class TimSort
        {
            public TimsortInteger ts1;
            public TimsortFlt ts2;
            public int kol;
            public string type;
            public TimSort()
            {
                ts1 = new TimsortInteger();
                ts2 = new TimsortFlt();
                type = "int";
            }
            ~TimSort() { }
        }
```

Достоинства приложения: выбор количества и типа элементов (целый, вещественный) массива.
Недостатки приложения: реализован только один алгоритм сортировки.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовое окно для вывода информации о сортировке (TextBox), таблицы для ввода элементов массива и вывода результата (DataGridView), кнопки для изменения настроек, запуска сортировки, очистки табличных полей, заполнения таблицы случайными числами (Button), статические поля для объяснений (Label), кнопки переключателей (RadioButton), панели для объединения компонентов.

Окончательно форма задачи имеет вид:

Дополнительные процедуры и функции, используемые в приложении:
1) void print_arr<T>(T[] a, int n) – процедура вывода элементов отсортированного массива в таблицу.

2) Обработчик кнопки «ОК»:
Запоминание количества элементов в массиве, запоминание выбора типа элементов (с помощью значения логической переменной), изменение количества столбцов таблицы в зависимости от заданного количества элементов. В этой процедуре происходит проверка на максимальный размер компонента DataGridView (не более 654) и на ошибочный ввод числа элементов.

3) Обработчик кнопки «Сортировать»:

```
if (timsort.type == "int")
            {
                int[] a = new int[timsort.kol];
                for (int i = 0; i != timsort.kol; i++)
                {
                    try
                    {
                        a[i] = Convert.ToInt16(dataGridView1.Rows[0].Cells[i].Value);
                    }
                    catch (OverflowException)
                    {
                        MessageBox.Show("Переполнение типа", "Переполнение типа",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                        break;
                    }

                }
                timsort.ts1.timSort(ref a, timsort.kol);
                print_arr(a, timsort.kol);

            }
```

В этом фрагменте реализуется сортировка целочисленного массива, аналогичный код для вещественного типа записывается в конструкции else. Массив для обработки берется из коллекции, используемой в соответствующем классе. Заполнение массива данными из сетки таблицы производится в блоке try/catch/break с проверкой значений на переполнение памяти. После этого вызывается метод для непосредственной сортировки и процедура вывода результата.

4) Обработчик кнопки «Заполнить случайными числами». Функция датчика случайных чисел меняется в зависимости от типа элементов.
Для целых чисел: `dataGridView1.Rows[0].Cells[i].Value= r.Next(-50, 50);`
Для вещественных чисел: `dataGridView1.Rows[0].Cells[i].Value =Math.Round((-50.6 + r.NextDouble() * (50.6 -20.3)),3);`
Метод Next() возвращает случайное целое число из заданного диапазона.
Метод NextDouble() возвращает случайное вещественное число с заданным количеством знаков после запятой.

5) Обработчик кнопки «Очистить»:
```
for (int i = 0; i != timsort.kol; i++)
            {
                dataGridView1.Rows[0].Cells[i].Value ="";
            }
```
Очищает все ячейки таблицы в цикле.

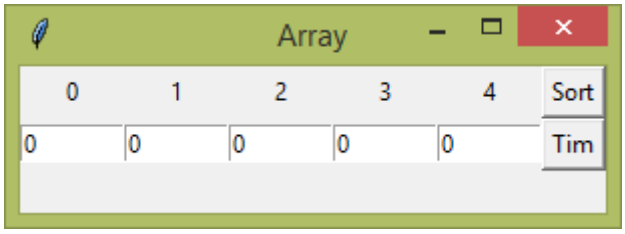
Полный код программы представлен в электронном приложении

Задача 2.6 в

Постановка задачи. Рассмотреть два метода сортировки одномерного массива.
Внешнее описание. Записать элементы массива в текстовые поля, нажать соответствующую алгоритму сортировки кнопку, выдать результат в статическое поле в виде текста.
Функциональная спецификация. Для упрощения работы зададим заранее количество элементов массива, равное пяти. Соответственно, создадим пять текстовых полей, размещённых с помощью метода grid() (сетка).
Система программирования. Python.
Особенности реализации и возникшие трудности: в стандартной библиотеке Tkinter() нет виджета, отвечающего за создание таблиц, поэтому при решении задачи можно использовать два метода:
1) Изменить библиотеку на wxPython, Qt и др.
2) Использовать виджеты Entry, выровняв их по сетке с помощью цикла.
Для решения задачи воспользуемся вторым способом.
По кнопке «Sort» реализуется алгоритм сортировки простым выбором. По кнопке «Tim» - алгоритм TimSort, рассмотренный в предыдущей задаче. В отличие от C# и многих других систем программирования, в Python TimSort

является стандартным методом сортировки, поэтому он вызывается одной командой a.Sort() или Sorted(a).
В отдельной функции реализуется формирование значения текстовой переменной, состоящей из элементов отсортированного массива (списка).

Достоинства приложения: простота реализации.
Недостатки приложения: пользователем нельзя изменить размер массива.
Разработка интерфейса. В приложении использовались следующие виджеты: статический текст для вывода индексов элементов массива и для вывода результата (Label), пять текстовых полей для ввода элементов, выровненных по сетке (Entry), кнопки для сортировки по выбранному алгоритму (Button).
Окончательно форма задачи имеет вид:



Классы, процедуры и функции, используемые в приложении:
1) Основной класс App(Frame), в котором перечислены функции, влияющие на формирование графического окна приложения.
2) get_values(root) – получение числового значения из текстового поля.
3) sort (root, array1) – алгоритм сортировки методом простого выбора.
4) ts(root,array1) – встроенный алгоритм сортировки TimSort.
5) calc_CR(root) и calc_ts(root) – ввод данных в функцию сортировки и вывод значения результата в текстовую переменную для каждого метода соответственно.
6) create_widgets(root) – создание элементов управления на экране.
7) run(root) – функция для запуска приложения.

Полный код программы представлен в электронном приложении

Задача 2.7

Постановка задачи. Кодирование сигнала в цифровых системах связи с помощью натурального и симметричного кода.
Внешнее описание: Ввести напряжение ограничения, кодируемые значения, число уровней квантования. Вывести двоичные коды значений в зависимости от выбора алгоритма кодирования.
Функциональная спецификация: расчет производится в двух соответствующих процедурах — обработчиков кнопок. Т. к. коды представляют собой двоичные числа, то в программе производится перевод числа из десятичной системы счисления в двоичную.
Система программирования: Microsoft Visual Basic

Особенности реализации и возникшие трудности: перевод числа из десятичной системы в двоичную осуществляется по правилу последовательного деления числа на 2 и сбор остатков с конца. Остатки записываются в числовой массив. Этот алгоритм реализован с помощью цикла:

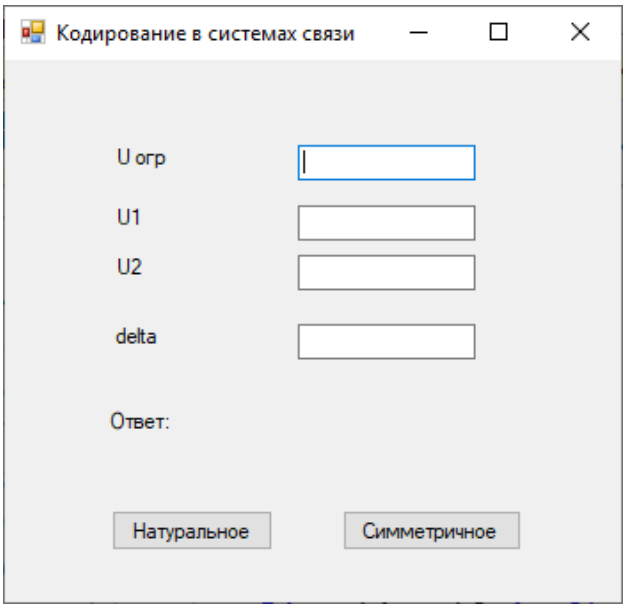
```
For i = 1 To m
            a1(i) = u1 Mod 2
            u1 = u1 \ 2
        Next
```

Сбор остатков происходит конвертацией элементов массива в строку в обратном цикле с отрицательным шагом:

```
 For i = m To 1 Step -1
            ot1 = ot1 + Str(a1(i))
        Next
```

В формуле подсчета числа импульсов в кодовой комбинации используется функция округления в сторону большего целого. В VB встроенная в библиотеку Math функция Round производит округление по правилам математики, поэтому к аргументу этой функции надо прибавить 0,5, чтобы округление всегда велось в большую сторону.

Достоинства приложения: выполнение всех необходимых функций.
Недостатки приложения: не учитываются неверные значения.
Разработка интерфейса. В приложении использовались следующие компоненты: 4 текстовых поля для ввода исходных данных (TextBox), статические поля для пояснений и вывода результата (Label), 2 кнопки для выполнения соответствующего расчета (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.8

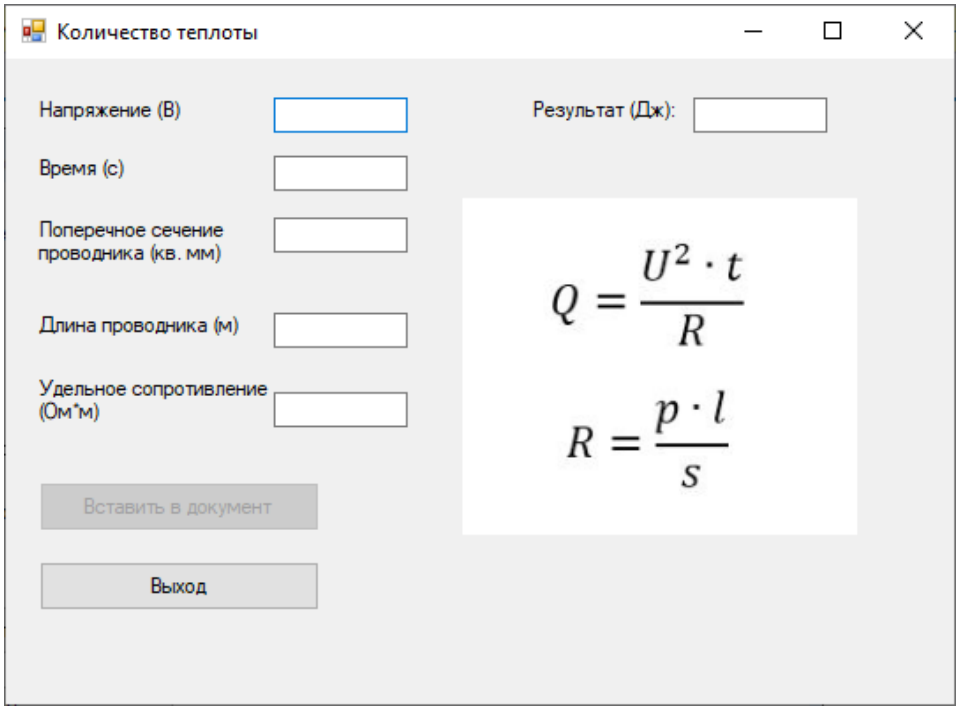
Постановка задачи. Подсчитать количество теплоты, выделяемой в проводнике при протекании по нему тока. Возьмем соответствующую формулу: $Q=U^2t/R$, где Q — количество теплоты, Дж; U — напряжение, В; t — время, с; R — сопротивление, Ом. При этом $R=pl/s$, где p — удельное сопротивление материала проводника, Ом•м; l — длина проводника, м; s — площадь поперечного сечения проводника, $см^2$.
Внешнее описание: Ввести необходимые исходные данные. Результат вывести на форму и сохранить в текстовом файле.
Функциональная спецификация: после ввода значений в текстовые поля результат автоматически появляется в последнем текстовом поле, благодаря обработчику события TextBox_TextChanged. Полученный результат можно вставить в текстовый документ с помощью соответствующей кнопки.
Система программирования: Microsoft Visual Basic
Особенности реализации и возникшие трудности: непосредственный расчет производится в отдельной процедуре solve(), которая вызывается из каждого обработчика TextBox_TextChanged. Перед вычислением производится проверка на соответствие данному из поля числовому значению.
Обработчик кнопки «Выход» очищает поля ввода и вывода и выводит диалоговое окно закрытия программы.
Кнопка «Вставить в документ» активна только в том случае, когда в поле результата появится значение.
После основного расчета в обработчике этой кнопки формируется текстовая переменная, включающая значения из текстовых полей. Для записи информации в текстовый файл в VB существует метод StreamWriter пространства имен System.IO.
Блок вывода в файл имеет вид:

```
Dim fw As System.IO.StreamWriter
        Try
            fw = New System.IO.StreamWriter("Test1.txt", False)
            fw.WriteLine(s)
        Finally
            Process.Start("Test1.txt", "Notepad.exe")
            fw.Close()
        End Try
```

Логический параметр false означает, что файл можно перезаписывать. После успешного сохранения файл должен открыться в стандартном приложении Блокнот.

Достоинства приложения: отсутствие лишних кнопок для выполнения расчета.
Недостатки приложения: перегруженность полями ввода.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовые поля для ввода исходных данных и вывода результата (TextBox), 2 кнопки для сохранения результата в файл и выхода из приложения (Button), графический элемент для визуализации формул (PictureBox), поясняющий текст (Label).

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 2.9

Постановка задачи. Решить задачи из школьного курса физики по теме «Электричество», используя контрольно-измерительные материалы ЕГЭ 11 класс.

Внешнее описание: условие задачи выбрать из соответствующего пункта меню, ввести необходимые по смыслу данные и получить числовой результат

Функциональная спецификация: условие задачи отображается в статическом текстовом поле. В отдельном текстовом поле пользователем вводятся входные данные. Вывод осуществляется в том же окне при нажатии на кнопку «Решение». Количество значений для ввода данных регулируется программно в зависимости от поставленной задачи при нажатии дополнительной кнопки «ОК». Для каждого условия изображается чертеж. В данном приложении реализованы 3 задачи по теме «Электрические схемы» и одна задача на тему «Мощность».

Система программирования: Microsoft Visual C#

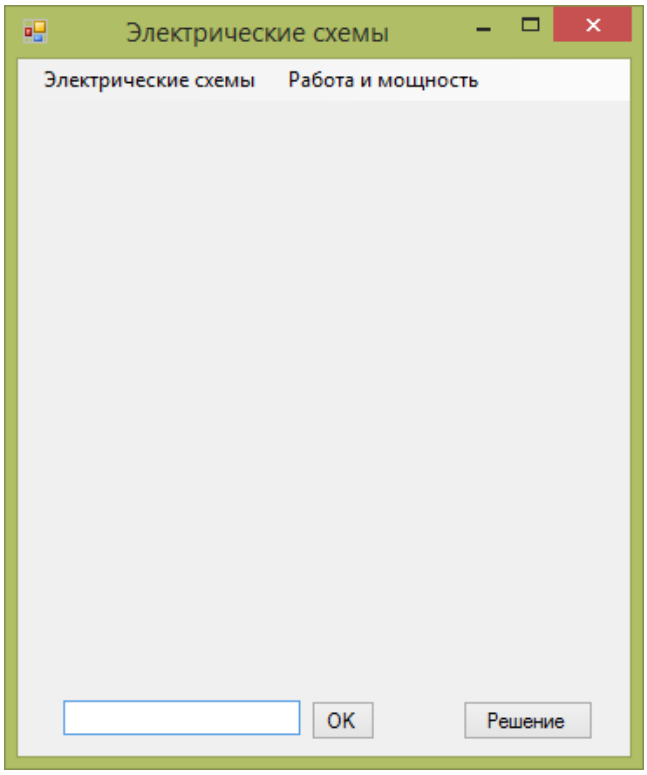
Особенности реализации и возникшие трудности: вывод результата (с нужным чертежом) производится в том же окне, что и ввод данных, поэтому лишние поля программно скрываются, текстовые описания заменяются новыми, а изображения заменяются в зависимости от вида задачи.

В первых трех задачах вводится только одно значение, а в четвертой – 4 значения, поэтому при выборе соответствующего пункта меню необходим обработчик дополнительной кнопки «ОК», в котором будут фиксироваться значения из текстового поля и присваиваться нужным числовым переменным:

```
private void button2_Click(object sender, EventArgs e)
        {

            k++;
            if (k == 1) { float.TryParse(textBox1.Text, out r1); label2.Text = "R2=";
textBox1.Text = ""; }
            if (k==2) { float.TryParse(textBox1.Text, out r2);  label2.Text = "R3=";
textBox1.Text = ""; }
            if (k==3) { float.TryParse(textBox1.Text, out r3); label2.Text = "EDS=";
textBox1.Text = ""; }
            if (k==4) { float.TryParse(textBox1.Text, out eds);  }
        }
```

Значения переменных соотносятся с количеством нажатий на кнопку (переменная k).

Достоинства приложения: для всех задач используется одна общая форма.
Недостатки приложения: однообразные типы задач.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для ввода исходных данных (TextBox), статические поля для пояснений и вывода результата (Label), 2 кнопки для выполнения соответствующего расчета и фиксации вводимых значений (Button), окно графического вывода (PictureBox), меню (MenuStrip). Окончательно форма задачи имеет вид:



В обработчиках пунктов меню происходит смена изображения, отображение или скрытие текстовых полей и кнопки «ОК» и присваивание значения числовой переменной – номеру задачи.

В обработчике кнопки «Решение» производится непосредственный расчет по формулам и вывод результата в зависимости от значения переменной-номера. Например, для первой задачи фрагмент кода имеет вид:

```
if (f == 1)
            {
                float R, Z;
                float.TryParse(textBox1.Text, out R);
                Z = R + R * R / (R + R);

                pictureBox1.Image = Properties.Resources.get_file_1;
                pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
                string s = "";
                if (Z - R > 0) s = " и увеличится на ";
                if (Z - R < 0) s = " и уменьшится на ";
                label1.Text = "Сопротивление = " + Convert.ToString(Z) + " Ом" + s +
Convert.ToString(Z - R) + " Ом";
                label2.Visible = false;
                textBox1.Visible = false;
            }
```

Изображения берутся из файла ресурсов.

Условие первой задачи соответственно записывается при помощи кода:

```
private void задача1ToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            pictureBox1.Image = Properties.Resources.get_file;
            pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
            label1.Text = "На сколько изменится сопротивление участка цепи AB,
изображенного на рисунке, если ключ K разомкнуть? ";
            label2.Text = "Введите сопротивление каждого резистора";
            f = 1;
            textBox1.Visible = true;
            label2.Visible = true;
            button2.Visible = false;
        }
```

Полный код программы представлен в электронном приложении

Задача 2.10

Постановка задачи. Закон Ома. Ввести два из трех значений (сила тока, напряжение, сопротивление) и получить третье значение, а также мощность. Дополнительно вычислить делитель напряжения указанной схемы.

Внешнее описание: Ввести необходимые исходные данные. Результат вывести на форму. Для делителя напряжения по соответствующей кнопке вызывается новый модуль с собственной формой.

Функциональная спецификация: для основной формы расчет производится по нажатию каждой из трех кнопок «Вычислить». В каждом случае вводятся и выводятся по два числовых значения в текстовые поля.

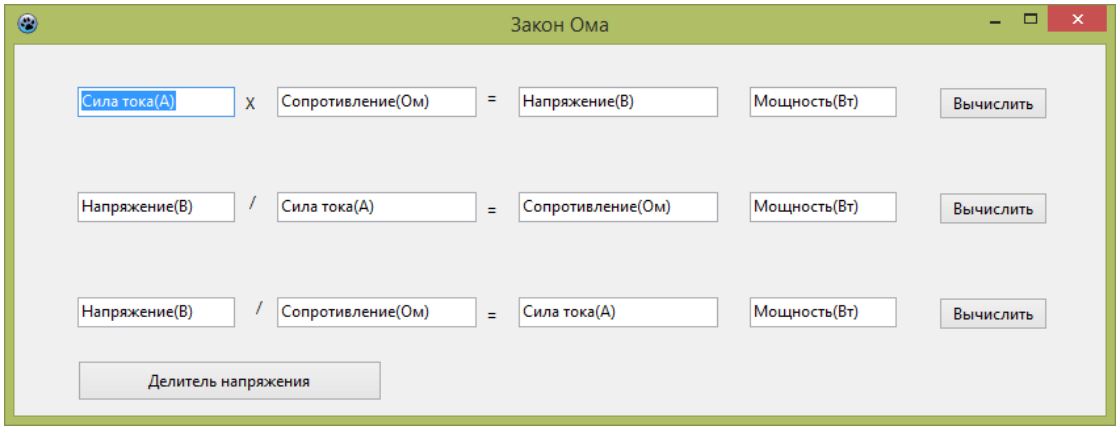
Для дополнительной формы (Делитель напряжения) вводятся 3 значения. По кнопке «Расчет» производится вывод результата в диалоговое окно с последующим отображением на графическом изображении. По кнопке «Очистить» происходит обнуление значений текстовых полей.

Система программирования: Lazarus.

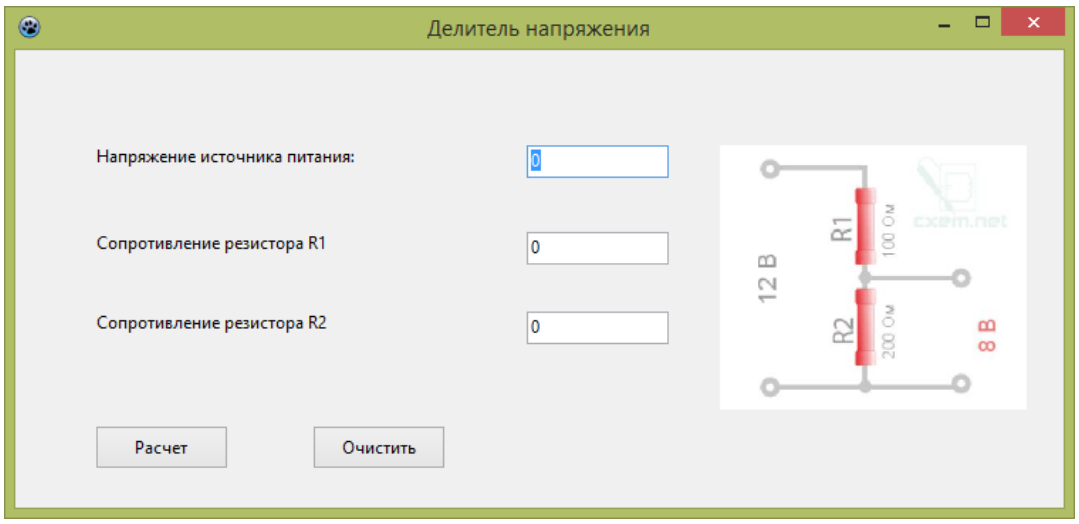
Особенности реализации и возникшие трудности: пояснения к вводу и выводу записаны в тех же текстовых полях. В которые будут вводиться и

выводиться значения. Вывод к схеме делителя напряжения осуществляется на иллюстрации объекта Image. Для этого надо правильно распределить координаты расположения текста (позицию).

Достоинства приложения: простой и наглядный интерфейс.
Недостатки приложения: перегруженность полями ввода, нет проверки корректности ввода данных.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовые поля для ввода исходных данных и вывода результата (Edit), кнопки для выполнения расчета (Button), графический элемент для визуализации схемы (Image), поясняющий текст (Label).
Окончательно форма задачи имеет вид:



Дополнительная форма:



Работа над приложением:
1) Объявить переменные класса:
var A,Om,W, M:double;

2) Записать процедуры обработчиков нажатия на кнопки «Вычислить», например, для первой кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  A := strtofloat(Edit1.Text);
  Om := strtofloat(Edit2.Text);
  W:=A*Om;
  M:=A*W;
  Edit3.Text := floattostr(W);
  Edit10.Text:=floattostr(M);
  end;
```

3) Создать процедуры для очистки текстовых полей одним щелчком мыши:

```
procedure TForm1.Edit10Click(Sender: TObject);
begin
   Edit10.Clear;
end;
```

4) Добавить кнопку с надписью «Делитель напряжения».
5) Выполнить пункт Файл – Создать форму.

6) Нанести элементы управления на вторую форму и вписать код в обработчики событий. Например, фрагмент кода для выдачи результата после расчета имеет вид:

```
ShowMessage(floattostr(u1));
image1.Canvas.TextOut(0,0,floattostr(u1/u));
```

7) Вернуться в код формы один и дописать в строку добавления модулей uses полученный файл unit2, а в обработчик созданной кнопки строку кода:
form2. Show;

Полный код программы представлен в электронном приложении

Глава 3. Работа с текстовой информацией.

Одной из главных повседневных задач пользователя компьютера является работа с текстовой информацией. Доклады и отчеты, заметки и книги, статьи и документы… Порой приходится в потоке символов буквы отделять от чисел, на ходу придумывать имена и пароли, считать количество набранных символов, добавлять дату работы с документом и многое другое. Для этих целей используются разнообразные программные средства.

Задача 3.1

Постановка задачи. Генератор новых имен путем перестановки букв из текстового файла.

Внешнее описание: к программе подгружаются 2 файла с мужскими и женскими именами. По щелчку на соответствующей кнопке формируется новое имя с помощью определенного алгоритма.

Функциональная спецификация: алгоритм для каждого имени (мужское и женское) реализуется с помощью обработчика кнопки.

Женские имена: если последняя буква имени в текстовом файле «-а» или «-я», то ее отбрасываем. Оставшееся слово переворачиваем. Если теперь последняя буква слова – гласная, то добавляем окончание «-я», если согласная, то окончание «-а».

Например, имя «Галина» превратится в «Нилага», а «Елена» в «Нелея».

Мужские имена: если последняя буква имени в текстовом файле «-й», то отбрасываем эту букву и предпоследнюю. Остаток переворачиваем. Если теперь последняя буква является гласной, то к концу слова добавляем «-й».

Например, «Сергей» превратится в «Грес», а «Иван» станет «Навий».

Система программирования: Microsoft Visual C#

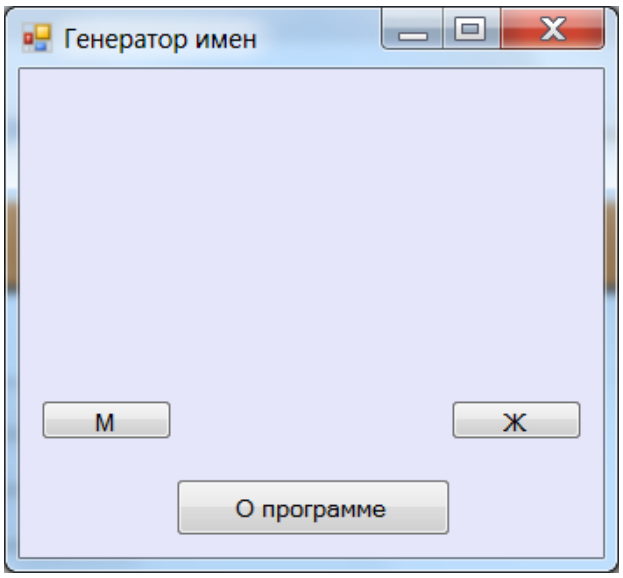
Особенности реализации и возникшие трудности: заранее создаются 2 текстовых файла в кодировке Unicode с мужскими и женскими именами. Из данного файла выбирается одна строка случайным образом. После обрезки последней буквы (если необходимо) данное слово помещается в массив символов, в котором используется функция reverse(). После дальнейших манипуляций с массивом он снова образует текстовую переменную, которая выводится в текстовое поле.

Достоинства приложения: простой понятный интерфейс, большое количество комбинаций, можно изменять исходные файлы.

Недостатки приложения: невозможность добавления полученного имени в новый файл, нет контроля на благозвучность имени.

Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для вывода результата (RichTextBox), 2 кнопки для выполнения соответствующего расчета (Button).

Окончательно форма задачи имеет вид:



Работа над приложением:
1) Random rnd = new Random(); - инициализация датчика псевдослучайных чисел.
2) string[] imyaM = File.ReadAllLines("m.txt"); - чтение данных из текстового файла в строковую переменную.
3) string resm=imyaM[rnd.Next(0, linesCount)]; - выбор случайной строки из файла.
4) конвертация в массив символов и переворачивание:

```
char[] resm2 = resm.ToCharArray();
        Array.Reverse(resm2);
```

5) if (resm[resm.Length - 1] == 'a') resm = resm.TrimEnd('a'); - пример удаления последней буквы в женском имени.
6) Пример добавления окончания к женскому имени:

```
if (resm3[resm3.Length - 1] == 'a' || resm3[resm3.Length - 1] == 'e' ||
resm3[resm3.Length - 1] == 'и' || resm3[resm3.Length - 1] == 'o')
   resm3 = String.Concat(resm3, "я");
else
   resm3 = String.Concat(resm3, "a");
```

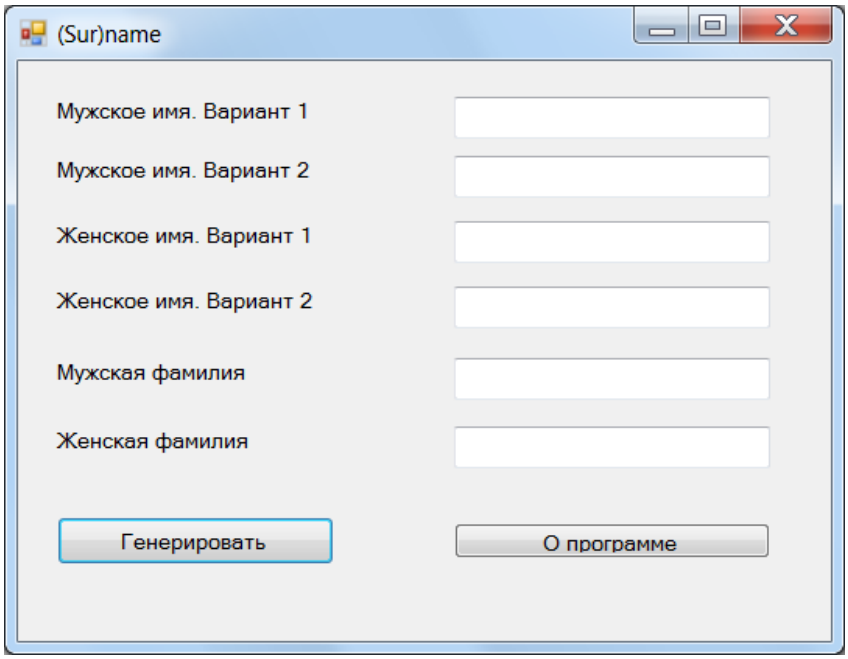
Полный код программы представлен в электронном приложении

Задача 3.1 а

Постановка задачи. Расширенная версия генератора мужских и женских имен и фамилий.
Внешнее описание: из массивов с буквами формируются по 2 мужских и женских имени с разными окончаниями, а также по одной мужской и женской фамилии с тем же корнем.
Функциональная спецификация: алгоритм для каждого имени и фамилии (мужское и женское) реализуется с помощью обработчика одной кнопки. С помощью датчика случайных чисел из массивов с гласными и согласными

буквами формируется основа (корень слова), а затем к нему дописывается окончание по правилу русского языка.
Система программирования: Microsoft Visual C#
Особенности реализации и возникшие трудности: чтобы слово начиналось как с гласной, так и с согласной буквы, создается дополнительный массив из гласных букв и пробелов для формирования первой буквы слова. Для удобочитаемости имени гласные и согласные буквы в слове чередуются. К сожалению, это откладывает негативный отпечаток на генератор, например, нельзя создать слово с двумя согласными подряд – «Петр, Петров», или с двумя гласными – «Виола, Виолова».

Достоинства приложения: простой понятный интерфейс, большое количество комбинаций для имен и фамилий.
Недостатки приложения: невозможность добавления полученного имени и фамилии в новый файл, нет контроля на благозвучность имени, не учитывается частота появления буквы в именах и фамилиях.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовые поля для вывода результатов (TextBox), кнопки для выполнения соответствующего расчета и вывода информации о программе (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.1 б

Постановка задачи. Генератор учетных данных пользователя.

Внешнее описание: программа состоит из двух частей – генерирование имен, фамилий, отчеств и генерирования логина и пароля. Полученный результат можно скопировать в буфер обмена.
Функциональная спецификация: для реализации алгоритма первой части задачи заранее создаются 9 текстовых файлов: основы имени, фамилии и отчества (без окончаний), окончания имен, фамилий и отчеств для мужского и женского варианта. Для второй части задания создаются массивы с буквами, цифрами и другими символами, из которых комбинируются учетные данные.
Система программирования: Microsoft Visual C#
Особенности реализации и возникшие трудности: необходимо записать каждый текстовый файл в кодировке Unicode. Затем подгрузить его командой, например, для генерации мужского имени:

```
string[] imyaM = File.ReadAllLines("Names.txt");
```

В этом коде каждая строка (т.е. каждое имя) записывается в символьный массив, в котором происходит дополнительная обработка. С помощью датчика случайных чисел по индексу массива в текстовое поле помещается случайно выбранная основа имени и случайно выбранное окончание:

```
richTextBox1.Text = imyaM[rnd.Next(0, 104)] + oconchaniya_imen_M[rnd.Next(0, 23)];
```

Аналогичным образом добавляются сформированные фамилия и отчество. Функция rnd.Next(0,104) означает выбор случайного числа из диапазона от 0 до 104, учитывая, что в текстовом файле, а значит, и в массиве содержится 105 строк.
Имена в файле не делятся на женские и мужские, они регулируются только окончаниями, например, от корня «ларис» можно получить мужские имена «Ларисур», «Ларисий» и женские «Лариса», «Ларисия», что представляет широкий диапазон для творчества.
Процедуры для создания логина и пароля почти одинаковы, различаются только количество элементов в символьных массивах (для пароля добавлено больше символов) и длиной результирующей комбинации. Создать массив символов из строки можно несколькими способами, в этом приложении использовалась конвертация:

```
char[] data = symbols.ToCharArray(0, symbols.Length);
```

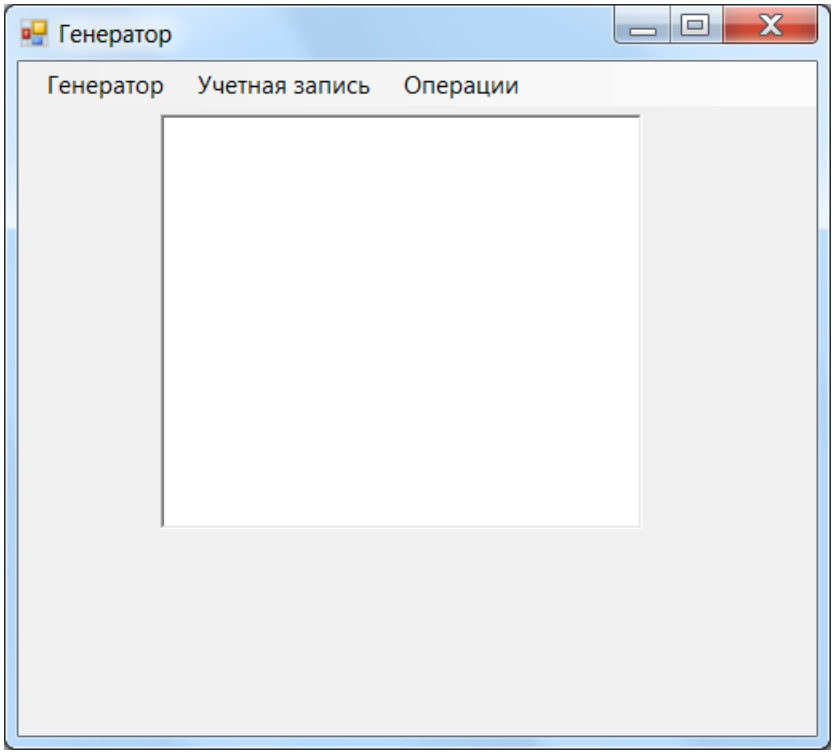
Затем формируется последовательность из случайно взятых (по индексу) символов:

```
s += Convert.ToString(data[rnd.Next(0, (symbols.Length - 1))]);
```

В этом алгоритме не учитывается регистр букв (заглавные или строчные), нет проверки на похожесть некоторых символов, например, цифры «0» и буквы «О». Кроме того, такой пароль не поддерживает правила криптографии на устойчивость, но для демонстрационных целей он подходит.

Достоинства приложения: простой понятный интерфейс, большое количество комбинаций для имен и фамилий.
Недостатки приложения: невозможность добавления полученного имени и фамилии в новый файл, нет контроля на благозвучность имени, не учитывается частота появления буквы или цифры в логинах и паролях.

Разработка интерфейса. В приложении использовались следующие компоненты: расширенное текстовое поле для вывода результата (RichTextBox) и меню для выполнения всех операций в программе (menuStrip). Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.1 в

Постановка задачи. Модификация генератора для создания имен, фамилий и отчеств.

Внешнее описание: по нажатию на кнопку с нужным полом генерируется либо мужское, либо женское полное имя.

Функциональная спецификация: текстовые файлы с шаблонами имен и их окончаний записываются в ресурсы проекта. Затем с помощью трех разных алгоритмов (комбинация основы и окончания из файла; инверсия букв в готовых именах из файла; набор последовательности из отдельных букв случайным образом) формируется строковый массив из трех имен. После этого с помощью датчика псевдослучайного числа выбирается одно имя и записывается в текстовое поле результата. Аналогичные операции осуществляются для фамилии и отчества. Два одинаковых алгоритма с разными исходными данными реализуются в обработчике соответствующей кнопки.

Система программирования: Microsoft Visual C#

Особенности реализации и возникшие трудности: для простоты рассмотрим алгоритм на примере генерирования мужского имени.

1) Комбинация данных из двух текстовых файлов:

```
string imyaM = Properties.Resources.Names; //создание текстовой переменной с данными из файла ресурсов
genmi[0] = imyaM1[rnd.Next(0, 257)].ToString() + oconchaniya_imen_M1[rnd.Next(0, 23)].ToString(); // формирование имени из случайно выбранных частей
```

При формировании отчества могут возникнуть трудности с изменением букв, например, «Валерий – Валерьевич». В программе не предусмотрены случаи-исключения, поэтому можно получить некорректное отчество «Валеривич».

2) Инверсия букв из слов файле с последующим дописыванием окончания:

```
string[] imyaM22 = imyaM2.Split('\n'); // создается массив строк
string resm = imyaM22[rnd.Next(0, linesCount)]; // выбирается случайное имя
if (resm[resm.Length - 1] == 'й') resm = resm.TrimEnd('й'); // удаление последней буквы «й» из имени
char[] resm2 = resm.ToCharArray(); // создание массива из отдельных букв основы
Array.Reverse(resm2); // запись символов в обратном порядке
string resm3 = new string(resm2); // создание новой текстовой переменной
resm3 = resm3.Trim(); // удаление лишних пробелов в начале и конце слова
resm3 = resm3.Substring(0, 1).ToUpper() + resm3.Substring(1, resm3.Length - 1); // выделение первой буквы слова, чтоб сделать ее прописной, а остальные буквы записать строчными
if (resm3[resm3.Length - 1] == 'a' || resm3[resm3.Length - 1] == 'e' || resm3[resm3.Length - 1] == 'и' || resm3[resm3.Length - 1] == 'о')
resm3 = String.Concat(resm3, "й"); // добавление к имени буквы «й» последней, если основа заканчивается на гласную
genmi[1] = resm3; // запись сформированного имени во второй элемент массива
```

При формировании отчества могут возникнуть те же проблемы с некорректностью формирования окончания.

3) Формирование последовательности из символьных массивов с гласными и согласными буквами, учитывая их чередование:

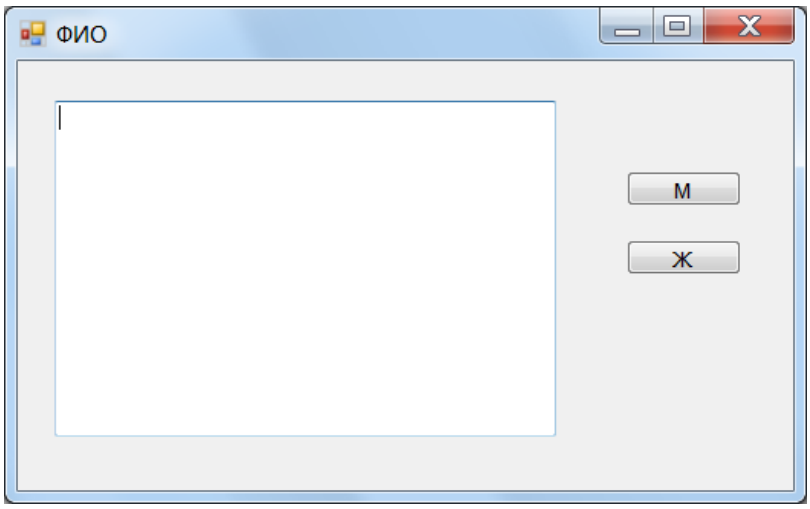
```
string m1 = n[rnd.Next(0, 14)].ToString() + (char)s[rnd.Next(0, 17)] … // формирование первых двух букв имени
textBox1.Text= genmi[rnd.Next(0, 2)].ToString(); // выбор окончательного случайного имени из трех сформированных
```

Достоинства приложения: простой понятный интерфейс, большое количество комбинаций для имен и фамилий.

Недостатки приложения: невозможность добавления полученного имени и фамилии в новый файл, нет контроля на благозвучность имени, не согласованность основы с окончанием в отчествах.

Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для вывода результата (TextBox) и две кнопки для реализации алгоритма (Button).

Окончательно форма задачи имеет вид:



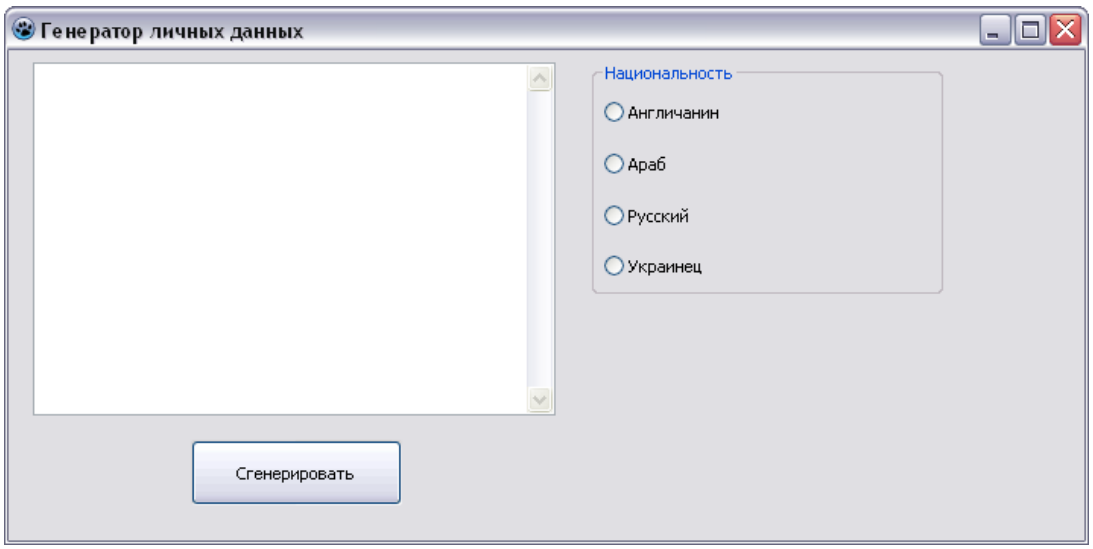
Полный код программы представлен в электронном приложении

Задача 3.1 г

Постановка задачи. Генератор учетных данных пользователей разных национальностей.
Внешнее описание: программа состоит из выбора национальности человека из четырех предложенных с последующим генерированием имен, фамилий, отчеств и некоторых других сведений (в зависимости от выбора). Полученный результат можно скопировать в буфер обмена.
Функциональная спецификация: все данные для случайного выбора хранятся в соответствующем массиве. После выбора национальности в процедуре обработки кнопки выбирается соответствующий массив. Количество данных тоже неодинаково, например, для английских полных имен нет отчеств. После нажатия на кнопку в диалоговом окне надо выбрать пол – мужской или женский. После формирования системой полного имени (в некоторых случаях дополнительно профессии и города) в текстовом поле записывается случайная дата рождения, после чего программой вычисляется знак зодиака и символ восточного гороскопа по данной дате.
Система программирования: Lazarus
Особенности реализации и возникшие трудности: для данных о каждой национальности формируется многомерный массив, например:
HumanDataR: Array[1..AttrCountR,1..2,1..VarCount] of String
Имена в массиве делятся на женские и мужские, а их выбор осуществляется с помощью второго измерения массива (1..2). Константа AttrCountR указывает на количество атрибутов, например, для национальности «русский» она равна пяти: имя, фамилия, отчество, профессия, город. Константа VarCount равна количеству элементов массива для генерации, например, VarCount=10, если в массиве используются 10 разных имен.

Достоинства приложения: простой понятный интерфейс, возможность использования не только русскоязычных имен, добавление дополнительных сведений, в том числе, знаков зодиака.

Недостатки приложения: не большое количество комбинаций для имен и фамилий, который устраняется только программистом путем добавления нужных слов в соответствующий массив, невозможность добавления полученного имени и фамилии в файл.
Разработка интерфейса. В приложении использовались следующие компоненты: многострочное текстовое поле для вывода результата (Memo), группа радиокнопок для выбора национальности (RadioGroup), основная кнопка для генерирования данных (Button).
Окончательно форма задачи имеет вид:



Основные команды кода приложения:
1) Sex := strtoint(inputbox('Введите пол:','1 - мужской, 2 - женский',' ')); - ввод цифры пола для связывания с массивом данных.

2) Фрагмент кода для выдачи данных англичанина:

```
if radiogroup1.ItemIndex=0 then   begin
 for Attr := 1 to AttrCountA
 do begin
    Index := Random(VarCount)+1;
    Memo1.Lines.Add(HumanDataAn[Attr,Sex,Index]);
    end;
 end;
```

3) Выбор случайной даты, например, года (аналогично для дня и месяца):

```
year:=1919+Random(100);
```

4) Селектор для вывода знака зодиака (фрагмент):

```
case month of
 1: if day<20 then res:='Козерог' else res:='Водолей';
 2: if day<19 then res:='Водолей' else res:='Рыбы';
…
```

5) Селектор для вывода символа восточного гороскопа (фрагмент):
```
case year of
 1919,1931,1943,1955,1967,1979,1991,2003,2015:vost:='Овца';
…
```

6) Memo1.Lines.Add(DateToStr(EncodeDate(year,month,day))); - добавление сформированной даты рождения в конец текста.

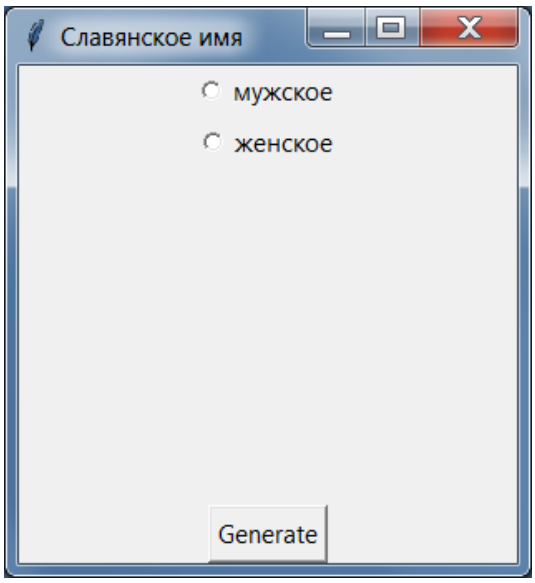
Полный код программы представлен в электронном приложении

Задача 3.1 д

Постановка задачи: генератор славянских имен путем соединения корня и окончания.
Внешнее описание: Щелчком по радиокнопке выбирается имя – мужское или женское, которое отображается в текстовом поле.
Функциональная спецификация: Вариант №1. Используются 2 массива из символов с первой и второй частями имени. Из каждого массива случайным образом выбирается по одному слову и записывается в одну строку. Для женского имени добавляется окончание «а».
Система программирования: Python
Особенности реализации и возникшие трудности: генерирование слов происходит в функции-обработчике кнопки. Элементы массива фиксированы, что уменьшает количество неповторяющихся имен.
Основной алгоритм (без учета условия радиокнопки):

```
    n=round(random()*15) #выбор случайного индекса массива 1 части
   k=round(random()*15) #выбор случайного индекса массива 2 части
   s1=namesn[n-1]+namesk[k-1]   #генерирование имени
   lab.config(text=s1) #формирование переменной для последующего вывода в статическую текстовую область.
```

Достоинства приложения: простой интерфейс, реализация основной функции.
Недостатки приложения: фиксированный размер массива, количество элементов для генерирования ограничено.
Разработка интерфейса. В приложении использовались следующие виджеты: статическое текстовое поле для вывода (Label), кнопка генерации (Button), радиокнопки для выбора имени (RadioButton).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Вариант №2.

Модифицируем программу таким образом, чтобы фрагменты имени для генерации хранились в текстовом файле. Пусть текстовый файл имеет вид:
[part1]
Слова первой части
[part2]
Слова второй части.
Тогда алгоритм генератора имеет вид:
1) Открыть файл для чтения.
2) Создать пустой список.
3) Добавить строки файла в список с помощью среза, учитывая местоположение квадратных скобок.
4) Выбрать случайным образом два значения (из верхней и нижней части) и записать вместе в текстовую переменную.
Соответствующий фрагмент программы на языке Python:

```
with open('parts.txt', 'r') as f:
        currentList = []
        for line in f.readlines():
            line = line.strip()
            if line.startswith('[') and line.endswith(']'):
                currentList = []
                parts[line[1:-1]] = currentList
            else:
                currentList.append(line.strip())
    for partName in sorted(parts):
        s+=random.choice(parts[partName])
```

Полный код программы представлен в электронном приложении

Вариант №3.

Объединим оба алгоритма в одном. Для этого запишем 2 текстовых файла. Файл name1.txt содержит первые части имен, а name2.txt – вторые части. Создадим два соответствующих списка, в которые производится чтение данных из каждого файла. Пример для первой части:

```
with open('name1.txt') as f1:
    first_names = f1.read().split('\n')
```

Разделитель – признак конца строки.

В первой функции пользователя происходит генерирование имени путем случайного выбора слов из двух списков:

```
first = random.sample(first_names, 1)[0]
```

Возвращаемым значением функции является сгенерированное имя.

Во второй функции – обработчике кнопки – реализуется условие повторения имени. Неповторяющееся имя записывается в новый список OUT для сравнения с другими элементами этого списка в цикле, а затем выводится в текстовую переменную:

```
if name not in OUT:
            OUT.append(name)
```

Полный код программы представлен в электронном приложении

Задача 3.1 е

Постановка задачи: генератор имен путем случайного выбора гласных и согласных букв алфавита.

Внешнее описание: щелчком по кнопке генерируется имя и фамилия, которые отображаются в статическом текстовом поле.

Функциональная спецификация: Используются 2 массива из гласных и согласных букв. Из каждого массива случайным образом выбирается по одной букве и записывается в одну строку.

Система программирования: Python

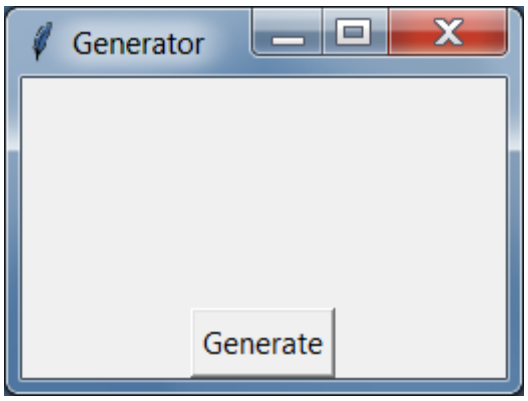
Особенности реализации и возникшие трудности: для случайного выбора числа из определенного промежутка используется функция randint библиотеки random.

Сначала выбирается случайная длина слова – будущего имени, например, random.randint(3,8) – от 3 до 8 символов. Это число будет конечным значением цикла в генераторе. Затем реализуется алгоритм чередования, когда гласная буква сменяется согласной и наоборот:

```
if(x == 0 or x % 2 == 0):
        firstName = firstName + bLetters[random.randint(0,18)]
else:
        firstName = firstName + aLetters[random.randint(0,7)]
```

Аналогичным образом выглядит алгоритм для фамилии. Затем оба слова, записанные с прописной буквы, через пробел формируются в одной текстовой переменной для вывода.

Достоинства приложения: простой интерфейс, реализация основной функции.
Недостатки приложения: нет проверки на благозвучие имени.
Разработка интерфейса. В приложении использовались следующие виджеты: статическое текстовое поле для вывода (Label), кнопка генерации (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении
Чтобы избавиться от предыдущего недостатка, изменим алгоритм на следующий:
1) Из букв случайным образом формируются два слога: а) гласная-согласная-гласная; б) согласная-гласная-согласная:

```
firstName=aLetters[random.randint(0,7)].upper()+bLetters[random.randint(0,18)]+aLetters[r
andom.randint(0,7)]
```

2) В цикле эти слоги помещаются в два списка (по аналогии со списками из файлов):

```
first_names.append(firstName)
```

3) Из каждого списка выбираются случайные слова, которые соединяются в одно имя:

```
first = random.sample(first_names, 1)[0]

last = random.sample(last_names, 1)[0]

name=first + last
```

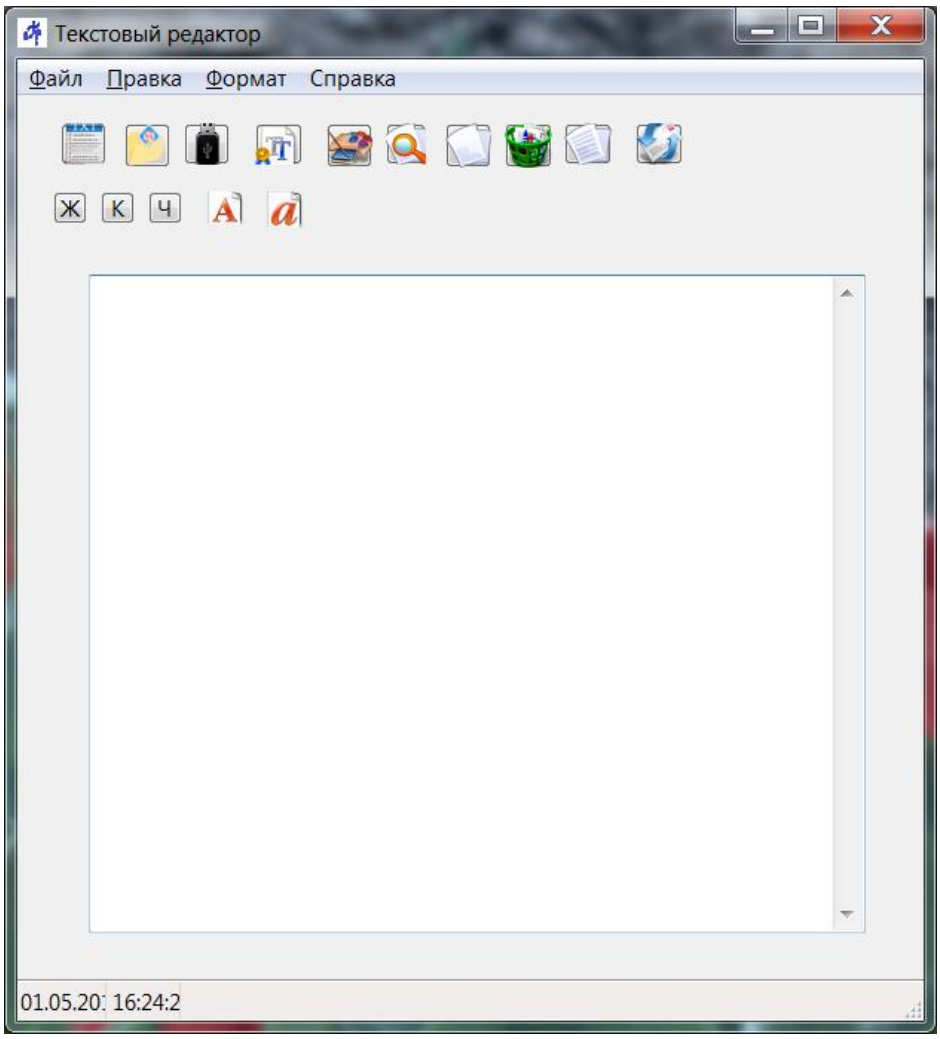
Полный код программы представлен в электронном приложении

Задача 3.2

Постановка задачи. Текстовый редактор с основными функциями – замена стандартного Блокнота.
Внешнее описание: С помощью пунктов меню или быстрых кнопок в окне данного редактора можно осуществлять основные функции работы с текстом.
Функциональная спецификация: создание, открытие, сохранение, печать файла; работа с буфером обмена, вставка и показ даты и времени; изменение

шрифта и цвета фона, поиск и замена слов; вывод количества символов и строк.
Система программирования: Lazarus
Особенности реализации и возникшие трудности: в среде Lazarus существует только один стандартный компонент для многострочного текста – Memo, который работает только с plain-текстом. Это означает, что все виды форматирования (изменения шрифта, начертания) применимы только ко всему тексту целиком, а не к отдельным символам.
Поиск и замена символов осуществляются с помощью стандартных диалоговых окон FindDialog и ReplaceDialog, но в данном случае замена корректно выполняется только после вызова окна поиска.
Некоторые пункты меню дублируются через быстрые кнопки (SpeedButton), но полное соответствие отсутствует, например, некоторые кнопки не имеют подобные функции в меню.
В обработчике события меню и кнопки «Создать» реализуется вывод диалогового окна с вопросом о сохранении изменений. В случае ответа «да» происходит вызов диалогового окна сохранения файла, а потом поле очищается. При ответе «нет» содержимое текстового поля очищается.

Достоинства приложения: простой интерфейс, реализация основных функций текстового редактора.
Недостатки приложения: невозможность форматирования отдельных символов, некорректность работы замены текста.
Разработка интерфейса. В приложении использовались следующие компоненты: многострочное текстовое поле для основной работы (Memo), основное меню (MainMenu), кнопки для быстрого доступа к функциям приложения (SpeedButton), стандартные диалоговые окна (OpenDialog, SaveDialog, SelectDirectoryDialog, FontDialog, ColorDialog, FindDialog, ReplaceDialog, PrinterSetupDialog, PrinterDialog, PageSetupDialog), таймер для отображения текущей даты и времени (Timer), строка состояния (StatusBar).
Окончательно форма задачи имеет вид:



Основные команды и процедуры кода приложения:

1) Окно открытия файла:

```
  with opendialog1 do
if execute then
begin
  memo1.Lines.LoadFromFile(Filename);    // загрузка строк файла в строки мемо
  Caption:='Текстовый редактор'+extractfilename(Filename);  // добавление к заголовку
имени открытого файла
  savedialog1.filename:=Filename;
  end;
```

2) Фрагмент процедуры Memo1Change() для подсчета символов и строк:

```
if s[i]<>'' then k:=k+1;   // подсчет количества символов для вывода в строке состояния
  StatusBar1.Panels[2].Text := 'Символов: '+inttostr(k)+'   Строк:
'+inttostr(memo1.Lines.Count);
```

3) Процедура поиска FindDialog1Find():

```
Var   Found, StartPos: Integer;
Begin
   if Memo1.SelLength <> 0 then // для повторного поиска
      StartPos := Memo1.SelStart + Memo1.SelLength
   else
```

```
    StartPos := 0;
  // Для PosEx надо подключить StrUtils
  Found := PosEx(FindDialog1.FindText, Memo1.Text, StartPos + 1);
  if Found <> 0 then
  begin
    Memo1.HideSelection := False;
    Memo1.SelStart := Found - 1;
    Memo1.SelLength := Length(FindDialog1.FindText); // поиск текста в каждой строке
  end
  else
    MessageDlg('Строка ' + FindDialog1.FindText + ' не найдена!', mtConfirmation, [mbYes],
0);
```

4) Фрагмент процедуры создания нового файла:

```
bs:= MessageDlg('Сохранить изменения?', mtError, [mbYes, mbNo, mbCancel],0);
 if  bs=mrYes then
 begin
if savedialog1.filename<>'' then memo1.lines.SaveToFile(savedialog1.filename)
else savedialog1.execute;
memo1.Clear;  // сохранение изменений перед очисткой
end;
if bs=mrNo then memo1.Clear;   // очистка экрана без сохранения
```

5) Процедура печати текста:

```
IF PrintDialog1.Execute THEN
  BEGIN
  Printer.Canvas.Font:= Memo1.Font; // Установка шрифта печати из свойства шрифта
текущего текста
  Printer.Canvas.Font.Size:=memo1.font.Size;
 Printer.Canvas.Font.Color:=memo1.font.Color;
  Printer.BeginDoc;    // печать текста по строкам
  for N:=0 to Memo1.Lines.Count-1 do begin
   if (N>0) and (N mod 40=0) then Printer.NewPage; // если в тексте более 40 строк, то печать
каждых 40 строк начинать с новой страницы
Printer.Canvas.TextOut(printer.Canvas.Textwidth('I'),printer.Canvas.TextHeight('I')*(N mod
40),Memo1.Lines[N]); // выбор печати от начального значения до конечного по строкам
поля Memo
       end;
      Printer.EndDoc; // конец печати
      END;
```

6) Вставка информации в текстовое поле:

```
memo1.Lines.Insert(memo1.Lines.Count, DateToStr( Date )); // вставка в текст даты
memo1.Lines.Insert(memo1.Lines.Count, TimeToStr( Time ));  // вставка в текст времени
```

7) Открытие html-файла справки:

```
ShellExecute(0,nil, PChar('cmd'),PChar('/c start main.htm'),nil,0);
```

8) Процедура ReplaceDialog1Replace() замены текста:

```
WITH Sender AS TReplaceDialog DO
```

```
    WHILE  True DO
  BEGIN
    IF  Memo1.SelText <> FindText THEN
    FindDialog1Find(Sender);
    IF  Memo1.SelLength = 0 THEN Break;
    Memo1.SelText:= ReplaceText; // замена текста (работает в паре с поиском)
    IF NOT (frReplaceAll IN Options) THEN Break;
  END;
```

9) Форматирование текста:

```
memo1.SelText:= memo1.SelText.ToUpper;  // прописные буквы
memo1.Font.Style:= font.style + [fsitalic];   // курсив
```

10) Для отображения текущей даты и времени в строки состояния с ежесекундным обновлением используется процедура обработки таймера:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 StatusBar1.Panels[0].Width:=60;
 StatusBar1.Panels[0].text:=DateToStr(now);
   StatusBar1.Panels[1].text:=TimeToStr(now); // дата и время в строке состояния
end;
```

Полный код программы представлен в электронном приложении

Задача 3.2 а.

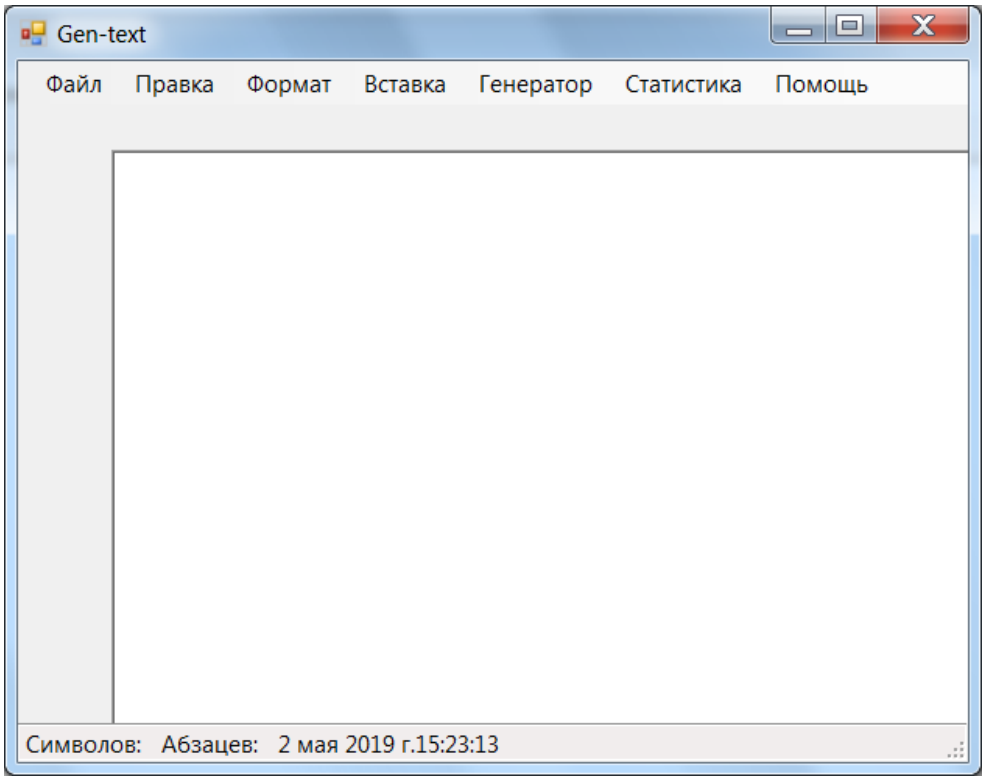
Постановка задачи. Текстовый редактор с дополнительными функциями.
Внешнее описание: создание, открытие и сохранение файлов форматов .rtf, .txt. Работа с основными функциями форматирования и вставки текста.
Функциональная спецификация: создание, открытие, сохранение, печать файла; работа с буфером обмена, вставка и показ даты и времени; изменение шрифта и цвета фона, поиск и замена слов; вывод количества символов и строк, вставка сгенерированных имен, фамилий и паролей.
Система программирования: Microsoft Visual Basic.
Особенности реализации и возникшие трудности: по умолчанию текстовое поле в VB не поддерживает автоматическое контекстное меню, поэтому необходимо создавать его отдельно вручную. Для простоты реализации все операции над текстом проводятся только в пунктах основного меню.
Отдельная проблема с форматом файла. Для работы с rich text и plain text в этой среде предусмотрены разные методы, поэтому необходимый метод вызывается исходя из условия:

```
Try
RichTextBox1.LoadFile(OpenFileDialog1.FileName, RichTextBoxStreamType.RichText)
        Catch
RichTextBox1.LoadFile(OpenFileDialog1.FileName, RichTextBoxStreamType.PlainText)
        End Try
```

В этом фрагменте кода происходит попытка открыть .rtf документ. Если эта попытка не удачна, то документ открывается как .txt.

В программе реализованы основные функции работы с текстом, как в предыдущей задаче: открытие, сохранение, печать документа, работа с буфером обмена, поиск и замена слов, откат предыдущих действий, форматирование символов (шрифт, начертание, регистр, цвет).
Кроме того, добавлена возможность вставки изображения, таблицы и списка, как в стандартном редакторе WordPad.
Дополнительно в программе реализован пункт меню «Генератор», в котором добавлены функции из задачи 3.1: вставка имени, фамилии, отчества, даты рождения, логина и пароля. И пункт «Статистика», позволяющий выводить числовую информацию для анализа символов.

Достоинства приложения: простой интерфейс, реализация основных и дополнительных функций текстового редактора с элементами генератора и статистических данных.
Недостатки приложения: нет быстрых кнопок для доступа к основным командам и контекстного меню.
Разработка интерфейса. В приложении использовались следующие компоненты: многострочное текстовое поле для основной работы (RichTextBox), основное меню (MenuStrip), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog, FontDialog, ColorDialog, PrintDialog, PageSetupDialog, PrintDocument), таймер для отображения текущей даты и времени (Timer), строка состояния (StatusStrip).
Окончательно форма задачи имеет вид:



Основные команды и процедуры кода приложения:

1) Сохранение файла:

```
If SaveFileDialog1.FileName <> "" Then RichTextBox1.SaveFile(SaveFileDialog1.FileName)
        RichTextBox1.Modified = False
```

2) Печать документа:

```
PrintDialog1.Document = PrintDocument1
        Dim result As DialogResult = PrintDialog1.ShowDialog()
        '    'Если сделан щелчок на OK, печатаем документ на принтере
        If result = DialogResult.OK Then
            PrintDocument1.Print()
        End If
```

3) Смена шрифта для выделенного текста:

```
FontDialog1.ShowDialog()
RichTextBox1.SelectionFont = FontDialog1.Font
```

4) Поиск слов:

```
Dim txt As String
        txt = InputBox("Поиск слов", "Найти?", "")
        RichTextBox1.Find(txt)
```

5) Замена одного слова на другое:

```
Dim txt, zam As String
        txt = InputBox("Поиск слов", "Найти?", "")
        zam = InputBox("Заменить?", "Замена", "")
        RichTextBox1.Text = RichTextBox1.Text.Replace(txt, zam)
```

6) Вставить логин (как случайную комбинацию латинских букв):

```
Dim n() As String
        Dim i, k As Integer
        Dim name As String
        name = ""
        ReDim n(26)
        Randomize()
        Const ab As String = "a b c d e f g h i j k l m n o p q r s t u v w x y z"
        n = Split(ab, " ")
        k = Int((6 * Rnd()) + 3)
        For i = 1 To k
            name = name + n(Int(26 * Rnd()))
        Next
        name = name.Substring(0, 1).ToUpper + name.Substring(1, name.Length - 1)
        RichTextBox1.Text += name + " "
```

7) Вставка случайного мужского имени из массива:

```
Randomize()
        Dim name1 As String() = New String(30) {"Андрей", "Борис", "Виктор", "Глеб",
"Денис", "Егор", "Александр", "Алексей", "Артем", "Арсений", "Владислав", "Дмитрий",
"Евгений", "Иван", "Игорь", "Илья", "Кирилл", "Максим", "Матвей", "Михаил", "Никита",
"Роман", "Руслан", "Сергей", "Тимофей", "Тимур", "Ярослав", "Антон", "Семён", "Николай",
"Степан"}
        RichTextBox1.Text += name1(Int(30 * Rnd())) + " "
```

8) Вставка даты рождения, учитывая количество дней (28, 30, 31) в каждом месяце:

```
Dim dr As String
        Dim y, m, d As Integer
        Randomize()
```

```
        y = Int(99 * Rnd() + 1910)
        m = Int(12 * Rnd() + 1)
        If m = 1 Or m = 3 Or m = 5 Or m = 7 Or m = 8 Or m = 10 Or m = 12 Then d = Int(31
* Rnd() + 1)
        If m = 4 Or m = 6 Or m = 9 Or m = 11 Then d = Int(30 * Rnd() + 1)
        If m = 2 Then d = Int(28 * Rnd() + 1)
        dr = Str(d) + "." + Str(m) + "." + Str(y) + " "
        RichTextBox1.Text += dr
```

9) Женские имена, а также фамилии, отчества, пароль генерируются аналогичными способами.

10) Подсчет авторских листов (1 авторский лист составляет 40000 знаков с пробелами):

```
        Dim k As Integer
        Dim s As Single
        k = RichTextBox1.TextLength
        s = Math.Round(k / 40000, 2)
        MsgBox(Str(s) & " alk")
```

11) Стандартный метод подсчета строк в тексте Lines.Count не учитывает автоматический перенос текста на новую строку, поэтому правильнее назвать эту операцию подсчетом количества абзацев. Эта информация высвечивается в статус-строке при каждом обновлении текста, поэтому используется процедура RichTextBox1_TextChanged():

```
        Dim st, k As Integer
        k = RichTextBox1.TextLength
        st = RichTextBox1.Lines.Count
        ToolStripStatusLabel1.Text = "Символов: " & Str(k)
        ToolStripStatusLabel2.Text = "Абзацев: " & Str(st)
```

12) Отображение в статус-строке текущей даты и времени формируется в процедуре Timer1_Tick():

```
ToolStripStatusLabel3.Text = Now.ToLongDateString & Now.ToLongTimeString
```

13) Прописные буквы:

```
RichTextBox1.SelectedText = RichTextBox1.SelectedText.ToUpper()
```

14) Подсчет количества слов, первое из которых выделено в тексте:

```
Dim txt, slovo As String
        txt = RichTextBox1.Text
        txt = txt.Replace(vbLf, " ")
        slovo = RichTextBox1.SelectedText
        slovo = slovo.Trim
        If slovo = "" Then MsgBox("Выделите нужное слово!")
        Dim all() As String
        all = txt.Split()
        Dim i, k As Integer
        For i = 0 To UBound(all)
            If all(i) = slovo Then k = k + 1
        Next
        MsgBox(Str(k))
```

15) Поиск в тексте чисел и их суммирование:

```
Dim txt As String
```

```
        Dim summa As Single
        txt = RichTextBox1.Text
        Dim all() As String
        summa = 0
        Dim i As Integer
        txt = txt.Replace(vbLf, " ")
        all = txt.Split()
        For i = 0 To UBound(all)
            If IsNumeric(all(i)) Then
                summa += all(i)
                RichTextBox1.SelectionStart = txt.IndexOf(all(i))
                RichTextBox1.SelectionLength = all(i).Length
                RichTextBox1.SelectionColor = Color.Blue
            End If
        Next
        MsgBox(Str(summa))
```

Для этой операции сначала все строки текста помещаются в переменную с единым разделителем «пробел». Затем из этой строки создается массив строк. Если элемент массива – число, то оно прибавляется к предыдущему. Каждая позиция числа в тексте выделяется, и цвет числа меняется на синий.

16) В выделенном тексте начертание меняется на курсив и обратно с курсива на обычный:

```
If RichTextBox1.SelectionFont.Italic Then
            RichTextBox1.SelectionFont = New Font(RichTextBox1.Font.FontFamily,
RichTextBox1.Font.Size, FontStyle.Regular)
        Else
            RichTextBox1.SelectionFont = New Font(RichTextBox1.Font.FontFamily,
RichTextBox1.Font.Size, FontStyle.Italic)
        End If
```

17) Вставка рисунка с помощью диалогового окна открытия файла:

```
OpenFileDialog1.ShowDialog()
        Dim bmp As New Bitmap(OpenFileDialog1.FileName)
        Clipboard.SetImage(bmp)
        RichTextBox1.Paste()
```

18) Преобразование выделенного текста в маркированный список:

```
If Not RichTextBox1.SelectionBullet Then
            RichTextBox1.SelectionBullet = True
            RichTextBox1.SelectionIndent = 20
            RichTextBox1.SelectionHangingIndent = 15
            RichTextBox1.SelectionRightIndent = 20
        Else
            RichTextBox1.SelectionBullet = False
            RichTextBox1.SelectionIndent = 0
            RichTextBox1.SelectionHangingIndent = 0
            RichTextBox1.SelectionRightIndent = 0
        End If
```

19) Вставка простой таблицы из одной строки и двух столбцов. Клавиша Enter добавляет новую строку к таблице. Код опирается на тэги формата .rtf:

```
Dim table As String
        table = "\trowd\cellx1000\cellx2000\intbl \cell\intbl \cell\row"
        Dim Index As Integer
        Index = RichTextBox1.Rtf.LastIndexOf("}")
        RichTextBox1.Rtf = RichTextBox1.Rtf.Substring(0, Index) + table + "}"
```

20) Открытие стандартной программы Windows «Таблица символов» для вставки символа:

```
Shell("CharMap.exe")
```

21) Дополнительная процедура PrintDocument1_PrintPage() для печати текстовой области со всем содержимым:

```
Dim printFont = New Font("Arial", 12)
        ' Отрисовываем текст
        e.Graphics.DrawString(RichTextBox1.Text, printFont, Brushes.Black,
e.MarginBounds, New StringFormat())
        ' Страниц больше нет
        e.HasMorePages = False
```

22) Справка создана с использованием стандартного диалогового окна MsgBox(), а пункт меню «О программе» с помощью вызова модального окна (новой формы).

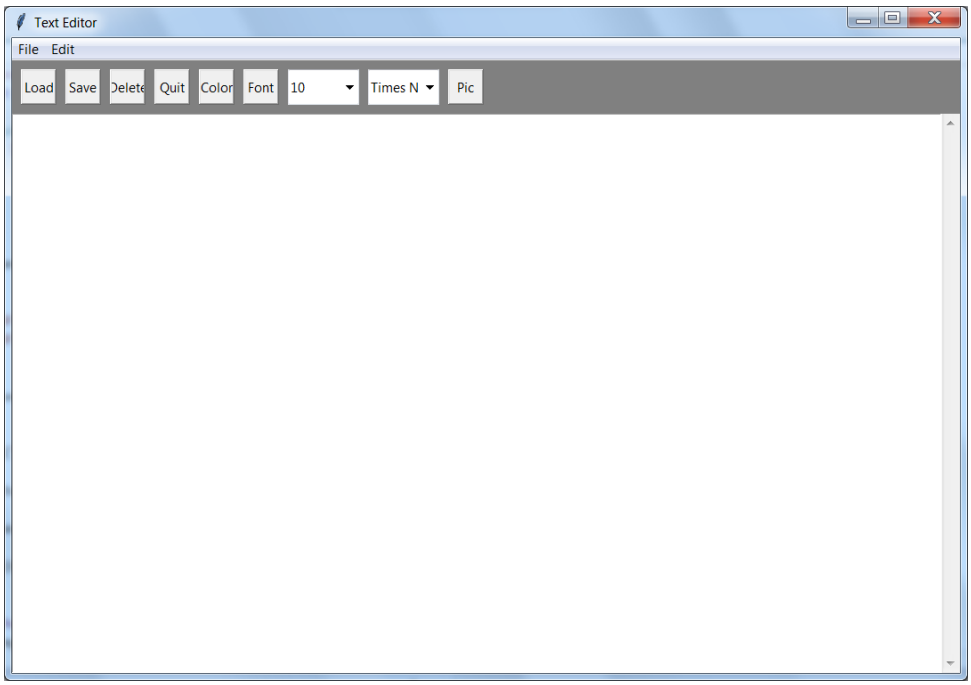
Полный код программы представлен в электронном приложении

Задача 3.2 б

Постановка задачи. Простой текстовый редактор с основными функциями.
Внешнее описание: создание, открытие и сохранение файлов формата .txt. Работа с основными функциями форматирования и вставки текста.
Функциональная спецификация: создание, открытие, сохранение файла; работа с буфером обмена, вставка и показ даты и времени; изменение шрифта и цвета, вставка рисунка.
Система программирования: Python
Особенности реализации и возникшие трудности: в стандартной библиотеке Tkinter() нет многих виджетов для диалоговых окон: печать, выбор шрифта и т.д. Эти функции можно реализовать либо через системные библиотеки Windows API, либо через другие модули Python (например, PyQt5). В данном приложении рассмотрим только выбор нескольких шрифтов из выпадающего списка, созданного вручную.
Каждая операция реализована с помощью отдельной функции, имя которой записывается в качестве имени команды для кнопки или пункта меню, или пункта контекстного меню.
Открытие файла с изображением: эта функция корректно работает только при непосредственном вводе имени файла. Если использовать диалоговое окно FileDialog, то необходимо функции реализовывать как методы внутри класса. В противном случае имя файла нельзя использовать в других частях программы, если процедура открытия была реализована в функции-обработчике.

Достоинства приложения: простой интерфейс, реализация основных функций текстового редактора.

Недостатки приложения: нет большинства функций, используемых в классических текстовых редакторов (смена шрифта и начертания, печать документа).
Разработка интерфейса. В приложении использовались следующие виджеты: многострочное текстовое поле для основной работы (Text), основное меню (Menu), стандартные диалоговые окна (FileDialog, ColorChooser), кнопки быстрого доступа (Button), выпадающие списки для имени и размера шрифта (Combobox).
Окончательно форма задачи имеет вид:



Основные команды и процедуры кода приложения:
1) Открытие файла:

```
fn = filedialog.Open(root, filetypes = [('*.txt files', '.txt')]).show()
    if fn == '':
        return
    textbox.delete('1.0', 'end')
    textbox.insert('1.0', open(fn, 'rt').read())
```

Для использования диалогового окна его необходимо импортировать из библиотеки: from tkinter import filedialog. Если выбран текстовый файл, то его открыть в режиме чтения - 'rt'.

2) Сохранение файла:

```
fn = filedialog.SaveAs(root, filetypes = [('*.txt files', '.txt')]).show()
    if fn == '':
        return
    if not fn.endswith(".txt"):
        fn+=".txt"
    open(fn, 'wt').write(textbox.get('1.0', 'end'))
```

Это же диалоговое окно с методом SaveAs() позволяет сохранить текст из области под файловым именем.

3) Удаление текста (или создание нового файла):

```
answer=messagebox.askyesno("Подтверждение", message="Вы хотите удалить текст?")
    if answer==True:
        textbox.delete(0.0, END)
```

Для этой операции используется диалоговое окно messagebox, которое аналогично добавляется в список импортируемых виджетов, к нему применяется метод askyesno(), и при положительном ответе содержимое текстовой области удаляется.

4) Смена цвета символов при помощи диалогового окна:

```
(rgb, hx) = colorchooser.askcolor()
    textbox.config(foreground=hx)
```

Если “foreground” заменить на bg (background), то изменится цвет фона текстовой области.

5) Смена шрифта:

Т.к. готового виджета диалогового окна в стандартной библиотеке не существует, то в качестве аргументов функции используются данные, заранее выбранные в поле комбинированного списка:

```
textbox.config(font=(fname.get(), fsize.get()))
```

При этом в строку импорта виджетов добавляется “font”, а в основной программе при описании виджетов добавляется Combobox, например:

```
fname=StringVar() #создание текстовой переменной
combo2 = Combobox(panelFrame)  #инициализация поля со списком внутри рамки
combo2['values'] = ('Times New Roman', 'Monotype Corsiva', 'Courier', 'Arial', 'Impact',
'Comic Sans MS')  #перечень элементов списка – названий для шрифтов
combo2.current(0) #установка курсора на первом элементе списка
fname=combo2 #присваивание выделенного элемента списка текстовой переменной
combo2.place(x=400,y=10,width=80,height=40) #размещение поля со списком на форме
```

6) Копирование текста в буфер обмена:

```
textbox.event_generate('<<Copy>>')
```

7) Добавление даты и времени к тексту (необходимо импортировать соответствующие модули):

```
textbox.insert(END, datetime.now())
```

8) Формирование контекстного меню:

```
pmenu.tk_popup(event.x_root, event.y_root)
…
pmenu = Menu(textbox)
pmenu.add_command(label='Cut', accelerator='Ctrl+X', command=cut)
…
textbox.bind('<Button-3>', popup)
```

9) Формирование пунктов основного меню, например, «Файл»:

```
menuBar = Menu(root)
fileMenu = Menu(menuBar)
fileMenu.add_command(label="New", accelerator='Ctrl+N', command=delete_text)
…
menuBar.add_cascade(label="File", menu=fileMenu)
```

10) Формирование одной из кнопок с командой (на примере открытия файла):

```
loadBtn = Button(panelFrame, text = 'Load')
loadBtn.bind("<Button-1>", LoadFile)
loadBtn.place(x = 10, y = 10, width = 40, height = 40)
```

11) Формирование текстовой области внутри рамки с полосами прокрутки (scrollbar):

```
textFrame.pack(side = 'bottom', fill = 'both', expand = 1)
textbox = Text(textFrame, font='Arial 14', wrap='word')
scrollbar = Scrollbar(textFrame)
scrollbar['command'] = textbox.yview
textbox['yscrollcommand'] = scrollbar.set
textbox.pack(side = 'left', fill = 'both', expand = 1)
scrollbar.pack(side = 'right', fill = 'y')
```

12) Вставка изображения в текстовую область:
Для работы с изображениями формата .jpg необходимо подключить дополнительную библиотеку Python Image Library (PIL):

```
from PIL import Image, ImageTk
```

В процедуре-обработчике для кнопки написать команду вставки изображения в текстовое поле в указанную позицию курсора:

```
textbox.image_create(INSERT,image=img)
```

В основной программе добавить строку с явным указанием имени файла:

```
img = ImageTk.PhotoImage(Image.open("smile.jpg"))
```

Полный код программы представлен в электронном приложении

Задача 3.2 в

Постановка задачи. Текстовый редактор с поддержкой вычислений.
Внешнее описание: создание, открытие и сохранение файлов форматов .rtf, .txt. Работа с основными функциями форматирования и вставки текста.
Функциональная спецификация: создание, открытие, сохранение, печать файла; работа с буфером обмена; изменение шрифта и цвета фона, поиск и замена слов; вывод количества совпадений, слов, строчных и прописных букв, подсчет суммы, произведения, разности, частного и среднего арифметического выделенных в тексте чисел.
Система программирования: Microsoft Visual C#.
Особенности реализации и возникшие трудности: часть функций дублируется в пунктах меню и на панели кнопок простым копированием в обработчиках событий. Операции «Открыть файл» и «Сохранить как» реализуются с помощью стандартных диалоговых окон. Для операции «Сохранить файл под существующим именем» необходимо задать переменную класса, в которой будет храниться имя ранее открытого или сохраненного файла.
Фрагмент кода:

```
string current_fn = null;
…
            current_fn = openFileDialog1.FileName;
            this.Text = $"Текстовый редактор | Файл : {current_fn}";
```

Для функции сохранения:

```
if (current_fn == null)
                toolStripButton3_Click(sender, e);
            else
            {
                if (MessageBox.Show("Сохранить?", "Запрос сохранения файла",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                    MainText.SaveFile(current_fn);
            }
```

Если имя файла – пусто (т.е. он не был открыт или сохранен), то управление передается функции-обработчику сохранения файла под именем. В данном случае команда «Сохранить как» находится на панели кнопок toolStripButton3.

Для настроек принтера и печати применяются соответствующие диалоговые окна, но в отдельной функции необходимо указать область для печати с предустановленным шрифтом текста:

```
void PrintPageHandler(object sender, PrintPageEventArgs e)
        {
            MainText.Font = new Font(MainText.Font.Name, MainText.Font.Size,
MainText.Font.Style);
            e.Graphics.DrawString(MainText.Text, MainText.Font, Brushes.Black, 10, 10);
        }
```

К сожалению, в C# нет диалоговых окон для поиска и замены текста и инструментов для ввода значений (по типу InputBox), поэтому для реализации этих операций используется 3 основных способа:

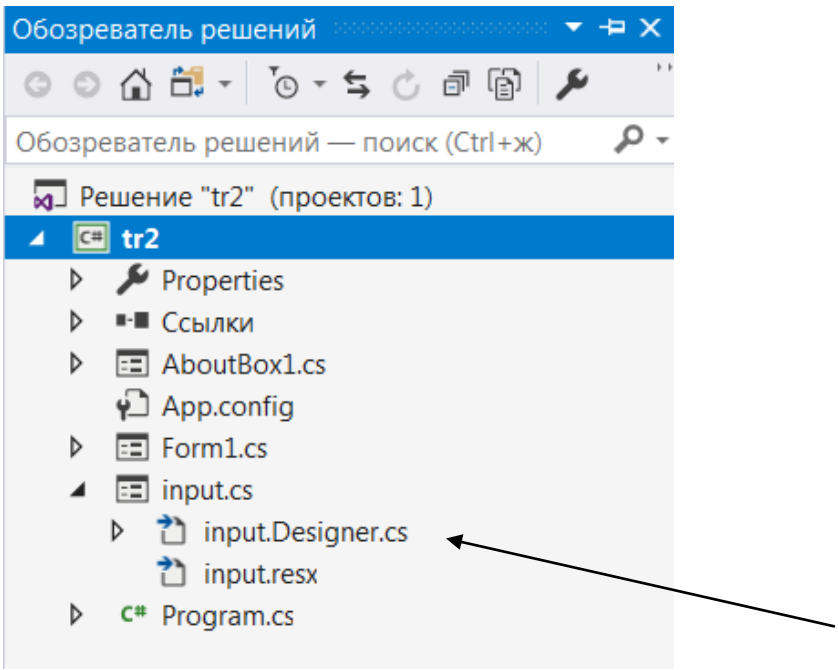
1) Подключить соответствующий модуль из Visual Basic.

2) Создать пользовательский класс с необходимыми элементами управления и методами.

3) Создать новую форму и разрешить обмен данными между основной формой и созданной.

Воспользуемся в данной задаче последним способом.

После создания формы (в примере она называется input) перейти в обозреватель решений и выбрать файл для изменения input.Designer.cs:



Компоненты, служащие для обмена с другой формой, объявить, как public:

```
public System.Windows.Forms.TextBox textBox1;
```

Теперь содержимое текстового поля доступно для основной формы. Метод ShowDialog() позволяет вывести на экран вторую форму в качестве модальной, т.е. автоматически расположенной поверх экрана. Закрытие модальной формы не влияет на основную.

Рассмотрим фрагмент кода с поиском слова в тексте:

```
string txt;
input input1 = new input(); //инициализация формы с вводом слова
input1.Owner = this; //передача прав собственника текущей (основной) форме
input1.ShowDialog(); //открытие модальной формы
txt = input1.textBox1.Text; //присваивание значения из поля модальной формы
MainText.Find(txt); //встроенный метод поиска
```

Для замены производятся аналогичные действия, только либо создается новая модальная форма с двумя текстовыми полями, либо созданная форма вызывается два раза. Затем для замены применяется метод replace().

Работа с буфером обмена аналогична, как в других системах. Например:

```
MainText.Copy();
```

Работа с изменением начертания символов производится с помощью подобных методов:

```
SetSelectionFont(FontStyle.Bold);
```

Рассмотрим подробно функции подсчета в тексте:

1) Количество совпадений среди слов:

```
string[] arrString = MainText.Text.Split(); //массив строк с любым разделителем из текстового поля
            var a = from aa in arrString group aa by aa.Substring(aa.IndexOf(aa)); //выделение одинаковой подстроки из каждой строки
            string output = "";
            foreach (var item in a)
            {
                output += $@"{item.Key} встречается: {item.Count()}{Environment.NewLine}";
```

```
                /*интерполяция строк @$. Этот символ позволяет указывать переменные,
окруженные фигурными скобками, прямо в строках без использования конкатенации или
форматирования.*/
            }
            MessageBox.Show(output, "Количество совпадений");
```

2) Количество слов в тексте:

```
string str;
            int k = 0;
            str = MainText.Text;
            string[] words = str.Split(new char[] { },
                StringSplitOptions.RemoveEmptyEntries); //преобразование текста в массив
строк с удалением пустых строк
            foreach (string s in words) //цикл по элементам
            {
                k++;
            }

            MessageBox.Show($"{k}", "Количество слов");
```

3) Количество прописных букв в тексте (фрагмент функции):

```
foreach (char bukva in stroka)
            {
                if (Char.IsUpper(bukva))
                    lb++;
}
```

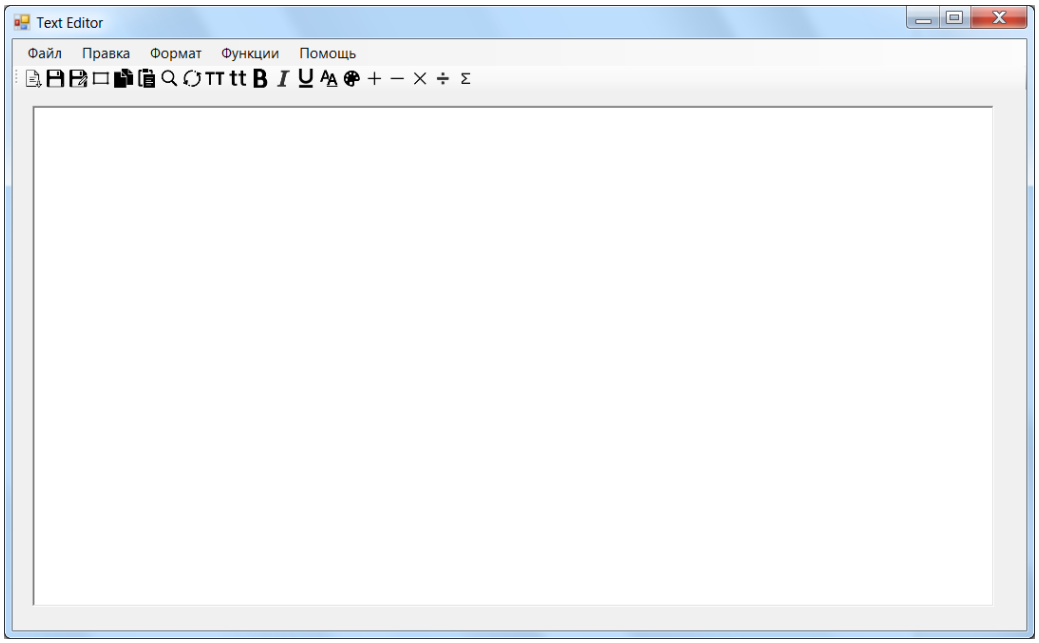
4) Арифметические операции (на примере сложения):

```
try
            { decimal result = array[0];
                for (int i = 1; i < array.Count; i++)

                    result += array[i];
                    MessageBox.Show($" {result}", "Сумма");

            }
            catch (Exception)
            {
            }
```

Предварительно необходимо объявить массив, в котором будут храниться выбранные числа, и разрешить множественное выделение. Числа из текста выделаются с помощью комбинации клавиши Ctrl и движения мыши, поэтому весь метод выделения числа из текста и помещение его в числовой массив осуществляется с помощью соответствующих функций:
- MainText_KeyDown() – разрешение множественного выделения и определения цвета выделения;
- MainText_MouseUp() – добавление при нажатой клавиши ctrl выделенного числа в массив;
- MainText_MouseClick() – восстановление первоначальных значений и обнуление массива.

Достоинства приложения: простой интерфейс, реализация основных и дополнительных функций текстового редактора с подсчетом числовых значений.

Недостатки приложения: нет контекстного меню.
Разработка интерфейса. В приложении использовались следующие компоненты: многострочное текстовое поле для основной работы (RichTextBox=MainText), основное меню (MenuStrip), панель кнопок (toolStrip - button), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog, FontDialog, ColorDialog, PrintDialog, PageSetupDialog, PrintDocument.
Окончательно форма задачи имеет вид:



Для добавления пункта меню «О программе» использовалась встроенная форма. Она вызывается с помощью пункта: Проект – Добавить форму Windows – Окно «О программе». Данные для этой формы подставляются из окна: Проект – Свойства – Приложение – Сведения о сборке…
Подгружается это модальное окно в основной форме с помощью команды:

```
new AboutBox1().ShowDialog();
```

Полный код программы представлен в электронном приложении

Задача 3.3

Постановка задачи. Создать простой словарь-справочник терминов языка программирования.
Внешнее описание: по щелчку на названии термина отображается его описание в текстовом поле.
Функциональная спецификация: задано ограниченное количество терминов непосредственно в настройках элемента управления или в коде обработчика. Для удобства чтения информации реализованы 2 кнопки увеличения и уменьшения размера шрифта. Курсор выделенного элемента имеет нестандартный шаблон цветов.
Система программирования: Lazarus.

Особенности реализации и возникшие трудности: эта программа используется в демонстрационных целях, т.к. не имеет возможности добавлять и удалять термины с сохранением информации. Для корректной работы параметров изменения размера шрифта и цвета необходимо заранее в Инспекторе объектов установить необходимые свойства компонентов Listbox и Memo.

Memo:

Свойство	Значение	Пояснение
ScrollBars	ssVertical	Вертикальная полоса прокрутки
WordWrap	true	Разрешить перенос текста по строкам
Font.Name	Times New Roman	Название шрифта
Font.Size	10	Размер шрифта

Listbox:

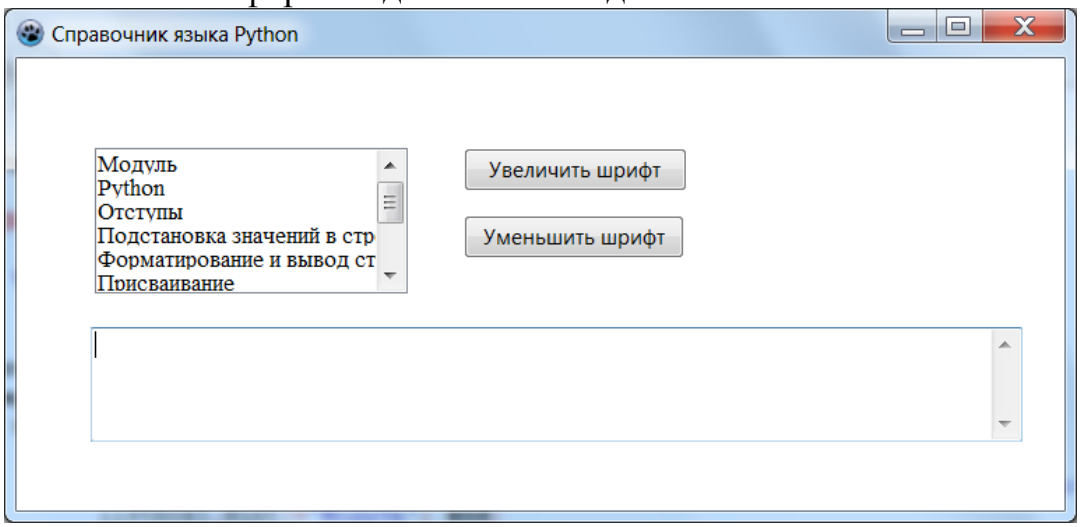
Свойство	Значение	Пояснение
Font.Name	Times New Roman	Название шрифта
Font.Size	10	Размер шрифта
ItemHeight	17	Высота ячейки
ShowHint	true	Отображать всплывающую подсказку
Options.Style	lbOwnerDrawFixed	Каждый элемент списка имеет фиксированную высоту, но способ его отображения определяется программистом
Items	(TStrings): Модуль Python Отступы Подстановка значений в строку Форматирование и вывод строк Присваивание Логические условия Цикл For Функции	Ввод терминов в ячейки – элементы списка

Достоинства приложения: простой понятный интерфейс, изменение размера символов для чтения.

Недостатки приложения: невозможность добавления или удаления терминов.

Разработка интерфейса. В приложении использовались следующие компоненты: список для терминов (Listbox), многострочное текстовое поле для вывода (Memo), кнопки изменения размера текста (Button)

Окончательно форма задачи имеет вид:



В программе использовались 4 основные процедуры:

1) ListBox1Click – обработчик нажатия кнопки мыши на списке.

Фрагмент кода:

```
if ListBox1.Selected[0] then
begin
   Memo1.Lines[0]:= 'Совокупность описаний, объединенных в общее пространство имен - глобальное пространство модуля.';
      listbox1.Hint:='Модуль';
end;
```

Если выбран элемент списка с номером «0» (первый элемент), то вывести соответствующее сообщение в первую строку текстового поля и над этим элементом поместить при наведении мыши всплывающую подсказку с названием термина.

Аналогичные условные операторы для каждого элемента списка с терминами.

2) ListBox1DrawItem – процедура изменения внешнего вида элементов списка.

Фрагмент кода:

```
txt:=listbox1.Items[Index]; //поместить значение из выбранного элемента списка в текстовую переменную для дальнейшей обработки
     if (odSelected in State) then // условие: цвет строки ячейки в состоянии
        begin
         listbox1.Canvas.Brush.Color:=clGreen; //цвет фона ячейки - зеленый
         listbox1.Canvas.Font.Color:=clWhite; //цвет текста ячейки - белый
        end;
     listbox1.Canvas.FillRect(ARect); //создание прямоугольной области
  listbox1.Canvas.TextOut(ARect.Left, ARect.Top, txt)    // отрисовка текста с заданными параметрами в графическом виде
```

3) Button1Click – обработчик кнопки увеличения размеров шрифта.

Фрагмент кода:

memo1.Font.Size:=memo1.font.size+4; //размер шрифта текстового поля увеличивается на 4 по сравнению с предыдущим
listbox1.Font.Size:=listbox1.Font.Size+4; //размер шрифта списка увеличивается на 4 по сравнению с предыдущим
listbox1.ItemHeight:=listbox1.ItemHeight+5; //размер высоты каждой ячейки списка увеличивается на 5 по сравнению с предыдущим

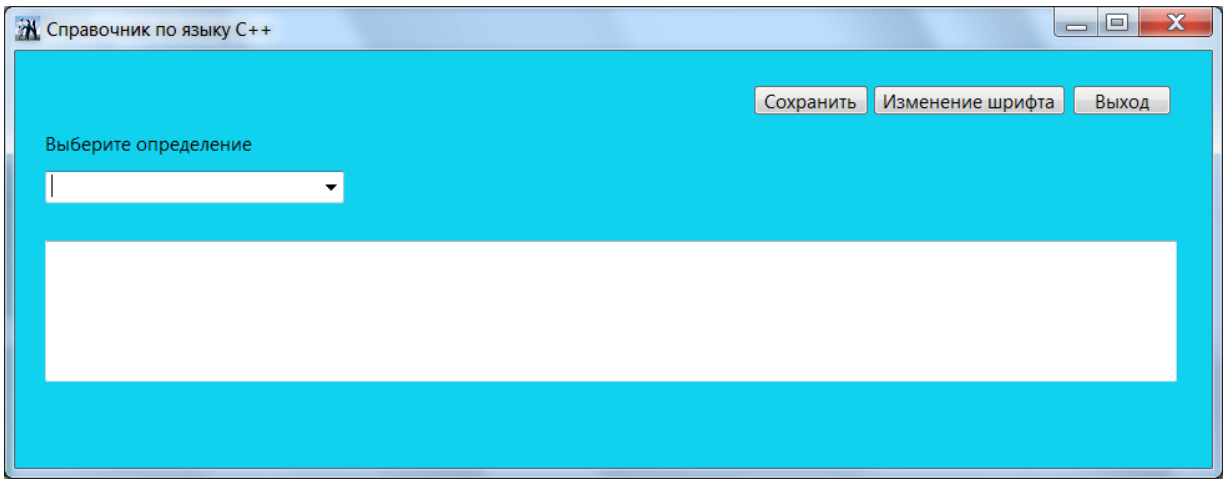
4) Button2Click – обработчик кнопки уменьшения размеров шрифта.
Код аналогичен предыдущему, но со знаком «минус».

Полный код программы представлен в электронном приложении

Задача 3.3 а

Постановка задачи. Изменим предыдущую задачу таким образом, чтоб термины выбирались из выпадающего списка, а их определения помещались в область вывода из текстового файла.
Внешнее описание: по выбору термина из списка отображается его описание в текстовом поле.
Функциональная спецификация: задано ограниченное количество терминов непосредственно в настройках элемента управления или в текстовом файле. Для удобства чтения информации реализована кнопка изменения шрифта с помощью стандартного диалогового окна. Выданное определение можно сохранить в текстовом файле.
Система программирования: Lazarus.
Особенности реализации и возникшие трудности: для начала необходимо подготовить текстовый файл, в котором количество строк с определениями равно количеству терминов в выпадающем списке. При сохранении указать кодировку utf-8. Затем добавить в свойство Items компонента Combobox нужное количество строк с терминами.
В программе используется одна основная процедура - ComboBox1Select и вспомогательные процедуры – обработчики кнопок для смены шрифта и сохранения определения.

Достоинства приложения: простой понятный интерфейс, изменение размера символов для чтения.
Недостатки приложения: невозможность добавления или удаления терминов.
Разработка интерфейса. В приложении использовались следующие компоненты: список для терминов (Combobox), многострочное текстовое поле для вывода (Memo), вспомогательное поле статического текста (Label), кнопки изменения шрифта текста, сохранения и выхода (Button), стандартное диалоговое окно смены шрифта (TFontDialog) и сохранения в файл (TSaveDialog).
Окончательно форма задачи имеет вид:



Алгоритм основной процедуры имеет вид:
1) Создать массив строк TStringList.
2) Загрузить в него полностью файл с определениями.
3) Связать строковую переменную со строкой файла по индексу выбранного термина в списке.
4) Вывести полученную строку в текстовом поле.
5) Очистить память массива строк.
Соответствующий фрагмент кода имеет вид:

```
tfile:= TStringList.Create;
    tfile.LoadFromFile('list2.txt');
  str:= tfile.Strings[combobox1.itemindex];
  Memo1.Text:= str;
    tfile.Free;
```

Рассмотрим код процедуры смены шрифта в списке и поле вывода:

```
if FontDialog1.Execute then
  begin
memo1.Font.Assign(FontDialog1.Font);
combobox1.Font.assign(FontDialog1.Font);
  end;
```

Код процедуры сохранения файла из текстового поля:

```
if SaveDialog1.Execute then memo1.lines.SaveToFile(SaveDialog1.FileName);
```

Полный код программы представлен в электронном приложении

Задача 3.3 б

Постановка задачи. Модифицируем предыдущую задачу таким образом, чтоб термины выбирались из выпадающего списка, а их определения помещались в область вывода из заранее созданных текстовых файлов.
Внешнее описание: по выбору термина из списка отображается его описание в текстовом поле.
Функциональная спецификация: задано ограниченное количество терминов непосредственно в настройках элемента управления или в текстовом файле.

Для удобства чтения информации реализована кнопка изменения шрифта с помощью стандартного диалогового окна.
Система программирования: Visual Basic.
Особенности реализации и возникшие трудности: список терминов загружается в компонент Combobox при загрузке формы на экран, т.е. в процедуре Form1_Load по образцу:

```
ComboBox1.Items.Add("Prolog - это ")
```

Для уменьшения количества строк использовалась дополнительная процедура открытия файла и загрузки его содержимого в текстовое поле:

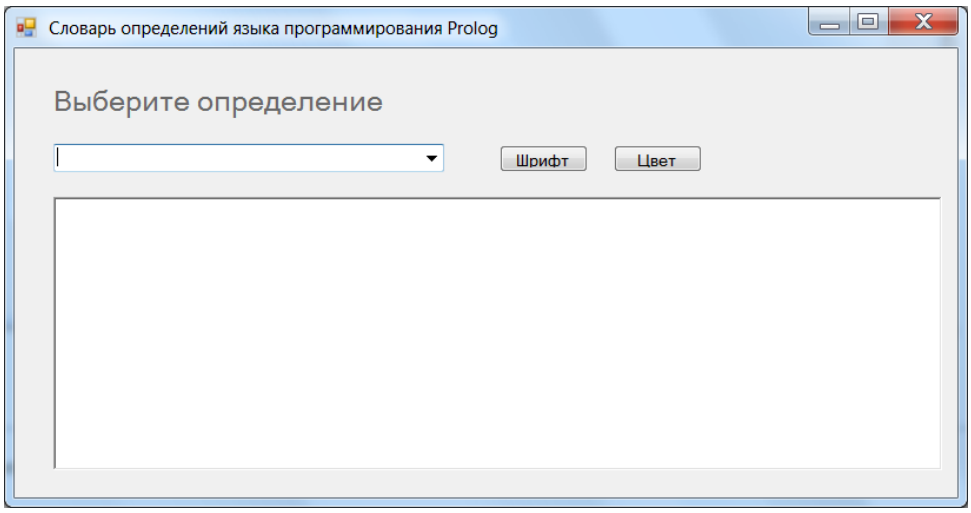
```
Private Sub filechange(ByVal filename As String)
        RichTextBox1.Clear()
        RichTextBox1.LoadFile(filename, RichTextBoxStreamType.PlainText)
    End Sub
```

Вывод информации из файла по нажатию на элемент выпадающего списка осуществляется с помощью процедуры-обработчика ComboBox1_SelectedIndexChanged.
В отличие от предыдущих задач условие вывода записывается не через условный оператор if, а через селектор case.
Фрагмент кода:

```
Select Case ComboBox1.SelectedIndex
            Case 0
                filechange("1.txt")
…
```

Вывод информации из разных файлов позволяет расширить определение и использовать программу в качестве полноценной энциклопедии.
Аналогично предыдущим задачам в приложении используются кнопки изменения шрифта и цвета формы с помощью стандартных диалоговых окон.

Достоинства приложения: простой понятный интерфейс, изменение размера символов для чтения.
Недостатки приложения: невозможность добавления или удаления терминов.
Разработка интерфейса. В приложении использовались следующие компоненты: список для терминов (Combobox), многострочное текстовое поле для вывода (RichTextBox), вспомогательное поле статического текста (Label), кнопки изменения шрифта текста, сохранения и выхода (Button), стандартное диалоговое окно смены шрифта (FontDialog) и изменения цвета окна (ColorDialog).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.3 в

Постановка задачи. Справочник терминов из нескольких языков программирования с возможностью выбора языка.

Внешнее описание: по выбору языка из списка отображается список терминов во втором выпадающем списке. А по выбору термина из второго списка происходит вывод определения в текстовую область.

Функциональная спецификация: задано ограниченное количество терминов в текстовом файле. В остальных файлах производится соответственно занесение терминов в качестве элементов списка, либо вывод определений.

Система программирования: Visual C#.

Особенности реализации и возникшие трудности: количество языков в первом файле и количество файлов для второго списка и текстового поля должно контролироваться программистом (или пользователем) вручную, т.е. для каждого языка должен быть один файл с терминами и один файл с определениями, а количество строк в файле с определениями должно совпадать с количеством терминов.

Список языков загружается в выпадающий список из файла lang.txt построчно при загрузке формы в функции Form1_Load. Фрагмент кода:

```
string S1;
            StreamReader lang = new StreamReader("lang.txt");
            comboBox1.BeginUpdate();
            while ((S1 = lang.ReadLine()) != null)
            {
                comboBox1.Items.Add(S1);
            }
            lang.Close();
            comboBox1.EndUpdate();
```

Для корректной работы с выпадающим списком применяются методы BeginUpdate() и EndUpdate(), указывающие соответственно на начало и окончание формирования списка. Для этого сначала вызывается первый метод BeginUpdate(). После завершения задачи добавления элементов в список вызывается метод EndUpdate(), который перерисовывает ComboBox. Этот

способ добавления элементов позволяет избежать мерцания ComboBox при добавлении большого числа элементов в список.
В функции comboBox1_SelectedIndexChanged происходит добавление терминов во второй список в соответствии с выбором языка из первого списка. Только программист может менять количество элементов в операторе-селекторе вида:

```
switch (i)
{
case 0:  files = "pas.txt"; break;
…
```

В этой же функции термины отображаются в списке в соответствии с файлом терминов:

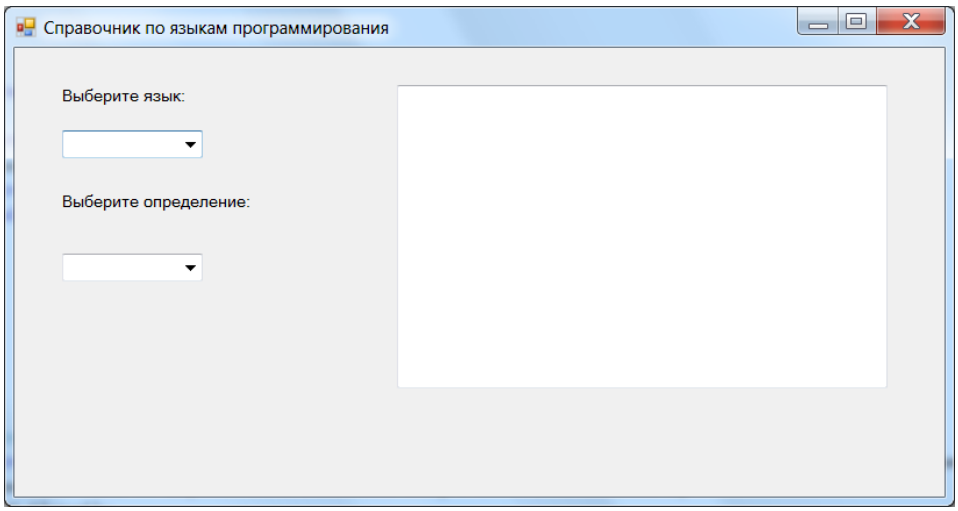
```
StreamReader lang0 = new StreamReader(files);
            comboBox2.BeginUpdate();
            while ((s2 = lang0.ReadLine())!= null)
            {

                comboBox2.Items.Add(s2);
            }
            lang0.Close();
            comboBox2.EndUpdate();
```

Аналогичная функция SelectedIndexChanged создается и для компонента comboBox2, только данные из файла записываются не в список, а в текстовый блок:

```
StreamReader sr2 = new StreamReader(files);
            string[] db = sr2.ReadToEnd().Split(Environment.NewLine.ToCharArray(),
StringSplitOptions.RemoveEmptyEntries);
            textBox1.Text += db[comboBox2.SelectedIndex] + Environment.NewLine;
            sr2.Close();
```

Здесь количество терминов и их определений внутри существующих файлов может менять и пользователь.

Достоинства приложения: простой понятный интерфейс, возможность добавления или удаления терминов.
Недостатки приложения: невозможность добавления или удаления языков программирования.
Разработка интерфейса. В приложении использовались следующие компоненты: списки для языков и терминов (Combobox), многострочное текстовое поле для вывода (TextBox), вспомогательное поле статического текста (Label).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.3 г

Постановка задачи. Справочник терминов с возможностью редактирования словаря пользователем (добавление, удаление и сохранение терминов и их описаний из файла)

Внешнее описание: по выбору термина из списка отображается его описание в текстовом поле.

Функциональная спецификация: задано ограниченное количество терминов и описаний в текстовом файле. По нажатию кнопки из файла термины загружаются в поле списка. По двойному щелчку мыши на элементе списка в текстовое поле загружается определение из второго файла. Кроме того, элементы списка можно удалять, а полученный список сохранять в тот же файл.

Система программирования: Python.

Особенности реализации и возникшие трудности: в программе используются три функции, вызываемые нажатием соответствующей кнопки, и функция-обработчик двойного щелчка мыши с основным алгоритмом.

Рассмотрим фрагменты кода:

1) Добавление элементов из текстового файла в список:

```
with open('list000.txt', 'r') as file:
        lst = file.readlines()
    for item in lst:
        lbox.insert(END, item)
```

2) Удаление элемента из списка с сохранением индекса (для этого список переворачивают, чтоб индексы удалялись с конца списка):

```
select = list(lbox.curselection())
    select.reverse()
    for i in select:
        lbox.delete(i)
```

3) Сохранение списка в текстовый файл:

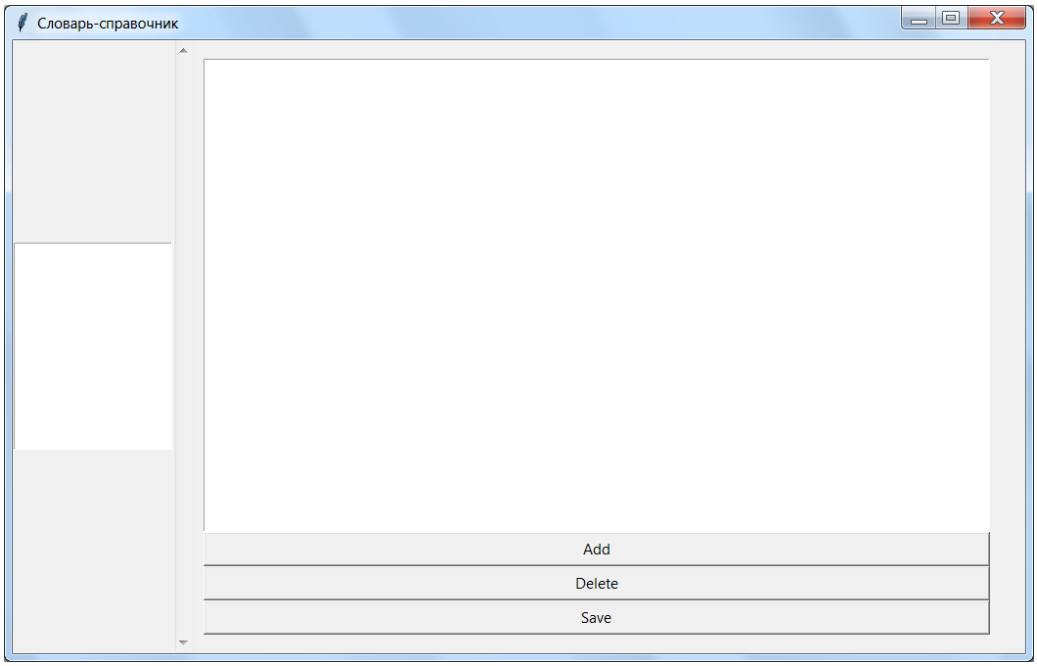
```
f = open('list000.txt', 'w')
    f.writelines("\n".join(lbox.get(0, END)))
```

4) Загрузка определения в текстовую область в соответствии с выбранным элементом списка:

```
sel=lbox.curselection()
entry.delete("0.0",END)
with open('def.txt', 'r') as file:
        lst1 = file.readlines()
entry.insert("0.0",lst1[sel[0]])
```

Т.к. метод curselection() возвращает индекс выделенного элемента в виде кортежа (tuple), например, (0,), то его нужно преобразовать в число, взяв элемент этого кортежа с индексом «0». При добавлении элементов из файла считывается строка с тем же номером.

Достоинства приложения: простой понятный интерфейс, возможность добавления или удаления терминов.
Недостатки приложения: невозможность добавления или удаления определений, несогласованность файлов.
Разработка интерфейса. В приложении использовались следующие виджеты: список для терминов (Listbox=lbox), многострочное текстовое поле для вывода (Text=entry), кнопки добавления, удаления и сохранения терминов (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.4

Постановка задачи. Создание тестирующей программы для проверки знаний. Вариант №1. Каждый вопрос находится в отдельной вкладке.
Внешнее описание: по выбору термина из списка отображается его описание в текстовом поле.

Функциональная спецификация: создается коллекция вкладок, на которой размещаются элементы выбора и кнопки. Выбрав правильный ответ, пользователь перемещается к следующей вкладке. На последней вкладке происходит подсчет результатов.
Система программирования: Visual Basic.
Особенности реализации и возникшие трудности: расположение элементов в каждой вкладке дублируется, что приводит к избыточности кода. С другой стороны, алгоритм программы упрощается:
1) При нажатии на радиокнопку с правильным ответом происходит увеличение счетчика ответов на единицу. Счетчик – это глобальная целочисленная переменная, задаваемая в классе формы.
2) При нажатии на кнопку «Далее» вне зависимости от результата осуществляется переход на новую вкладку.
3) При нажатии на кнопку «Выход» на любом этапе происходит выход из программы.
4) В последней вкладке происходит вывод результата в диалоговом окне.

Рассмотрим фрагменты кода данного алгоритма.
Пример пунктов 1 и 2:

```
If RadioButton1.Checked = True Then
            k = k + 1
        End If
        TabControl1.SelectedIndex = 1
```

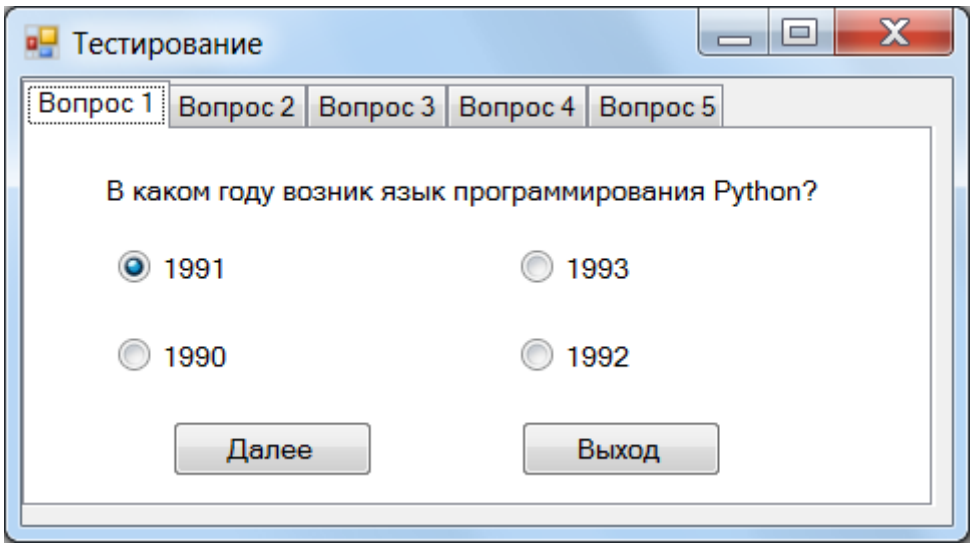
Пример пункта 3:

```
Private Sub Button2_Click(sender As System.Object, e As System.EventArgs) Handles
Button2.Click
        End
    End Sub
```

Пример пункта 4:

```
Dim s As String
If k = 5 Then s = "Отлично"
If k = 4 Then s = "Хорошо"
If k = 3 Then s = "Удовлетворительно"
If k < 3 Then s = "Неудовлетворительно"
MsgBox("Вы ответили правильно на " + CStr(k) + " вопроса." + vbLf + "Ваша оценка: " + s)
```

Достоинства приложения: простой понятный интерфейс, сосредоточенность только на текущем вопросе.
Недостатки приложения: избыточность кода, ограниченное количество вкладок.
Разработка интерфейса. В приложении использовались следующие компоненты: коллекция вкладок (TabControl), статический текст для пояснений (Label), радиокнопки выбора ответа (RadioButton), кнопки управления тестом (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.4 а

Постановка задачи. Создание тестирующей программы для проверки знаний. Вариант №2. Все вопросы и варианты ответов находятся на одной форме. После прохождения теста в текстовом окне по нажатию кнопки высвечивается оценка.

Внешнее описание: каждый вопрос с вариантами ответа размещается на отдельной панели, кнопка «Оценка» неактивна до тех пор, пока в текстовом поле не введено имя.

Функциональная спецификация: В программе предусмотрена защита на повторное нажатие радиокнопки с вариантом ответа во избежание неверного результата. В текстовом поле «оценка» показывается не только значение, но и количество правильных ответов, а также уровень знаний.

Система программирования: Visual C#.

Особенности реализации и возникшие трудности: подсчет правильных ответов можно осуществить двумя способами:

1) через переменную-счетчик наращивания для каждого нажатия радиокнопки (как в предыдущей задаче);

2) сопоставление правильного варианта с элементом массива.

Первый способ быстрей компилируется, но код программы занимает много места из-за повторяющихся операторов. Второй способ работает медленней из-за инициализации массивов, но код программы намного короче, поэтому воспользуемся вторым способом.

Для этого создадим два числовых массива – один пустой с будущими номерами ответов, второй заполненный с правильными вариантами ответов, например: answersRight = new int[] { 1, 2, 2, 3, 4 }.

В функции обработки кнопки подсчета результата переменная-счетчик увеличивается внутри цикла:

```
for (int i = 0; i < answers.Length; i++)
            {
                if (answers[i] == answersRight[i])
```

```
                {
                    countRightA++;
                } }
```

В этой же функции производится подсчет количества баллов по схеме, которое затем выводится в текстовую область:

```
double level = (double)countRightA / answers.Length * 100;
            string grade = "";
            if (level <= 50)
                grade = "2";
```

Функция обработки щелчка по радиокнопке упрощена, благодаря тому, что можно обращаться к родительскому классу RadioButton, из которого выделяется необходимый индекс. Например, числа в названии radioButtonQ1Answer1 означают вопрос №1 и ответ №1.

Фрагмент кода:

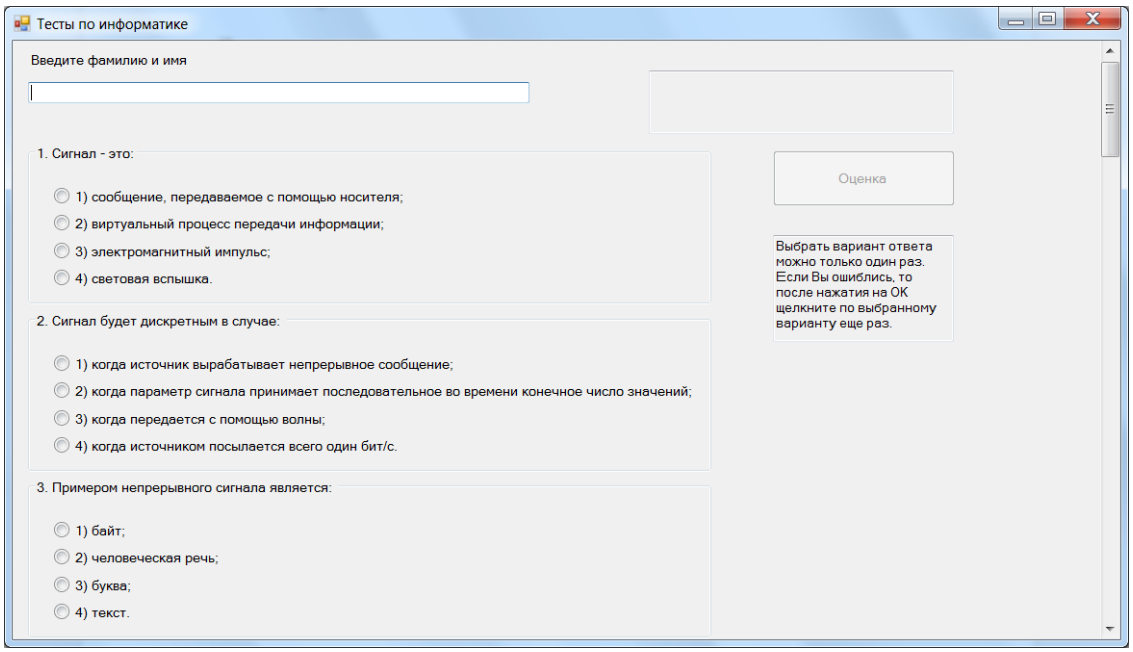
```
int indexQ = Convert.ToInt32((((RadioButton)sender).Parent.Name).Substring(16)); //
выделение из имени радиокнопки номера вопроса
            int indexA = Convert.ToInt32((((RadioButton)sender).Name).Substring(12 +
indexQ.ToString().Length + 6)); // выделение из имени номера ответа
            if (answers[indexQ - 1] != 0)
            {
                MessageBox.Show("Выбирать вариант можно только один раз!"); //защита от
повторного нажатия
                answers[indexQ - 1] = 0;
                ((RadioButton)sender).Checked = false;
            }
            else
            {
                answers[indexQ - 1] = indexA; //заполнение массива номеров ответов
индексами нажатых радиокнопок.
            }
```

Достоинства приложения: одно окно для всех вопросов, возможность проходить тест не по порядку, защита от повторного нажатия.

Недостатки приложения: для большого количества вопросов приходится использовать полосы прокрутки.

Разработка интерфейса. В приложении использовались следующие компоненты: текстовые поля для ввода данных и вывода результатов (TextBox), панели для вопросов и ответов (GroupBox), радиокнопки выбора ответа (RadioButton), кнопка вывода результата (Button).

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.4 б

Постановка задачи. Создание тестирующей программы для проверки знаний. Вариант №3. Все вопросы и варианты ответов находятся на одной форме, но результат необходимо вписывать вручную в текстовое поле.

Внешнее описание: каждый вопрос с вариантами ответа размещается на отдельной панели. При нажатии кнопки «Итог» появляется количество правильных ответов, балл в процентах и оценка прописью.

Функциональная спецификация: после внесения ответа в текстовое поле, оно становится неактивным для редактирования. Кнопка «Итог» активна после ответа хотя бы на первый вопрос.

Система программирования: Lazarus.

Особенности реализации и возникшие трудности: такой вариант теста подходит для небольшого количество вопросов, чтобы окно можно разместить на видимой части экрана монитора. В системе Lazarus можно использовать полосы прокрутки, но зачастую они работают некорректно.

Программа состоит из необходимого количества однотипных процедур изменения текста в блоке записи варианта ответа, например, для первого блока:

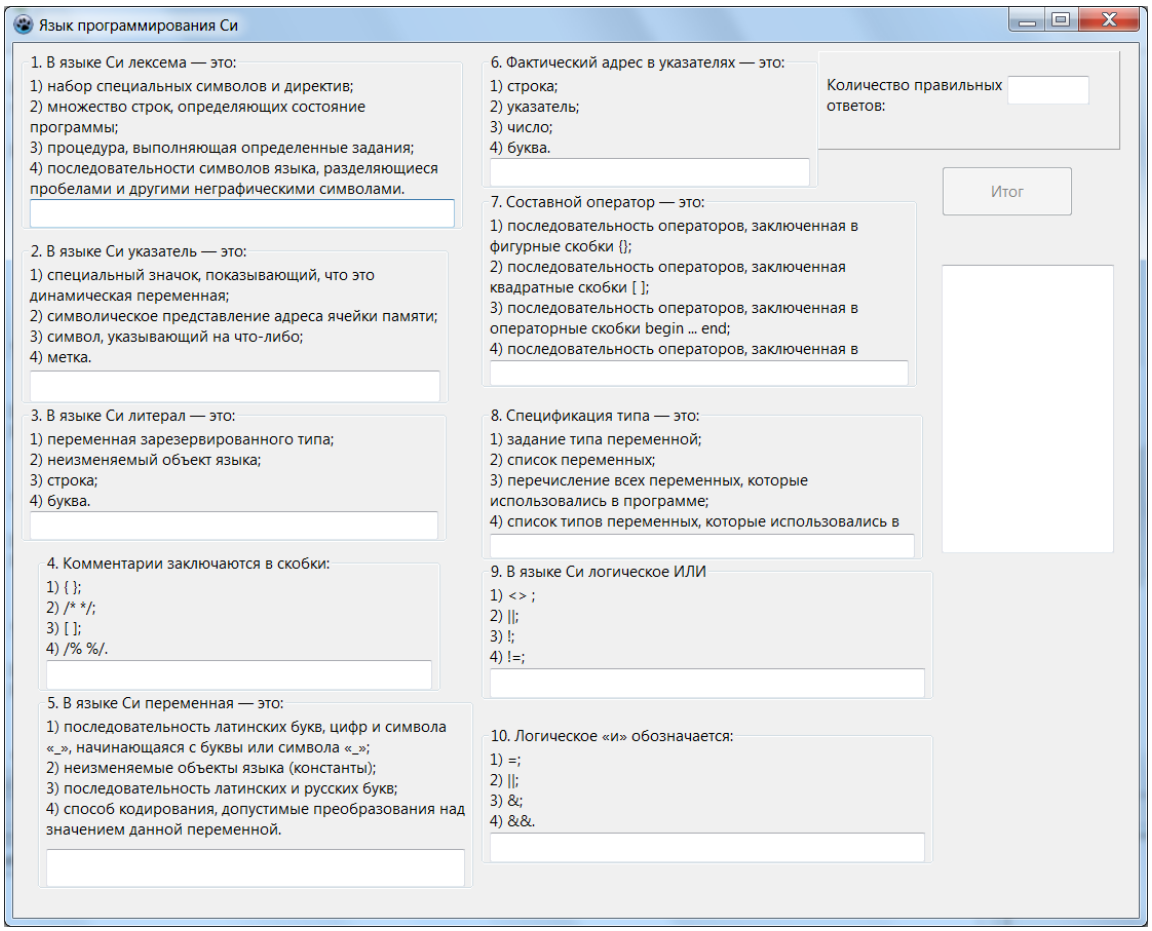
```
procedure TForm1.Edit1Change(Sender: TObject);
begin
 if Edit1.Text ='1'   then inc(k);
 if edit1.Text<>'' then edit1.Enabled:=false;
 button1.enabled:=true;
end;
```

Остальные блоки «вопрос-ответы» кодируются аналогично, только команда активации кнопки не нужна.

Перед началом работы необходимо задать свойство button1.enabled:=false или в настройках (инспекторе объектов), или в процедуре загрузки формы FormCreate.
Рассмотрим фрагмент кода подсчета результата, который записывается в процедуре-обработчике кнопки:

```
Edit29.Caption:=inttostr(k);
mark:=k*10;
memo1.Lines.Add('Ваша оценка:');
if mark<51 then s:=inttostr(mark)+'% - неудовлетворительно';
if (mark>=51) and (mark<76) then s:=inttostr(mark)+'% - удовлетворительно';
if (mark>=76) and (mark<94) then s:=inttostr(mark)+'% - хорошо';
if (mark>=94) then s:=inttostr(mark)+'% - отлично';
memo1.lines.add(s);
```

В этом коде переменная k – глобальная (количество правильных ответов из каждой процедуры обработки текста), а переменные mark и s – локальные (вспомогательные, используемые для вывода оценки).
Достоинства приложения: одно окно для всех вопросов, возможность проходить тест не по порядку, защита от повторного нажатия.
Недостатки приложения: для большого количества вопросов приходится использовать полосы прокрутки.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовые поля для ввода данных и вывода результатов (Edit, Memo), группы и панели для вопросов и ответов (RadioGroup, Panel), поля с перечнем вариантов ответов (Label), кнопка вывода результата (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 3.4 в

Постановка задачи. Создание тестирующей программы для проверки психологических качеств человека (тест Майерса-Бриггса).

Вариант 4. Варианты ответов написаны на кнопках. Отдельная кнопка для подведения итогов теста.

Внешнее описание: этот тест для установления типа личности содержит всего 4 вопроса по 2 варианта ответа в каждом, поэтому их лучше разместить на одной форме.

Функциональная спецификация: после нажатия на кнопку ответа меняется цвет фона, а вторая кнопка становится неактивной. Результат появляется только при ответе на все 4 вопроса.

Система программирования: Visual Basic.

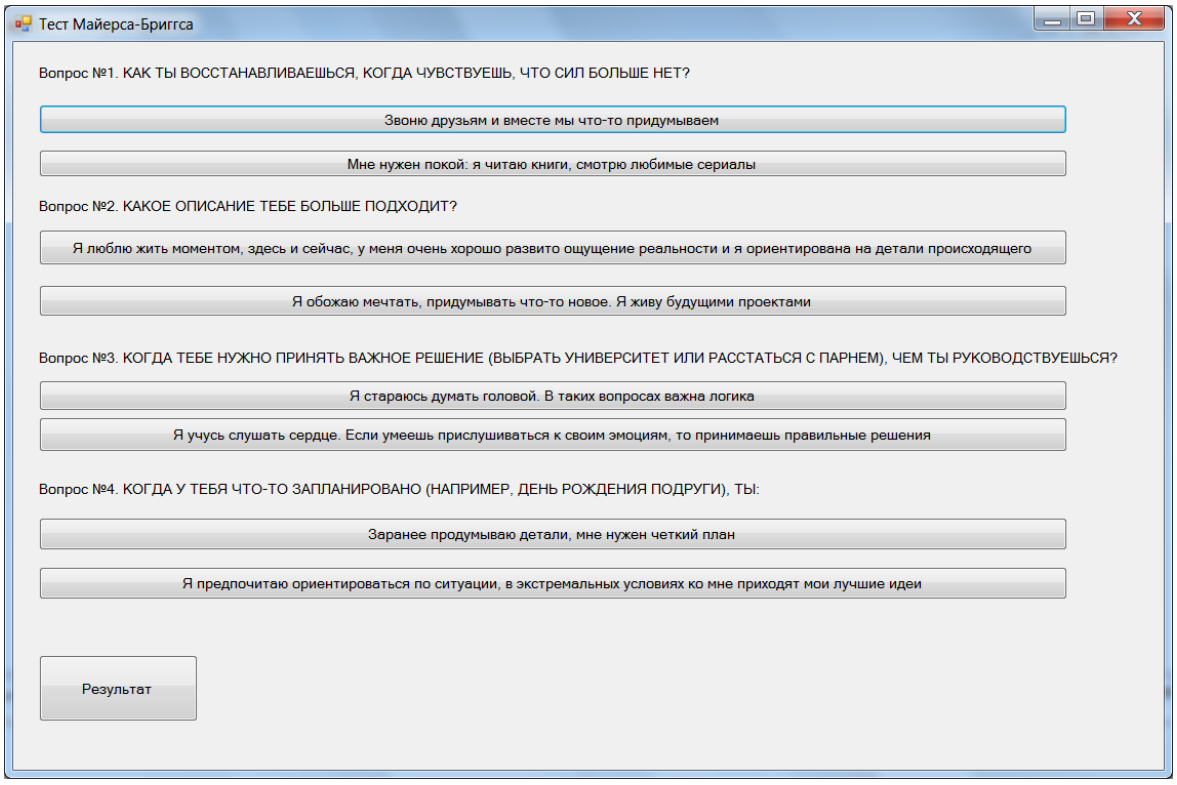
Особенности реализации и возникшие трудности: реализация программного кода зависит от проведения тестирования. При ответе на вопрос каждому варианту сопоставляется буква латинского алфавита (одна из двух), на выходе получаем комбинацию из четырех букв, которая интерпретируется в результат тестирования. Всего возможно 16 различных комбинаций.

Таким образом, коды обработчиков каждой из кнопок вопросов имеют вид (один из примеров):

```
s = s + "n"
```

```
Button4.BackColor = Color.Green
Button3.Enabled = False
```
По кнопке-обработчике результата сначала формируется значение переменной с толкованием, например:
```
If s = "istj" Then r = "ИНСПЕКТОР…"
```
Затем это значение помещается в текстовую область вывода:
```
Label5.Text = r
```

Достоинства приложения: одно окно для всех вопросов, защита от повторного нажатия.
Недостатки приложения: в программе используется только один вид тестирования.
Разработка интерфейса. В приложении использовались следующие компоненты: статические текстовые поля для вывода вопросов и результатов (Label), кнопка вывода результата и кнопки с перечнем вариантов ответов (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Глава 4. Работа с графической информацией.

Работа с графикой – одна из самых интересных задач как для программистов, так и для пользователей (художников, дизайнеров). Сделать поздравительную открытку или коллаж, нарисовать эскиз, просмотреть фотографию. Практически все современные системы программирования предоставляют доступ к открытию графических файлов и изображению графических примитивов (точки, прямой, кривой, сложной геометрической фигуры).

Задача 4.1

Постановка задачи. Создание открытки, состоящей из надписи и фонового рисунка.

Внешнее описание: Из предложенных вариантов (или из текстового файла) формируется текст поздравления, из графического файла добавляется фон. Затем все вместе можно сохранить как иллюстрацию в файле.

Функциональная спецификация: для заголовка и связанного с ним текста поздравления можно менять цвет и шрифт. Фоновый рисунок растягивается на видимую область окна. Результат сохраняется в виде графического файла.

Система программирования: Microsoft Visual Basic.

Особенности реализации и возникшие трудности: программа корректно работает, если размер рисунка для фона примерно совпадает с размером окна программы, в противном случает часть текста отсутствует.

Все основные функции (кроме сохранения открытки) реализуются в пунктах меню. А сохранение – по нажатию динамической кнопки, которая появляется на экране после выбора пункта «Готово».

Рассмотрим основные функции:

1) Заголовок открытки выбирается из пункта с соответствующим названием. Код каждого пункта имеет вид:

```
Private Sub СДнемРожденияToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles СДнемРожденияToolStripMenuItem.Click
        Label1.Text = СДнемРожденияToolStripMenuItem.Text
    End Sub
```

2) Выбор фона из графического файла осуществляется с помощью стандартного диалогового окна открытия файла. Фрагмент кода:

```
OpenFileDialog1.Filter = "JPEG|*.jpg|BMP|*.bmp|GIF|*.gif|PNG |*.png"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
            PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
        End If
```

3) Текст поздравления либо выбирается из готового шаблона с подстановкой заголовка:

```
s = Label1.Text
        s = s.Replace("!", " ")
        TextBox1.Text = s + "поздравляю!" & vbNewLine
```

Либо из файла:

```
If OpenFileDialog1.ShowDialog() Then
```

```
            Dim read = New System.IO.StreamReader(OpenFileDialog1.FileName,
System.Text.Encoding.GetEncoding(1251))
            TextBox1.Text = read.readtoend()
            read.close()
        End If
```

4) Выбор цвета и шрифта для текста производится с помощью двух стандартных диалоговых окон:

```
If ColorDialog1.ShowDialog() Then Label1.ForeColor = ColorDialog1.Color
If FontDialog1.ShowDialog() Then TextBox1.Font = FontDialog1.Font
```

5) Самый сложный код для пункта меню «Готово» - отрисовка нового изображения с текстом поверх изображения, скрытие ненужных элементов управления, добавления кнопки для сохранения в файл. Фрагменты кода:

```
Label1.Font = New Font(Label1.Font.Name, Label1.Font.Size, Label1.Font.Style) ‘установка
параметров шрифта для заголовка
Dim s As Size = TextRenderer.MeasureText(txt, Label1.Font) ‘установка размера
прямоугольной области в соответствии с занимаемым текстом
Format.Alignment = StringAlignment.Center ‘задает выравнивание строки текста по центру
Using g As Graphics = Graphics.FromImage(PictureBox1.Image) ' создаем полотно рисования
на основе картинки и далее настраиваем качество отрисовки
     g.TextRenderingHint = Drawing.Text.TextRenderingHint.AntiAlias ‘параметры
сглаживания текста
     g.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
     g.DrawString(txt, Label1.Font, New SolidBrush(Label1.ForeColor), New
RectangleF(Label1.Width / 4, Label1.Height / 3, s.Width, s.Height), Format) ' рисуем
текст
     g.DrawString(TextBox1.Text, TextBox1.Font, New SolidBrush(TextBox1.ForeColor), New
RectangleF(Label1.Width / 4, TextBox1.Height / 3, t.Width, t.Height), Format)
        End Using
Dim Button As New Button
        Button.Name = "Button1"
        Button.Text = "Сохранить"
        Button.Size = New Size(150, 30) 'Размеры
        Button.Location = New Point(10, 100)
        AddHandler Button.Click, AddressOf ButtonClick
        PictureBox1.Controls.Add(Button) ‘создание кнопки с заданными параметрами и
помещение ее поверх изображения
```

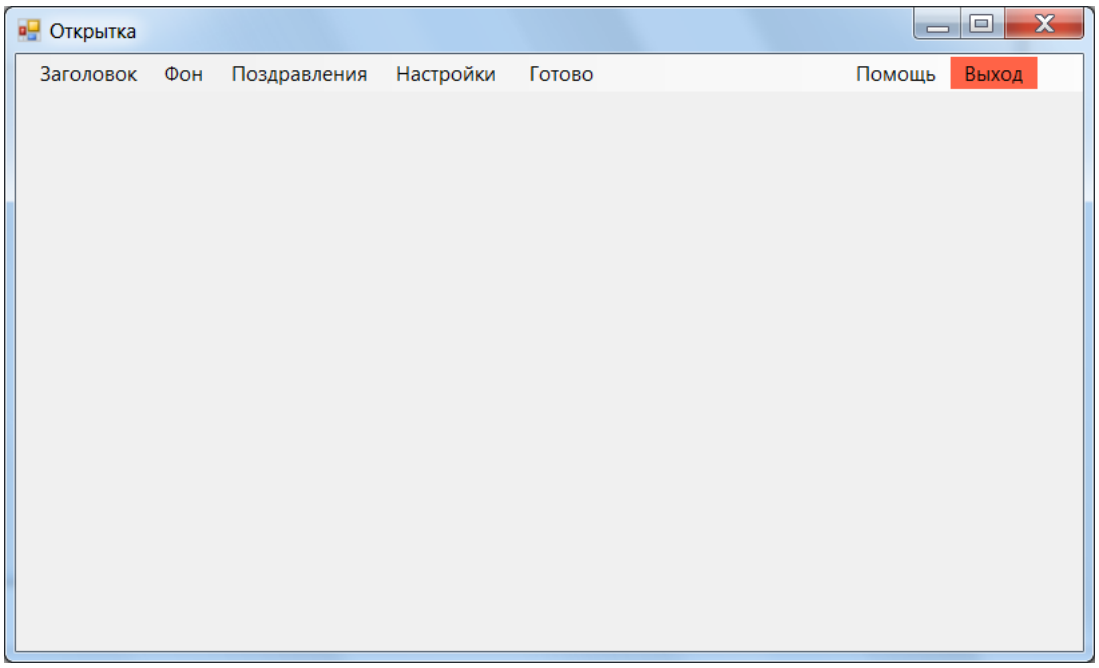
6) Сохранение файла в обработчике созданной кнопки:

```
Private Sub ButtonClick(sender As Object, e As System.EventArgs)
        SaveFileDialog1.Filter = "JPEG|*.jpg | BMP|*.bmp | GIF|*.gif |PNG |*.png"
        If SaveFileDialog1.ShowDialog() = DialogResult.OK Then
PictureBox1.Image.Save(SaveFileDialog1.FileName)
    End Sub
```

Еще одна особенность программы – изменение размеров текстовой и графической области в соответствии с изменениями размеров формы осуществляется в настройках необходимого компонента, например, PictureBox – Anchor – Top, Left, Right, Down.

Достоинства приложения: простой понятный интерфейс, возможность менять шрифт и цвет текстовых областей, сохранение открытки.

Недостатки приложения: нет контроля для размера графического файла фона, при неправильном выборе файла или размера текста общий вид открытки может отличаться от задуманного.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для вывода текста и заголовка (TextBox, Label), основное меню команд (MenuStrip), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog), программно созданная кнопка для сохранения результата (Button).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.1 а

Постановка задачи. Изменим предыдущую задачу, устранив часть недостатков. Для этого выбор иллюстраций для фона выбирается из заранее определенного списка файлов одинакового размера.
Внешнее описание: В отдельном окне записывается текст поздравления, с помощью кнопок со списком выбираются иллюстрации для фона и оформления. Затем все вместе можно сохранить в графическом файле.
Функциональная спецификация: для текста поздравления можно менять цвет и шрифт. С помощью кнопок со списком можно выбрать иллюстрацию из предложенных для оформления открытки. Результат сохраняется в виде графического файла.
Система программирования: Microsoft Visual Basic.
Особенности реализации и возникшие трудности: для реализации выпадающей из списка кнопки с изображением необходимо, во-первых, заранее открыть файл с изображением, во-вторых, создать процедуру появления на экране нужного пункта и закрытия всех остальных.

Фрагмент объявления переменных класса:

```
Dim цветок As New Bitmap("цветок.png")
Dim фон1 As New Bitmap("Фон 1.jpg")
```

Фрагмент процедуры обработки опускания кнопки мыши для выбора нужного элемента и закрытия остальных:

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        If ToolStripButton2.Checked = True Then
            'закрываем остальные кнопки при нажатии на необходимую
            справка.Checked = False
            ToolStripButton2.Checked = False
            ГлазToolStripMenuItem.Checked = False
…
Label1.Visible = False
            Form3.RichTextBox1.Text = ""
            Form3.ShowDialog()

            Гр.DrawString(Label1.Text, Label1.Font, цвет, e.X, e.Y)
            Граф.DrawString(Label1.Text, Label1.Font, цвет, e.X, e.Y)
        End If
```

В данном примере происходит обработка нажатия на кнопку вставки текста и закрытия остальных кнопок. При этом открывается модальная форма form3. Набранный текст записывается на изображение в позицию курсора мыши.

```
If КотёнокToolStripMenuItem.Checked = True Then
            'закрываем остальные кнопки при нажатии на необходимую
            справка.Checked = False
            ToolStripButton2.Checked = False
            ГлазToolStripMenuItem.Checked = False
…
Гр.DrawImage(котенок, e.X, e.Y)
            Граф.DrawImage(котенок, e.X, e.Y)
        End If
```

В этом фрагменте той же процедуры осуществляется вывод графического файла в позицию курсора мыши.

Аналогично открывается последующее изображение с закрытием остальных. При загрузке собственного изображения фрагмент кода той же процедуры имеет вид:

```
If ЗагрузкаСвоегоИзображения.Checked = True Then
            If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.Cancel Then Exit
Sub
            Гр.DrawImage(New Bitmap(OpenFileDialog1.FileName), e.X, e.Y)
            Граф.DrawImage(New Bitmap(OpenFileDialog1.FileName), e.X, e.Y)
            ЗагрузкаСвоегоИзображения.Checked = False
…
```

Окно для изображения и канва для вывода рисунка формируется в процедуре загрузки формы:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Лист = New Bitmap(Me.Width, Me.Height)
        Гр = Graphics.FromImage(Лист)
        Граф = Me.CreateGraphics
    End Sub
```

Сохранение результата аналогично предыдущей программе (4.1).

Обработчик кнопки «Выход»:

```
Dim n As Integer
        n = MsgBox("Вы действительно хотите выйти?", MsgBoxStyle.YesNo +
MsgBoxStyle.Question, "Выход из программы")
        If n = vbYes Then End
```

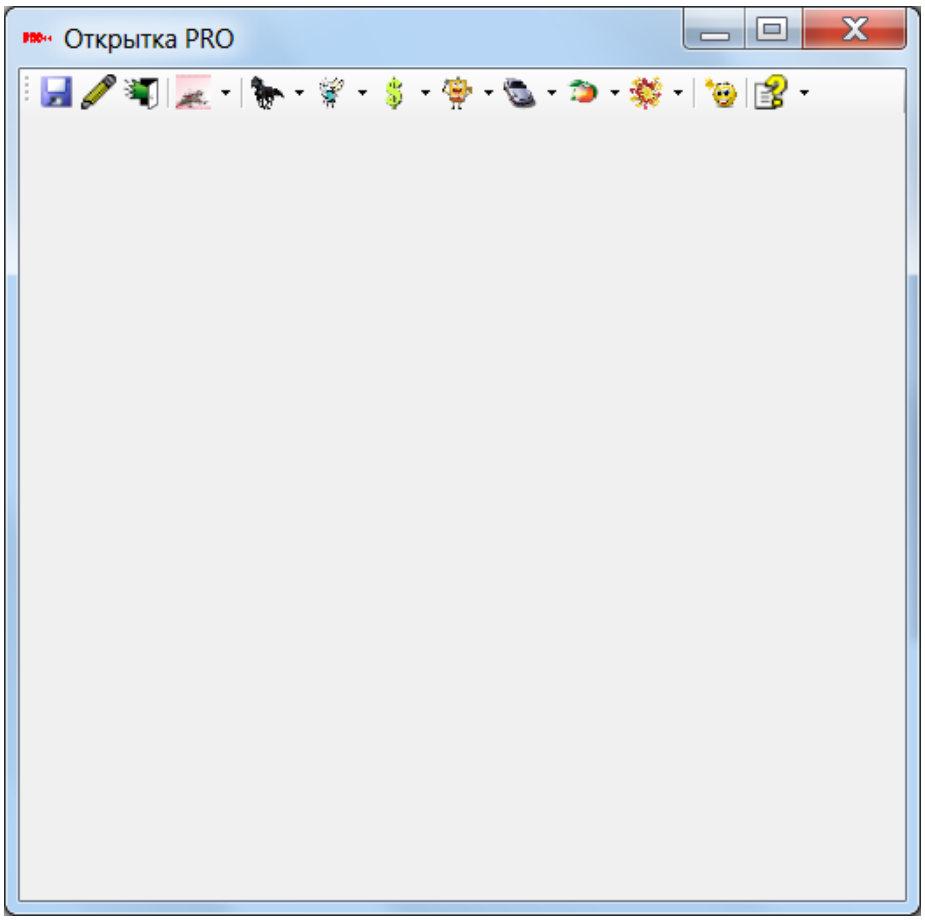
Обработчик выбора фона по нажатию кнопки:

```
Private Sub Фон1ToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Фон1ToolStripMenuItem.Click
        Гр.DrawImage(фон1, 0, 0)
        Граф.DrawImage(фон1, 0, 0)
    End Sub
```

Обработчик кнопки «Справка» - вызов файла помощи:

```
Help.ShowHelp(Me, "Help.chm")
```

Достоинства приложения: простой понятный интерфейс, возможность менять шрифт текстовой области, дополнительные изображения для украшения, сохранение открытки.
Недостатки приложения: фиксированный размер графического файла фона, нет предустановленных заголовков и шаблонов текста, избыточный код для открытия и закрытия графических элементов.
Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для вывода текста (RichtextBox, Label), основное меню кнопок (toolStrip), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.2

Постановка задачи. Создать фотоколлаж – изображение, созданное из нескольких изображений.

Внешнее описание: В отдельном окне выбирается макет коллажа, затем по очереди загружаются 3 изображения из файла. Затем все вместе можно сохранить в общем графическом файле.

Функциональная спецификация: макет можно выбрать по нажатию на соответствующую кнопку.

Система программирования: Lazarus.

Особенности реализации и возникшие трудности: программа состоит из четырех окон: на первой форме выбирается макет, а в формах 2-4 происходит обработка изображений по макету.

Обработчик кнопки имеет следующий вид (пример для первой кнопки):

```
Form2.Show;
  form1.Visible:=false;
```

Вторая и последующие формы состоят из элементов TImage, расположенных в нужном порядке, и кнопок с открытием и сохранением файла.

Код открытия трех файлов имеет вид:

```
if OpenDialog1.Execute then
Image1.Picture.LoadFromFile(OpenDialog1.FileName);
  if OpenDialog1.Execute then
```

```
  Image2.Picture.LoadFromFile(OpenDialog1.FileName);
 if OpenDialog1.Execute then
  Image3.Picture.LoadFromFile(OpenDialog1.FileName);
```

Обработчик кнопки «Сохранить» формирует область окончательного изображения и имя сохраненного файла с номером по порядку:

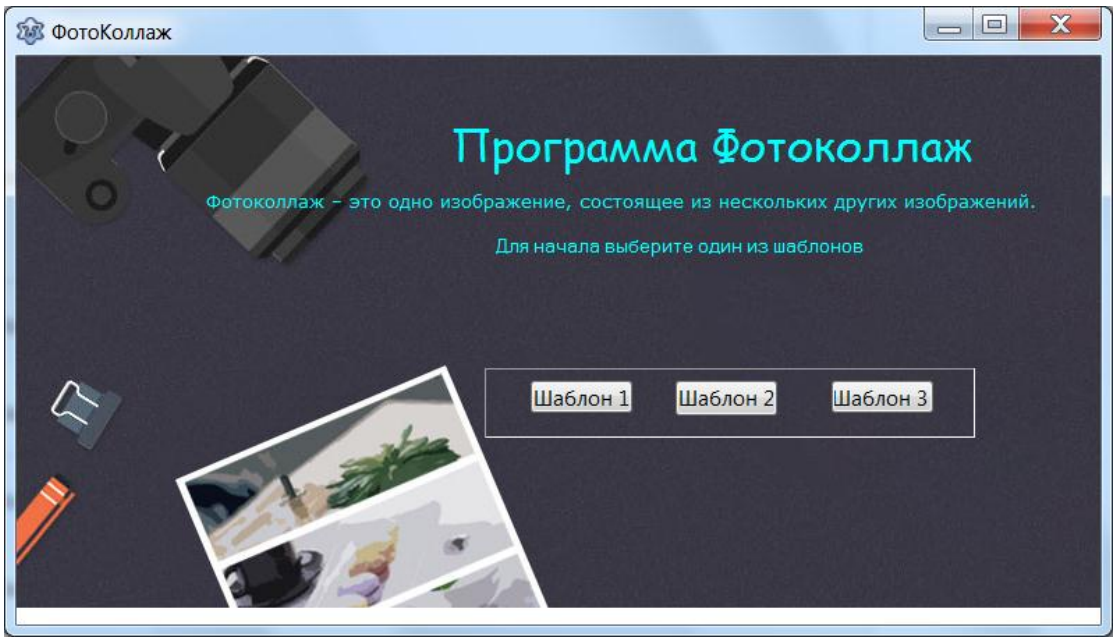
```
var
  b: TBitmap;
    i: integer;
begin
  i := 10; // Устанавливаем счетчик
  b := TBitmap.Create(); // Создаем объект типа  TBitmap
  b.Width := Form2.ClientWidth; // Устанавливаем ширину изображения
  b.Height := Form2.ClientHeight; // Устанавливаем высоту изображения
  b.Canvas.CopyRect(Rect(0, 0, b.Width, b.Height), Form2.Canvas,
    Form2.ClientRect);    // Копируем канву формы в наш TBitmap
  while FileExists(InttoStr(i) + '.bmp') do       // Генерируем уникальное имя
    inc(i);
  b.SavetoFile(InttoStr(i) + '.bmp');             // Сохраняем картинку
  b.Free;
end;
```

Достоинства приложения: простой понятный интерфейс, три вида макета для коллажа.

Недостатки приложения: избыточные формы для макетов, невозможность добавления собственных макетов, нет регулировки размещения изображения в графической области.

Разработка интерфейса. В приложении использовались следующие компоненты: текстовое поле для пояснений (Label), кнопки для работы с коллажем (Button), стандартные диалоговые окна (OpenDialog, SaveDialog), графические области для изображений (Image).

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.2 а

Постановка задачи. Создать фотоколлаж – изображение, созданное из нескольких изображений, используя только одно окно.

Внешнее описание: В общем окне загружается нужное количество изображений из файла, которые автоматически выстраиваются в прямоугольную область. Затем все вместе можно сохранить в общем графическом файле.

Функциональная спецификация: после загрузки всех необходимых файлов нужно нажать на кнопку «Создать коллаж». После этого можно подгружать дополнительные файлы или очищать окно с предыдущим коллажем.

Система программирования: Microsoft Visual C#.

Особенности реализации и возникшие трудности: основная проблема заключается в размере графических файлов для коллажа. Для эстетического восприятия желательно брать файлы одного размера или подгонять размеры заранее по ширине области вывода.

Загруженные файлы и их имена формируются в список List, который объявляется как переменная класса:

```
public   List<Bitmap> imag = new List<Bitmap>();// список изображений, аналогично массиву
     public List<string> img_name = new List<string>();// список имен
     public List<Bitmap> rescale_image = new List<Bitmap>(); // список изоображений с
подогнанными размерами, аналогично массиву
     public   Bitmap s_image = null;// готовое изображение коллажа
     public int counter = 0;//счётчик
```

Функция обработки кнопки «Загрузить» состоит из двух частей:

1) Открытие диалогового окна выбора файла:

```
OpenFileDialog dlg = new OpenFileDialog();
            if(dlg.ShowDialog() == DialogResult.OK)
```

```
            {
                if (s_image != null)
                {
                    s_image.Dispose(); // освобождаем если переменная не пустая
                }
```

2) Добавление изображения к списку изображений:

```
picBox.Image = (Bitmap)Image.FromFile(dlg.FileName);
                Bitmap btnmap = new Bitmap(Image.FromFile(dlg.FileName));
                imag.Add(btnmap);// добавление изображения в список
                img_name.Add(dlg.FileName);
                pic_choose.Items.Add(img_name[counter]);
                counter++;
```

Дополнительная функция выбора изображения по индексу из списка:

```
private void pic_choose_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (pic_choose.Items.Count < 1)
            { return; }
            else
            {//измененние изображения по выбору из PictureBox
                picBox.Image = imag[pic_choose.SelectedIndex];
                picBox.Update();
            } }
```

Обработчик кнопки создания коллажа:

```
if (imag.Count < 1)
            { return; }   //если в списке нет изображений, то выйти
     Bitmap final_image = new Bitmap (picBox.Width, picBox.Height);
        Graphics g = Graphics.FromImage(final_image);
        double H_scale = 1;// коофициент маштаба по высоте
        double W_scale = 1;//коофициент маштаба по ширине
            int _H=1;
            int _W=1;
            foreach(Bitmap image in imag)
            {
                _W += image.Width;
                _H += image.Height; //изменение размеров в цикле списка
            }
            if (_W > picBox.Width || _H > picBox.Height)
            {
                W_scale =(double) picBox.Width / _W;
                H_scale=(double)picBox.Height/_H;
            }
//Заполняем новый лист с измененными по размеру копиями оригинальных картинок
        foreach (Bitmap image in imag)
        {
            rescale_image.Add(new Bitmap(image, new
Size((Int32)Math.Ceiling(Convert.ToDouble(image.Width * W_scale)),
(Int32)Math.Ceiling(Convert.ToDouble(image.Height * H_scale)))));
        }
        // верхний левый угол картинки
        int x = 0;
        int y = 0;
        foreach (Bitmap image in rescale_image)
        {
            //рисуем изображение
            g.DrawImage(image,new Point(x,y));
            x+=image.Width;
            if (x > (picBox.Width / 2) && y < picBox.Height / 2) //подгонка размеров
картинок в коллаже
            {
                x = 0;
```

```
                y += image.Height; }
        }
picBox.Image = final_image;
        picBox.Update();
        s_image = new Bitmap(final_image);
        final_image.Dispose();
        g.Dispose();
        rescale_image.Clear();
```

Обработчик кнопки сохранения коллажа в файле:

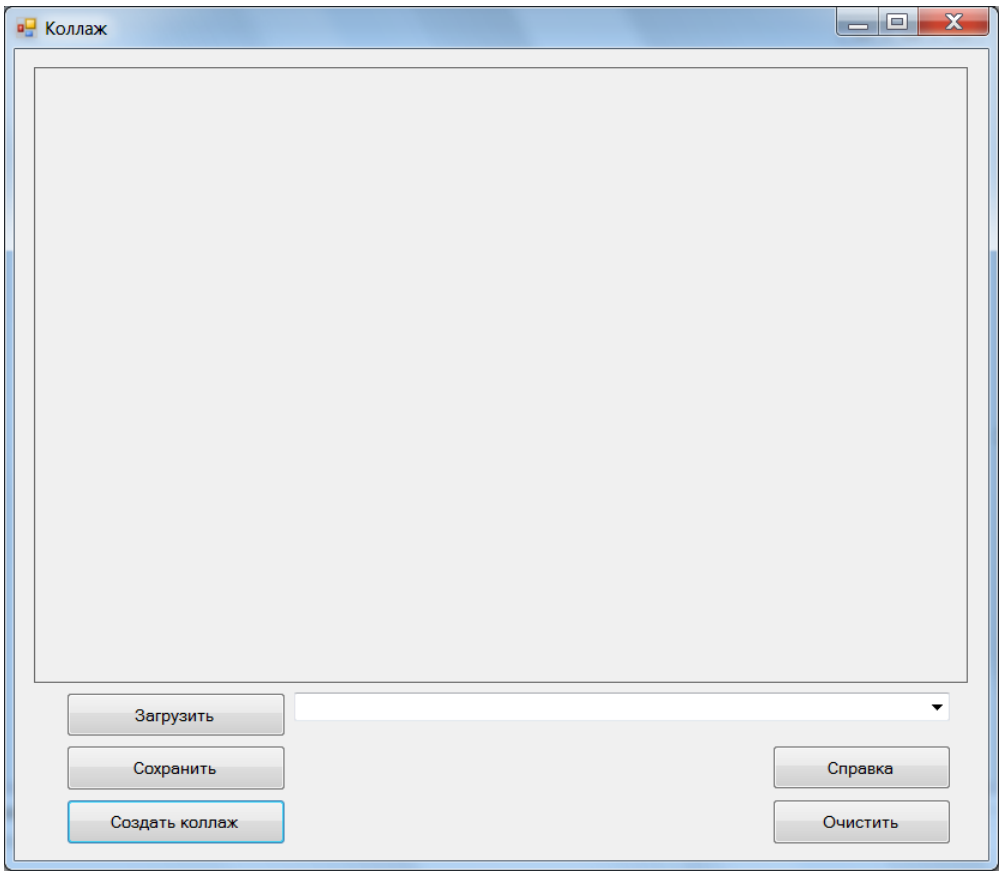
```
if (s_image != null)
            {
                SaveFileDialog dlg = new SaveFileDialog();
                dlg.Filter = "(*.PNG)|*.png|(*.JPEG)|*.jpeg| (*.BMP)|*.bmp";
            if (dlg.ShowDialog() == DialogResult.OK)
            { s_image.Save(dlg.FileName);
            MessageBox.Show("Сохранить в :\n " + dlg.FileName);
            } }
```

Функция очистки графической области содержит один метод:

```
picBox.Image = null;
```

Вывод справки реализуется любым способом, например, через стандартное окно сообщений MessageBox.Show().

Достоинства приложения: простой понятный интерфейс, любое количество изображений в коллаже с последующим сохранением.
Недостатки приложения: нет регулировки размера и размещения изображения в графической области.
Разработка интерфейса. В приложении использовались следующие компоненты: раскрывающийся список для показа имен открытых файлов (comboBox=pic_choose), кнопки для работы с коллажем (Button), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog), графическая область для изображений (pictureBox=PicBox).
Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.3

Постановка задачи. Калькулятор цвета – определение кода по цвету.
Внешнее описание: при выборе цвета из палитры печатается его название, десятичный код RGB, шестнадцатеричный код, яркость цвета.
Функциональная спецификация: цвет выбирается по нажатию кнопки с помощью диалогового окна цветовой палитры Windows. Он отображается на отдельной панели и в ячейке таблицы. В текстовых полях отображаются его коды, а во второй ячейки таблицы – общепринятое название. В программе можно вывести до 10 цветов.
Система программирования: Lazarus.
Особенности реализации и возникшие трудности: функция разложения цвета на RGB-составляющие в Lazarus отличается от соответствующей функции в Delphi. Вместо Edit1.Text:=IntToStr(GetRValue(c)) необходимо воспользоваться функцией RedGreenBlue(C, R, G, B), а затем написать строку Edit1.Text:= IntToStr(R);
При этом переменная с должна иметь тип TColorRef, который подключается в дополнительном модуле Uses LCLType.
Формула для определения яркости цвета RGB:
Luminance (вариант 1): 0.3*R + 0.59*G + 0.11*B.
Для упрощения вывода в программе ограничились 10 строками таблицы, после чего кнопка выбора цвета из палитры становится недоступной.

Количество описаний цветов в программе ограничено несколькими значениями, для остальных цветов в таблице выводится их шестнадцатеричный код.

Рассмотрим основные функции программы, реализованные в процедурах – обработчиках кнопок и загрузке формы.

1) Выбор и обработка цвета (фрагмент основной процедуры):

```
var R, G, B : Byte;
  k: string;
  c : TColorRef;
  s:string;
begin
  if ColorDialog1.Execute then
  begin
        Panel1.Color := ColorDialog1.Color;
        color1 := ColorDialog1.Color;
        c:= ColorToRGB(ColorDialog1.Color);
        RedGreenBlue(c, R, G, B);
        Edit1.Text:= 'Red: ' + IntToStr(R) +   ' Green: ' + IntToStr(G) +  ' Blue: ' + IntToStr(B);
        Edit2.Text :=  IntToHex(R, 2) + IntToHex(G, 2) + IntToHex(B, 2) ;
        Edit3.Text := FloatToStr(R*0.3 + G*0.59 + B*0.11);
        k:=edit2.text;
        case k of
        '000000': s:='черный';
        …
        'FFFFFF': s:='белый'
        else
        s:=edit2.text
        end;
        StringGrid1.Cells[1,i] := s;
        StringGrid1.Canvas.Brush.Color := color1;
        StringGrid1.Canvas.FillRect(Rect(0, i*25, 200, (i + 1)*25));
        inc(i);
       if i>9 then button1.Enabled:=false;
  end;
  end;
```

2) Загрузка формы:

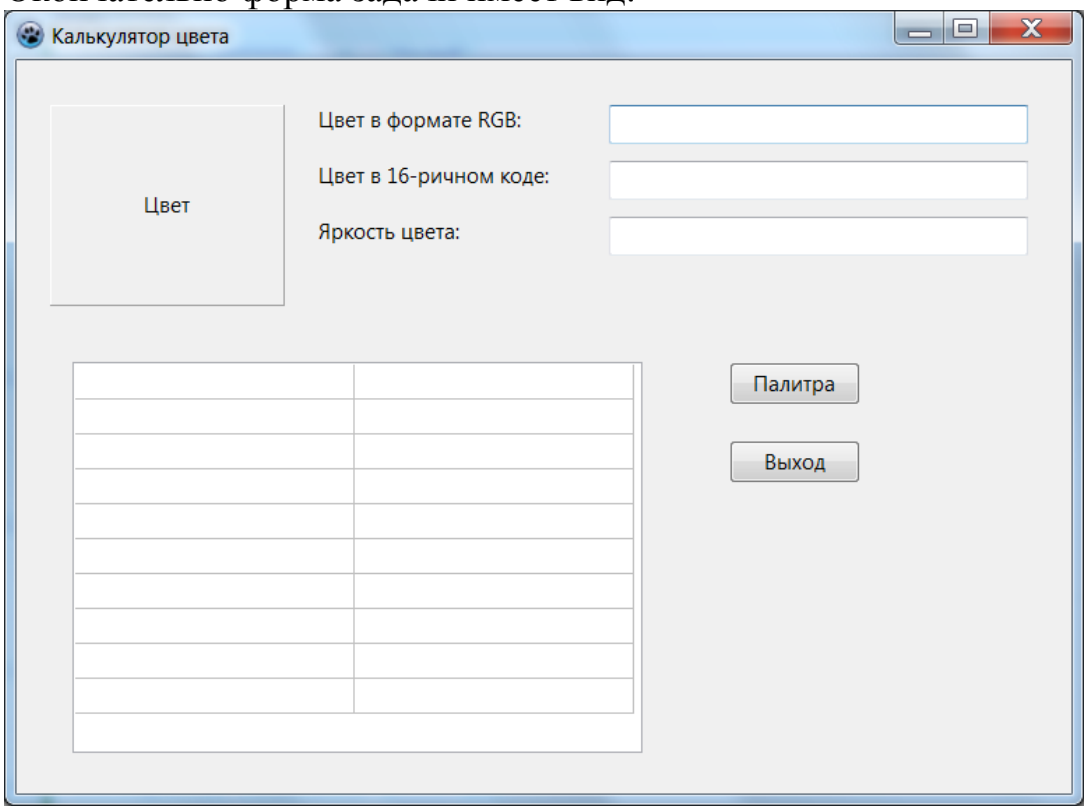
```
i := 0;
color1 := TColor($FFFFFF);
StringGrid1.ColWidths[0] := 200;
StringGrid1.ColWidths[1] := 200;
StringGrid1.Refresh;
```

Достоинства приложения: простой интерфейс, получение кода и названия цвета из палитры.

Недостатки приложения: ограниченное количество цветов в таблице и названий цветов, нет обратного преобразования.

Разработка интерфейса. В приложении использовались следующие компоненты: кнопки для работы с приложением (Button), стандартное диалоговое окно (ColorDialog), поясняющий текст (Label), текстовые поля вывода результата (Edit), таблица вывода результата (StringGrid), панель вывода цвета (Panel).

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.3 а

Постановка задачи. Калькулятор цвета – определение цвета по коду.
Внешнее описание: при выборе ползунка с переключателем основного цвета (красный, зеленый, синий) формируется новый цвет и указывается его шестнадцатеричный код.
Функциональная спецификация: для формирования цвета использовался Scale (шкала) – это виджет, который позволяет пользователю графически выбирать значение, перемещая определенный ползунок по ограниченной линии. Каждая шкала раскрашена в один из трех цветов – красный, зеленый, синий, а общий фон – в получаемый на выходе цвет.
Система программирования: Python.
Особенности реализации и возникшие трудности: программа состоит из двух частей – реализации интерфейса с передвижением каждой из трех шкал и основной функции смешивания цветов и вывода кода нового цвета.
1) Интерфейс виджета scale:

```
scales = {}
```

```
kwargs = {
    'master': root, 'orient': tkinter.HORIZONTAL, 'length': 350, 'from_': 0,
    'to_': 255, 'tickinterval': 50, 'resolution': 1,
    'command': lambda event: change_color(scales, display, out_hex)
}
for color in ('red', 'green', 'blue'):
    scales[color] = tkinter.Scale(bg=color, **kwargs)
    scales[color].pack()
```

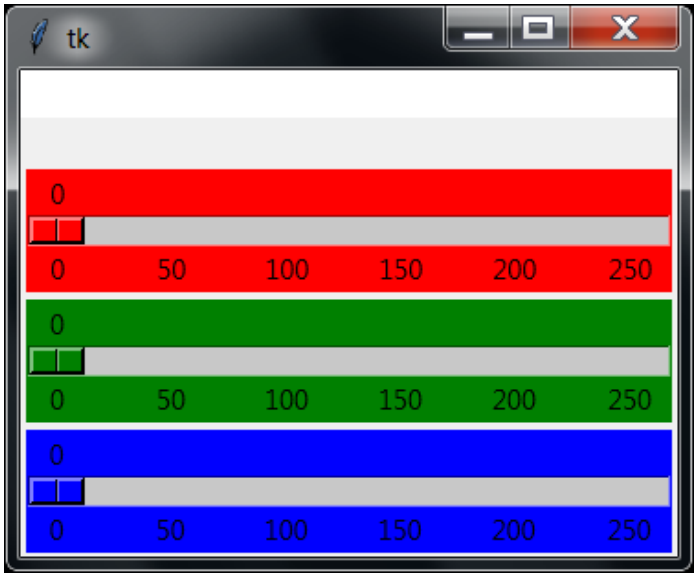
В этом фрагменте: ориентация ползунка – горизонтальная, указана длина, начальная позиция, конечная позиция, tickinterval – градация интервала, resolution – шаг; command - обработчик изменения значения виджета, т.е. привязка функции обработки.
В цикле for каждому из виджетов scale сопоставляется один из трех цветов в качестве фона.

2) Основная функция:

```
def change_color(scales, out_color, out_num):
    red = hex(scales['red'].get())[2:4] #перевод числа из шкалы в 16-ричный код
    green = hex(scales['green'].get())[2:4]
    blue = hex(scales['blue'].get())[2:4]
    if len(red) == 1:
        red = '0' + red
    if len(green) == 1:
        green = '0' + green
    if len(blue) == 1:
        blue = '0' + blue
    color = '#' + red + green + blue #формирование цвета из RGB
    out_color.configure(bg=color) #установка полученного цвета в качестве фона
    out_num['text'] = color
```

Достоинства приложения: простой интерфейс, получение кода и цвета с помощью передвижения ползунка.
Недостатки приложения: нет обратного преобразования, нет описания цвета.
Разработка интерфейса. В приложении использовались следующие виджеты: текст вывода результата со сменой фона (Label), ползунки-шкалы (Scale).

Окончательно форма задачи имеет вид:



Полный код программы представлен в электронном приложении

Задача 4.4

Постановка задачи. Изображение графических фигур.

Внешнее описание: по кнопке выбирается вид фигуры, для некоторых фигур можно настроить параметры.

Функциональная спецификация: составление изображений из графических примитивов (линия, овал, прямоугольник, многоугольник, кривая).

Система программирования: Microsoft Visual Basic.

Особенности реализации и возникшие трудности:

Прежде чем использовать графические методы класса System.Drawing.Graphics для рисования примитивных фигур, надо создать объекты Graphics (Область рисования), Pen (Перо) и Brush (Кисть). Объект Graphics(Область рисования) позволяет выбрать в качестве области рисования определенный элемент управления и обладает методами рисования графических фигур.

Объект Graphics можно создать тремя различными способами.

Первый способ состоит в использовании метода CreateGraphics() формы или элемента управления, на котором надо отобразить графику. Например, создадим объект g типа Graphics, а затем укажем определенный элемент управления в качестве области рисования. Обычно в качестве области рисования выбирается размещенное на форме графическое поле (например, PictureBox1):

```
Dim g As Graphics
g = PictureBox1.CreateGraphics
```

Объект Pen (Перо) определяет цвет и ширину линии рисования. В разделе объявления переменных необходимо определить имя объекта (например, Pn1), установить цвет (например, красный Color.Red) и ширину линии в пикселях (например, 3):

```
Dim Pn1 As New Pen(Color.Red, 3)
```

Объект Brush (Кисть) определяет цвет и стиль закрашивания прямоугольников, окружностей и других замкнутых фигур.

Сначала необходимо в разделе объявления переменных определить имя объекта (например, Brush1) и установить тип закраски и цвет (например, сплошная закраска синего цвета SolidBrush(Color.Blue)):

```
Dim Brush1 As New SolidBrush(Color.Blue)
```

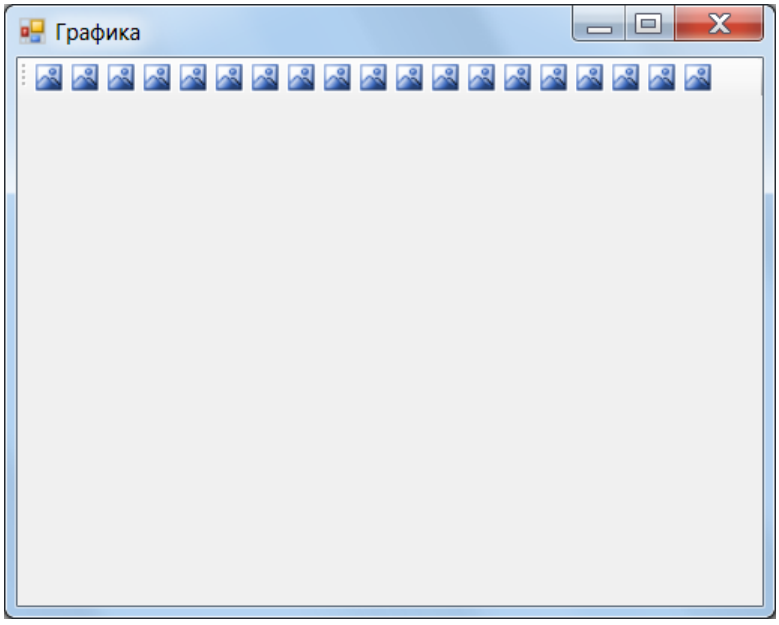
Для установки цвета в 24-битовой палитре цветов RGB используется метод Color.FromArgb(Red, Green, Blue), аргументами которого являются три числа в диапазонах от 0 до 255 (интенсивность красного, зеленого и синего цветов). Например, так можно установить пурпурный цвет для кисти Brush1:

```
Brush1.Color = Color.FromArgb(255, 0, 255)
```

Методом DrawLine рисуется отрезок прямой линии. Параметрами этого метода, кроме пера, являются координаты двух точек – начала и конца линии.
Методом DrawRectangle создается прямоугольник. Параметрами этого метода являются перо, координаты верхнего левого угла, ширина и высота.
Метод DrawEllipse используется для создания эллипса. Эллипс создается как геометрическая фигура, вписанная в прямоугольник.

Достоинства приложения: простой интерфейс, изображение большого количества фигур.
Недостатки приложения: ограниченное количество цветов для фигур, не для всех фигур есть дополнительные настройки.
Разработка интерфейса. В приложении использовались следующие компоненты: панель кнопок для функций изображений (toolStrip), компонент для вывода изображений (pictureBox).

Окончательно форма задачи имеет вид:



Рассмотрим некоторые функции программы, реализованные в процедурах – обработчиках кнопок:
1) Ряд прямоугольников заданной длины и ширины, сужающихся вверху, имитируя кирпичную кладку:

```
PictureBox1.Image = Nothing
        Refresh()' обновить графическую область
        g = PictureBox1.CreateGraphics
        Dim pn As New Pen(Color.Black, 2) ' установить цвет контура
        ' Задаем число рядов
        Dim rows As Integer
        rows = InputBox("Число рядов", "Число рядов", 5)
        ' Задаем ширину и высоту кирпича
        Dim width As Integer, height As Integer
        width = InputBox("Ширина кирпича", "Ширина кирпича", 40)
```

```
        height = InputBox("Высота кирпича", "Высота кирпича", 20)
        ' Определяем количество кирпичей
        Dim count As Integer
        For n As Integer = 1 To rows : count += n : Next n
        Dim rect(count - 1) As Rectangle
        Dim i, j As Integer, k As Integer = 0
        For i = 0 To rows - 1
            For j = 0 To i
        rect(k) = New Rectangle(width * (rows / 2 + j - i / 2), height * (i + 1), width,
height) ' формирование прямоугольной области вывода с новыми размерами
                k += 1
            Next j
        Next i
        Dim R As Integer, G1 As Integer, B As Integer
        Randomize()' датчик случайных чисел для формирования цвета
        R = CInt(Rnd() * 255)
        G1 = CInt(Rnd() * 255)
        B = CInt(Rnd() * 255)
        Dim pn1 As New SolidBrush(Color.FromArgb(R, G1, B)) ' формирование цвета заливки
        g.FillRectangles(pn1, rect) ' заливка фигуры
        g.DrawRectangles(pn, rect) ' рисование фигуры
```

2) Фрагмент процедуры для рисования циклических дуг случайным цветом (основные переменные заданы, как в предыдущей процедуре):

```
Dim rect As New Rectangle(350, 240, 50, 50)
        Dim angle As Single
        angle = InputBox("Введите угол", "Угол", 30.0F)
        Dim dlina As Integer
        dlina = InputBox("Введите длину", "Длина", 15)
        For i As Integer = 0 To dlina
            rect.Inflate(5, 5)
            For j As Integer = 0 To 5
                g.DrawArc(pn, rect, angle * 2 * j - 15.0F, angle)
            Next
        Next
```

3) Фрагмент процедуры для изображения трех кривых, заданных 4 точками с разным коэффициентом:

```
Dim bluePen As New Pen(Color.Blue, 3)
        Dim points As Point() = {New Point(0, 30), New Point(0, 105),
        New Point(75, 105), New Point(75, 30)} ' задание точек координатами
        Dim tension As Single() = {-3.0F, 6.0F, 9.0F} ' установка коэффициента для кривой
        Dim x As Integer() = {40, 200, 260} ' установка расстояния между фигурами
        For i As Integer = 0 To 2
            For j As Integer = 0 To 3
                points(j).Offset(x(i), 0) ' преобразование объекта
            Next

            g.DrawCurve(bluePen, points, tension(i)) ' рисование трех кривых
        Next
```

4) Изображение цветка с помощью кривых Безье:

```
Dim pn As New Pen(Color.Red, 3)
        Dim graphPath As New GraphicsPath
        ' Добавляем бутон
        Dim rect As New Rectangle(50, 20, 150, 150)
        Dim startAngle As Single = 15.0F
        Dim sweepAngle As Single = 30.0F
        For i As Integer = 0 To 8
            graphPath.AddPie(rect, startAngle * (3 * i + 1), sweepAngle)
        Next
        ' Добавляем стебель
```

```
        Dim pointsOfBeziers As Point() = {New Point(125, 105),
        New Point(110, 200), New Point(110, 230), New Point(130, 280)}
        graphPath.AddBeziers(pointsOfBeziers)
        ' Добавляем листик
        Dim pointsOfCurve As Point() = {New Point(115, 230),
        New Point(70, 170), New Point(30, 150), New Point(60, 200)}
        Dim tension As Single = 0.5F
        graphPath.AddClosedCurve(pointsOfCurve, tension)
        ' Рисуем весь путь
        g.DrawPath(pn, graphPath)
```

5) Изображение снеговика с заливкой из отдельных точек:

```
Dim brushBody As New HatchBrush(HatchStyle.Percent10, Color.Cyan,
 Color.Snow) ' формирование заливки из 10%-й плотности точек
        Dim rectBody As Rectangle() = {New Rectangle(50, 20, 80, 60),
        New Rectangle(40, 80, 100, 70), New Rectangle(30, 150, 120, 80)}
        Dim rectEyesAndNose As Rectangle() = {
        New Rectangle(70, 40, 10, 10), New Rectangle(100, 40, 10, 10),
        New Rectangle(86, 55, 8, 4)}
        Dim rectMouth As New Rectangle(75, 65, 30, 5)
        For i As Integer = 0 To 2
            g.FillEllipse(brushBody, rectBody(i)) ' Рисуем каждый круг
            g.FillEllipse(Brushes.Black, rectEyesAndNose(i)) ' Рисуем глаза и нос
        Next
        g.FillEllipse(Brushes.Red, rectMouth) ' Рисуем рот
```

6) Изображение треугольника с градиентной заливкой:

```
        Dim points As Point() = {New Point(200, 150), New Point(280, 30),
        New Point(360, 150)} ' определение точек треугольника
        Dim pGBrush As New PathGradientBrush(points) ' Градиентная заливка по контуру
фигуры
        pGBrush.CenterPoint() = New PointF(280, 100)
        pGBrush.CenterColor() = Color.LightPink ' внутренний цвет
        Dim colors As Color() = {Color.Blue} ' внешний цвет
        pGBrush.SurroundColors() = colors
        pGBrush.SetBlendTriangularShape(0.3F, 1.0F) ' параметры перехода градиента
        g.FillPolygon(pGBrush, points) ' рисование треугольника с заливкой
```

Полный код программы представлен в электронном приложении

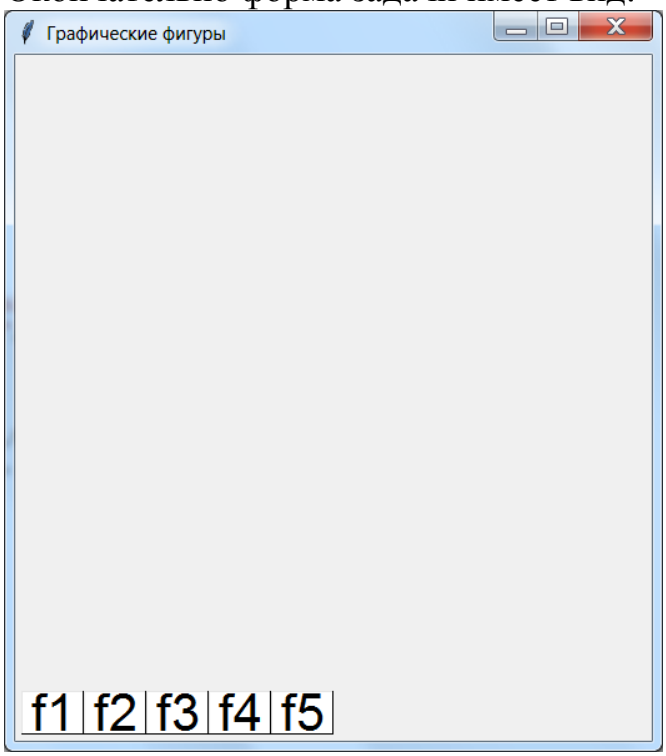
Задача 4.4 а

Постановка задачи. Изображение графических фигур.
Внешнее описание: по кнопке выбирается вид фигуры.
Функциональная спецификация: составление изображений из графических примитивов (линия, овал, прямоугольник, многоугольник, кривая).
Система программирования: Python.
Особенности реализации и возникшие трудности: каждый элемент canvas - это объект, который отслеживает Tkinter. Например, чтобы очистить холст, нужно использовать метод delete. Параметр "all" применяется для удаления всех элементов на холсте: canvas.delete("all").
Каждая фигура рисуется в отдельной процедуре, а изображение выводится на холст по нажатию кнопки.

Достоинства приложения: простой интерфейс, изображение различных фигур.

Недостатки приложения: ограниченное количество цветов для фигур, нет дополнительных настроек.
Разработка интерфейса. В приложении использовались следующие виджеты: кнопки для функций изображений (Button), компонент для вывода изображений (Canvas).

Окончательно форма задачи имеет вид:



Рассмотрим функции программы, реализованные в процедурах – обработчиках кнопок:
1) Домик, состоящий из линий, прямоугольников и овала:

```
    x=180
    y=100
    canvas.delete('all')
    canvas.create_rectangle(x-135,y+130,x+135,y+400, outline="black", fill="yellow",
width=2)
    canvas.create_rectangle(x-130,y+200,x,y+300,outline="black", fill="brown", width=2)
    canvas.create_line(x,y,x-140,y+130,fill="black",width=4)
    canvas.create_line(x,y,x+140,y+130,fill="black",width=4)
    canvas.create_line(x+140,y+130,x,y,fill="black",width=4)
    canvas.create_oval(x-30,y+65,x-30+50,y+100,outline="black",fill="blue",width=4)
```

2) Спираль из ромбов:

```
    i=1
    x=180
    y=180
    a=5
    canvas.delete('all')
    while i<40:
        xx=x
        yy=y
        x += int(a*i /sqrt(2.0))
        y -= int(a*i /sqrt(2.0))
```

```
        canvas.create_line(xx,yy,x,y,fill="red",width=2)
        i+=1
        xx=x
        yy=y
        x += int(a*i /sqrt(2.0))
        y += int(a*i /sqrt(2.0))
        canvas.create_line(xx,yy,x,y,fill="green",width=2)
        i+=1
        xx=x
        yy=y
        x -= int(a*i /sqrt(2.0))
        y += int(a*i /sqrt(2.0))
        canvas.create_line(xx,yy,x,y,fill="red",width=2)
        i+=1
        xx=x
        yy=y
        x -= int(a*i /sqrt(2.0))
        y -= int(a*i /sqrt(2.0))
        canvas.create_line(xx,yy,x,y,fill="green",width=2)
        i+=1
```

3) Фрагмент рисования цветка с лепестками из овалов:

```
    canvas.create_line(300,200,300,400,fill="green", width=5)
    canvas.create_oval(250,150,350,250,outline="", fill="yellow")
    canvas.create_oval(350,150,250,50,outline="", fill="lightpink")
```

…

4) Пирамида из сужающихся овалов:

```
    x=500/2
    y=500/2
    ellips_width   = 240
    ellips_height = 120
    f=0
    for i in range(0,10):
        canvas.create_oval(x-119+f,y-50,x+120-f,y+80-f,fill="blue")
        f+=10
    z=0
    for i in range(0,10):
        canvas.create_oval( x- ellips_width / 2+z,  y- ellips_height / 2,
                            x+ ellips_width / 2-z,  y + ellips_height / 2-z,
                            fill = "red",  width = 3  )
        z+=10
```

Полный код программы представлен в электронном приложении

Задача 4.5

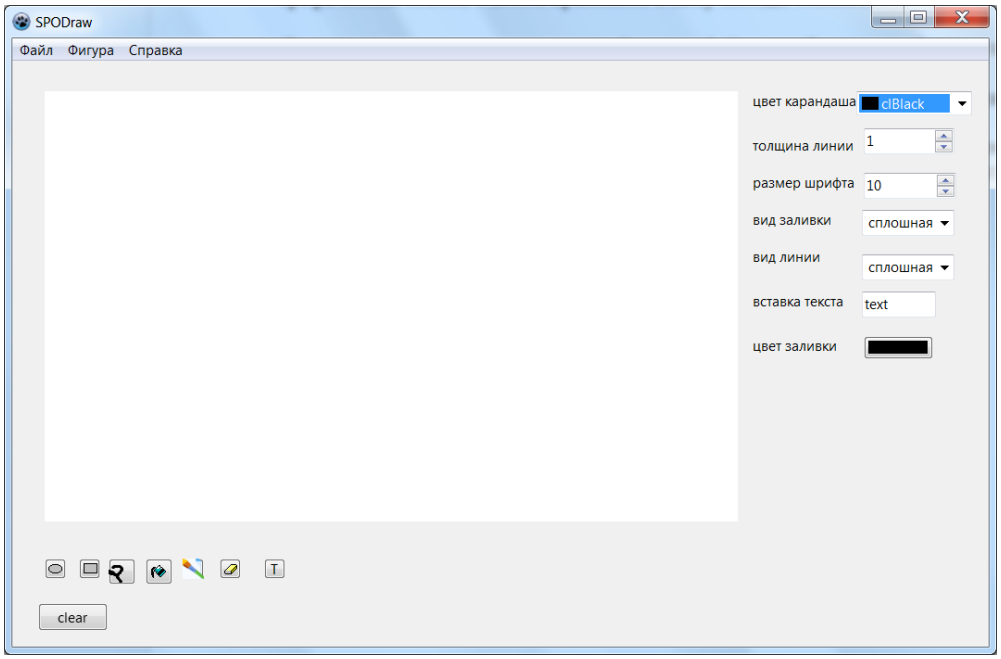
Постановка задачи. Графический редактор с изображением основных графических примитивов.

Внешнее описание: в программе используются все стандартные функции для графического редактора (открытие, сохранение, печать файла), изображение основных графических примитивов.

Функциональная спецификация: изображение основных графических примитивов: карандаш, заливка, ластик, вставка текста, овал, прямоугольник, треугольник, звезда, ромб, кривая, распылители. В отличии от изображения фигур предыдущего задания здесь используются параметры (координаты и радиус) положения курсора мыши.

Система программирования: Lazarus.
Особенности реализации и возникшие трудности: по нажатию кнопки или пункта меню происходит присваивание числовой переменной целого значения. А сами процедуры в зависимости значения – это обработчики мыши: MouseMove() – движение мыши, MouseUp() – кнопка мыши поднята, MouseDown() – кнопка мыши опущена. В этих процедурах запоминается позиция курсора мыши, т.е. текущие координаты (x,y).
При сохранении и печати файла необходимо записать рисунок как графический объект, а потом применять к нему методы объекта.
Достоинства приложения: простой интерфейс, изображение большого количества фигур.
Недостатки приложения: при изображении фигур на экране появляется только окончательный вид без анимации расширения или сужения объекта, как в стандартных графических редакторах.
Разработка интерфейса. В приложении использовались следующие компоненты: кнопки и меню для функций изображений (BitBtn, SpeedButton, ColorButton, MainMenu), компонент для вывода изображений (PaintBox), выпадающие списки для настроек типов линий и заливки (ComboBox), вспомогательный текст (Label), текст для ввода (Edit), поле со стрелками для изменения шрифта и ширины линии (SpinEdit), стандартные диалоговые окна (OpenDialog, SaveDialog, PrintDialog).

Окончательно форма задачи имеет вид:



Рассмотрим некоторые функции программы, реализованные в процедурах – обработчиках мыши:
1) Процедура движения мыши:
procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X,

```
  Y: Integer);
var i:integer;
begin
  // движение мыши
  randomize;
 if (figura=1) and (ssleft in shift) Then //если нажата кнопка с выбором фигуры №1, и мышь
движется без опускания кнопки
 begin
  PaintBox1.Canvas.Pen.Mode:=pmCopy;
 PaintBox1.Canvas.LineTo(X,Y);  // рисование линии
  end;
 if (figura=3) and (ssleft in shift) Then
 begin
end;
 if (figura=4) and (ssleft in shift) Then
 begin
 PaintBox1.Canvas.Pen.Mode:=pmNotXOR;
  end;
 if (figura=5) and (ssleft in shift) Then
 begin
 if combobox2.Itemindex=0 then paintbox1.Canvas.pen.Style:=psSolid;
 …
 if combobox2.Itemindex=4 then paintbox1.Canvas.pen.Style:=psdashdotdot;
 PaintBox1.Canvas.LineTo(X,Y);    // выбор вида линии для рисования кистью
  end;
  if (figura=8) and (ssleft in shift) then
    begin
    x1:=x;
    y1:=y;  // распылитель
 for i:=1 to 50 do
    begin
    paintbox1.Canvas.Pixels[x1+random(20), y1+random(20)]:=ColorBox1.Selected;
    end;
 end;
  if (figura=9) and (ssleft in shift) Then
 begin
  PaintBox1.Canvas.pen.Color:=clwhite;    // белый цвет прямоугольника для ластика
  PaintBox1.Canvas.brush.Color:=clwhite;
  x1:=x;
y1:=y;
PaintBox1.Canvas.Rectangle(x1,y1, x1+30,y1+30);
 end;
  if (figura=10) and (ssleft in shift) then
    begin
    x1:=x;
    y1:=y; // цветной спрей
 for i:=1 to 20 do
    begin
       paintbox1.Canvas.brush.Color:=Random(65535);
    paintbox1.Canvas.ellipse(x1,y1,x1+random(15),y1+random(15));
    end;
 paintbox1.Canvas.brush.Color:=clwhite;
```

```
 end;
if (figura=15) and (ssleft in shift) then
   begin
   x1:=x;
   y1:=y;  // краскопульт
for i:=1 to 7 do
   begin
      paintbox1.Canvas.brush.Color:=Random(65535);
   paintbox1.Canvas.RadialPie(x1,y1,x1+random(50),y1+random(50),45,120);
   end;
paintbox1.Canvas.brush.Color:=clwhite;
end;
 end;
```

2) Процедура обработки поднятия кнопки мыши:

```
procedure TForm1.PaintBox1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 var
// mb:hbrush;
Bitmap: TBitmap;
begin
Paintbox1.Canvas.Pen.Mode:= pmcopy;
// кнопка мыши поднята (вверх)
Paintbox1.Canvas.Brush.Handle:=CreatePatternBrush(bitmap.Picture.Bitmap.Handle);
if (figura=1) Then
begin
PaintBox1.Canvas.LineTo(X,Y);  // остановка рисования линии
end;
if (figura=2) then
  begin
      // выбор вида и цвета заливки
     paintbox1.Canvas.Brush.Color:=colorbutton1.ButtonColor;
    if combobox1.Itemindex=0 then paintbox1.Canvas.Brush.Style:=bsSolid;
…
paintbox1.Canvas.FloodFill(x,y,paintbox1.canvas.Pixels[x,y], fssurface); // заливка
end;
 if (figura=5) Then
begin
PaintBox1.Canvas.LineTo(X,Y);
end;
 end;
```

3) Процедура обработки опускания кнопки мыши:

```
procedure TForm1.PaintBox1MouseDown (Sender:TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
 var r, i:integer;
begin
// кнопка мыши опущена (вниз)
randomize;
Paintbox1.Canvas.Pen.Width:=spinedit1.Value;
 PaintBox1.Canvas.MoveTo(X,Y);
```

```
    if (figura=1) and (ssleft in shift) Then
     begin
 // изменение координат начала и конца движения
x1:=x;
y1:=y;
stx:=x;
sty:=y;
end;
if (figura=4) and (ssleft in shift) Then
 begin
  x1:=x;
y1:=y;
PaintBox1.Canvas.Rectangle(x1,y1, stx,sty);
   PaintBox1.Canvas.Rectangle(x1,y1, x,y);  // прямоугольник
 stx:=x;
 sty:=y;
 end;
 if (figura=3) and (ssleft in shift) Then
 begin
 x1:=x;
y1:=y;
PaintBox1.Canvas.Ellipse(x1,y1, stx,sty); // овал
 PaintBox1.Canvas.Ellipse(x1,y1, x, y);
 stx:=x;
 sty:=y;
end;
 if (figura=5) and (ssleft in shift) Then
     begin
x1:=x;    // изменение координат для кисти
y1:=y;
stx:=x;
sty:=y;
end;
 if (figura=6) then
    begin
     x1:=x;
y1:=y;
paintbox1.Canvas.Polygon([Point(stx, sty), Point(x1,y1),Point(x1-90, y1+200)]);
// треугольник
    stx:=x;
 sty:=y;
    end;
 if (figura=7) then
    begin
    x1:=x;
y1:=y;
r:=round(sqrt(sqr(x1-stx)+sqr(y1-sty)));
    StarLine(x1, y1, r);  // вызов процедуры рисования звезды
    stx:=x;
 sty:=y;
    end;
 if (figura=11) and (ssleft in shift) Then
```

```
 begin
  x1:=x;
y1:=y;
PaintBox1.Canvas.RoundRect(x1,y1, stx,sty, (stx-x1) div 2,(sty-y1) div 2);
// скругленный прямоугольник
 stx:=x;
 sty:=y;
 end;
  if (figura=12) and (ssleft in shift) Then
 begin
  stx:=x;
sty:=y;
PaintBox1.Canvas.Polygon([Point(stx,sty-y1),Point(stx-x1,sty), POint(stx,sty+y1),
Point(x1+stx,sty)]);
// ромб
 x1:=x;
 y1:=y;
 end;
  if (figura=13) and (ssleft in shift) Then
 begin
  stx:=x;
sty:=y;   // кривая Безье
PaintBox1.Canvas.PolyBezier([Point(stx, sty), Point(stx+x1,sty+y1), Point(x1+50, y1+50),
Point(x1+100,y1+100)]);
 x1:=x;
 y1:=y;
 end;
  if figura=14 then
      begin
           PaintBox1.Canvas.Font.Color:=colorbox1.Selected;
           PaintBox1.Canvas.Font.Size:=edit1.Font.Size+spinedit2.Value;
          // размер шрифта текста
      stx:=x;
sty:=y;
      PaintBox1.Canvas.brush.Color:=paintbox1.canvas.Pixels[stx,sty];
      PaintBox1.Canvas.TextOut(stx,sty,edit1.text); // вывод текста
      end;
 end;
```

4) Процедура выбора цвета и толщины для линий:

```
procedure TForm1.ColorBox1Change(Sender: TObject);
begin
  PaintBox1.Canvas.Pen.Color:=ColorBox1.Selected; // выбор цвета из списка
  Paintbox1.Canvas.Pen.Width:=spinedit1.Value;   // толщина линии
end;
```

6) Процедура выбора цвета для заливки:

```
colorbutton1.Color:=colordialog1.color; // открытие окна выбора цвета
```

7) Процедура печати изображения:

```
if not PrintDialog1.Execute then exit;  // печать файла
```

```
 with printer do
 begin
 BeginDoc;
r.Left:=paintbox1.Left;r.Top:=paintbox1.Top;r.Right:=paintbox1.left+paintbox1.Width;
 r.Bottom:=paintbox1.top+paintbox1.Height;
  EndDoc;
 end;
```

8) Процедура открытия файла:

```
if opendialog1.Execute then
     begin
     bmp:=TBitmap.create;
          bmp.loadfromfile(opendialog1.FileName);  // открыть файл
          paintbox1.width:=bmp.width;
          paintbox1.height:=bmp.height;
end;
```

9) Процедура сохранения файла:

```
var
 bitmap:TBitmap;
 src, dst:TRect;
begin
 bitmap:=TBitmap.Create;  // создание графического объекта из рисунка для сохранения в
файле
 try
 with bitmap do
 begin
 Width:= PaintBox1.Width;
 Height:=PaintBox1.Height;
 end;
 dst:=Rect(0, 0, PaintBox1.Width, PaintBox1.Height);
 src:=Rect(0, 0, PaintBox1.Width, PaintBox1.Height);
 bitmap.Canvas.CopyRect(dst, PaintBox1.Canvas, src);
 if SaveDialog1.Execute
 then bitmap.SaveToFile(SaveDialog1.FileName); // сохранить как
 finally
 bitmap.Free;
 end;
end;
```

Полный код программы представлен в электронном приложении

Задача 4.5 а

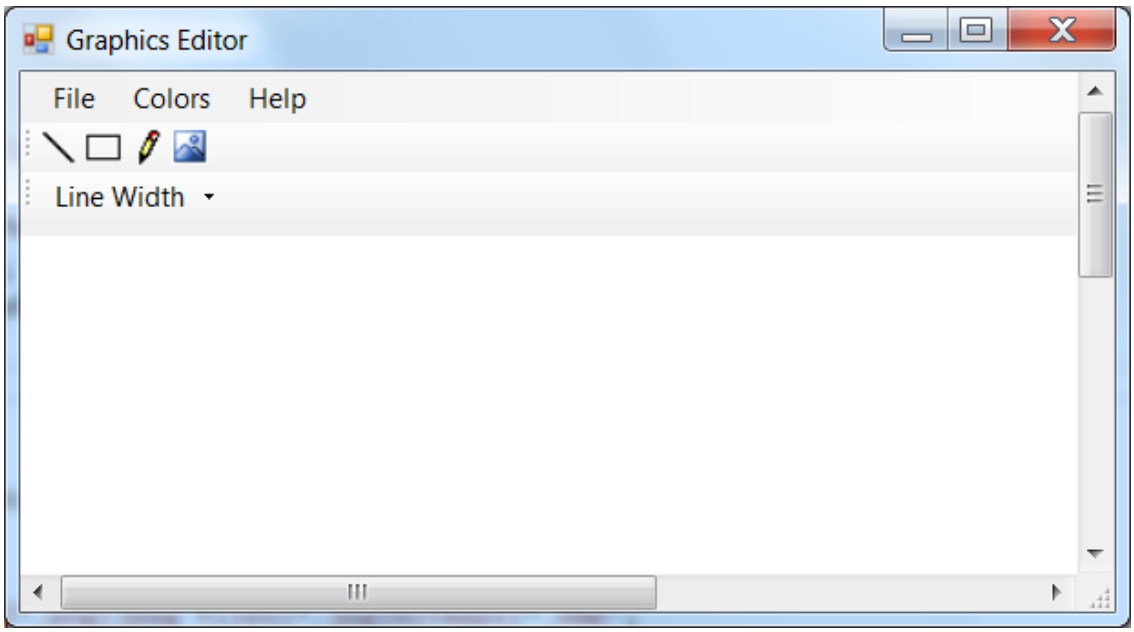
Постановка задачи. Графический редактор с изображением основных графических примитивов.

Внешнее описание: в программе используются некоторые стандартные функции для графического редактора (открытие, сохранение файла), изображение основных графических примитивов.

Функциональная спецификация: изображение основных графических примитивов: карандаш, овал, прямоугольник. В отличии от предыдущего задания здесь при рисовании фигур используются методы анимации промежуточных значений.
Система программирования: Microsoft Visual C#.
Особенности реализации и возникшие трудности. Приложение использует простой пользовательский элемент управления Panel, который был создан для двойной буферизации. Двойная буферизация уменьшает эффект, который иногда возникает при перерисовке сложного изображения.
Растровое изображение под названием "snapshot" создается для сохранения нарисованного изображения. В событии MouseDown панели начальные координаты сохраняются, а временное растровое изображение под названием "tempDraw" клонируется из изображения "snapshot".
Когда курсор мыши перемещается, то захватываются его координаты для прорисовки изображения. В процедуре рисования графический объект "g" создается из изображения tempDraw и на нем рисуется линия. Фактически, строка добавляется к временному битовому массиву. Затем новое изображение tempDraw переносится на графический объект "e" панели.
Когда происходит событие MouseUp, то рисунок определяется как законченный, и изображение tempDraw передается в наше растровое изображение snapshot, т.е. готово к следующей фигуре для рисования.

Достоинства приложения: простой интерфейс, прорисовка фигур как в редакторах.
Недостатки приложения: малое количество функций.
Разработка интерфейса. В приложении использовались следующие компоненты: кнопки и меню для функций изображений (menuStrip, toolStrip), компонент для вывода изображений (Panel), стандартные диалоговые окна (OpenFileDialog, SaveFileDialog, ColorDialog).

Окончательно форма задачи имеет вид:



Рассмотрим основные функции программы, реализованные в функциях – обработчиках мыши:

1) Функция движения мыши:

```
if (mouseDown )
            {
                x2 = e.X;
                y2 = e.Y;
                panel1.Invalidate();
                panel1.Update();
            }
```

2) Функция обработки поднятия кнопки мыши:

```
private void panel1_MouseUp(object sender, MouseEventArgs e)
        {
            mouseDown = false;
            snapshot = (Bitmap)tempDraw.Clone();
        }
```

3) Функция обработки опускания кнопки мыши:

```
mouseDown = true;
            x1 = e.X;
            y1 = e.Y;
            tempDraw = (Bitmap)snapshot.Clone();
```

4) Основная функция работы с изображениями:

```
private void panel1_Paint(object sender, PaintEventArgs e)
        {
            switch (selectedTool)
            {
                case "Line":
                    if (tempDraw != null)
                    {
                        tempDraw = (Bitmap)snapshot.Clone();
                        Graphics g = Graphics.FromImage(tempDraw);
                        Pen myPen = new Pen(foreColor, lineWidth);
                        g.DrawLine(myPen, x1, y1, x2, y2);
                        myPen.Dispose();
                        e.Graphics.DrawImageUnscaled(tempDraw, 0, 0);
                        g.Dispose();
```

```
                }
                break;
            case "Rectangle":
               if (tempDraw != null)
                {
                    tempDraw = (Bitmap)snapshot.Clone();
                    Graphics g = Graphics.FromImage(tempDraw);
                    Pen myPen = new Pen(foreColor, lineWidth);
                    g.DrawRectangle(myPen, x1, y1, x2-x1, y2-y1);
                    myPen.Dispose();
                    e.Graphics.DrawImageUnscaled(tempDraw, 0, 0);
                    g.Dispose();
                }
                break;
         case "Pencil":
                if (tempDraw != null)
                {
                    Graphics g = Graphics.FromImage(tempDraw);
                    Pen myPen = new Pen(foreColor, lineWidth);
                    g.DrawLine(myPen, x1, y1, x2, y2);
                    myPen.Dispose();
                    e.Graphics.DrawImageUnscaled(tempDraw, 0, 0);
                    g.Dispose();
                    x1 = x2;
                    y1 = y2;
                }
                break;
            case "Ellipse":
                if (tempDraw != null)
                {
                    tempDraw = (Bitmap)snapshot.Clone();
                    Graphics g = Graphics.FromImage(tempDraw);
                    Pen myPen = new Pen(foreColor, lineWidth);
                    g.DrawEllipse(myPen, x1, y1, x2-x1, y2-y1);
                    myPen.Dispose();
                    e.Graphics.DrawImageUnscaled(tempDraw, 0, 0);
                    g.Dispose();
                }
                break;
            default:
                break;
        }   }
```

5) Выбор нужной кнопки с помощью цикла:

```
private void toolStripButton1_Click(object sender, EventArgs e)
        {
            foreach (ToolStripButton  btn in toolStrip1.Items  )
            {
                btn.Checked = false;
            }
            ToolStripButton btnClicked = sender as ToolStripButton;
            btnClicked.Checked = true;
            selectedTool = btnClicked.Name;
        }
```

6) Выбор ширины линии с помощью меню:

```
private void ptToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (ToolStripMenuItem itm in toolStripSplitButton1.DropDownItems)
            {
                itm.Checked = false;
            }
            ToolStripMenuItem itmClicked = sender as ToolStripMenuItem;
```

```
            itmClicked.Checked = true;
            lineWidth = int.Parse(itmClicked.Text.Remove(1));
        }
```

7) Создание нового файла (очистка области рисования):

```
snapshot = new Bitmap(panel1.ClientRectangle.Width, this.ClientRectangle.Height);
            tempDraw = (Bitmap)snapshot.Clone();
            Graphics g = Graphics.FromImage(tempDraw);
            g.Clear(Color.White);
            g.Dispose();
```

8) Открытие графического файла:

```
openFileDialog1.Filter = "Png files|*.png|jpeg files|*.jpg|bitmaps|*.bmp";
            if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                panel1.Width = (Image.FromFile(openFileDialog1.FileName).Width);
                panel1.Height = (Image.FromFile(openFileDialog1.FileName).Height);
                snapshot = (Bitmap)Image.FromFile(openFileDialog1.FileName).Clone();
            }
```

9) Сохранение файла:

```
saveFileDialog1.Filter = "Png files|*.png|jpeg files|*.jpg|bitmaps|*.bmp";
            if (saveFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                snapshot.Save(saveFileDialog1.FileName);
            }
```

Полный код программы представлен в электронном приложении