# Министерство образования Республики Беларусь

УПРАВЛЕНИЕ ОБРАЗОВАНИЯ МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Могилевский государственный политехнический колледж»

# Домашняя контрольная работа №2

По дисциплине: «**Конструирование программ и языки программирования**»

Группа ПО-455

## Выполнил О. И. Любаль

Шифр 14

# 2021

**15 Опишите коллекции: словари**

Распространенный тип коллекции представляют словари. Словарь хранит объекты, которые представляют пару ключ-значение. Класс словаря Dictionary<K, V> типизируется двумя типами: параметр K представляет тип ключей, а параметр V предоставляет тип значений.

### Создания и инициализация словаря

Класс Dictionary предоставляет ряд конструкторов для создания словаря. Например, мы можем создать пустой словарь:

|  |  |
| --- | --- |
| 1 | Dictionary<int, string> people = new Dictionary<int, string>(); |

Здесь словарь people в качестве ключей принимает значения типа int, а в качестве значений - строки.

При определении словаря его сразу же можно инициализировать значениями:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var people = new Dictionary<int, string>()  {      { 5, "Tom"},      { 3, "Sam"},      { 11, "Bob"}  }; |

При инициализации применяется инициализитор - в фигурных скобках после вызова конструктора объекту передаются начальные данные. В случае со словаем мы можем передать в инициализаторе набор элементов, где каждый элемент заключается в фигурные скобки, например:

|  |  |
| --- | --- |
| 1 | { 5, "Tom"} |

Каждый элемент представляет два значения: первое значение представляет ключ, а второе значение - собственно значение элемента. Поскольку при объявлении словаря people для ключей указан тип int, а для значений - тип string, то в элементе словаря сначала указывается число int, а затем строка. То есть в случае выше элемент имеет ключ 5, а значение - "Tom". Затем по ключу элемента мы сможем получить его значение.

Также мы можем применять другой способ инициализации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var people = new Dictionary<int, string>()  {      [5] = "Tom",      [6] = "Sam",      [7] = "Bob"  }; |

При таком способе инициализации в квадратных скобках указывается ключ и ему присваивается значение элемента. Но в целом этот способ инициализации будет равноценен предыдущему.

### KeyValuePair

Стоит отметить, что каждый элемент в словаре представляет структуру KeyValuePair<TKey, TValue>, где параметр TKey представляет тип ключа, а параметр TValue - тип значений элементов. Эта структура предоставляет свойства Key и Value, с помощью которых можно получить соответственно ключ и значение элемента в словаре. И одна из версий конструктора Dictionary позволяет инициализировать словарь коллекцией объектов KeyValuePair:

|  |  |
| --- | --- |
| 1  2  3 | var mike = new KeyValuePair<int, string>(56, "Mike");  var employees = new List<KeyValuePair<int, string>>() { mike};  var people = new Dictionary<int, string>(employees); |

Конструктор типа KeyValuePair принимает два параметра - ключ элемента и его значения. То есть в данном случае создается один такой элемент - mike с ключом 56 и значением "Mike". И этот элемент добавляется в список employees, которым затем инициализируется словарь.

Можно совместить оба способа инициализации:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | var mike = new KeyValuePair<int, string>(56, "Mike");  var employees = new List<KeyValuePair<int, string>>() { mike };  var people = new Dictionary<int, string>(employees)  {      [5] = "Tom",      [6] = "Sam",      [7] = "Bob",  }; |

В данном случае в словаре people будет четыре элемента.

### Перебор словаря

Для перебора словаря можно применять цикл foreach:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var people = new Dictionary<int, string>()  {      [5] = "Tom",      [6] = "Sam",      [7] = "Bob"  };  foreach(var person in people)  {      Console.WriteLine($"key: {person.Key}  value: {person.Value}");  } |

При переборе каждый элемент будет помещаться в переменную, которая представляет тип KeyValuePair, соответственно с помощью свойств Key и Value мы сможем получить ключ и значение элемента. Консольный вывод программы:

key: 5 value: Tom

key: 6 value: Sam

key: 7 value: Bob

### Получение элементов

Для обращения к элементам из словаря применяется их ключ, который передается в квадратных скобках:

|  |  |
| --- | --- |
| 1 | словарь[ключ] |

Таким образом мы можем получить и изменить элементы словаря

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  1  1  1  1  1  1  1 | var people = new Dictionary<int, string>()  {      [5] = "Tom",      [6] = "Sam",      [7] = "Bob",  };  // получаем элемент по ключу 6  string sam = people[6];  // Sam  Console.WriteLine(sam);  // Sam  // переустанавливаем значение по ключу 6  people[6] = "Mike";  Console.WriteLine(people[6]);  // Mike    // добавляем новый элемент по ключу 22  people[22] = "Eugene";  Console.WriteLine(people[22]);  // Eugene |

Более того, таким образом мы можем также добавить новый элемент в словарь. При установке значения по ключу, если элемент с таким ключом уже есть в словаре, то значение переустанавливается. Если же элемента с подобным ключом нет в словаре, то элемент добавляется.:

### Методы и свойства Dictionary

Среди методов класса Dictionary можно выделить следующие:

* void Add(K key, V value): добавляет новый элемент в словарь
* void Clear(): очищает словарь
* bool ContainsKey(K key): проверяет наличие элемента с определенным ключом и возвращает true при его наличии в словаре
* bool ContainsValue(V value): проверяет наличие элемента с определенным значением и возвращает true при его наличии в словаре
* bool Remove(K key): удаляет по ключу элемент из словаря

Другая версия этого метода позволяет получить удленный элемент в выходной параметр: bool Remove(K key, out V value)

* bool TryGetValue(K key, out V value): получает из словаря элемент по ключу key. При успешном получении передает значение элемента в выходной параметр value и возвращает true
* bool TryAdd(K key, V value): добавляет в словарь элемент с ключом key и значением value. При успешном добавлении возвращает true

Из свойств следует отметить свойство Count, которое возвращает количество элементов в словаре.

**36 Опишите процесс создания и настройки форм**

Для создания графических интерфейсов с помощью платформы .NET применяются разные технологии - Window Forms, WPF, UWP. Однако наиболее простой и удобной платформой до сих пор остается Window Forms или сокращенно WinForms. Данное руководство ставит своей целью дать понимание принципов создания графических интерфейсов с помощью технологии WinForms и работы основных элементов управления.

После создания нового проекта Visual Studio откроет его с созданными по умолчанию файлами. Вид окна приложения Visual Studio предаставлен на рисунке 1.

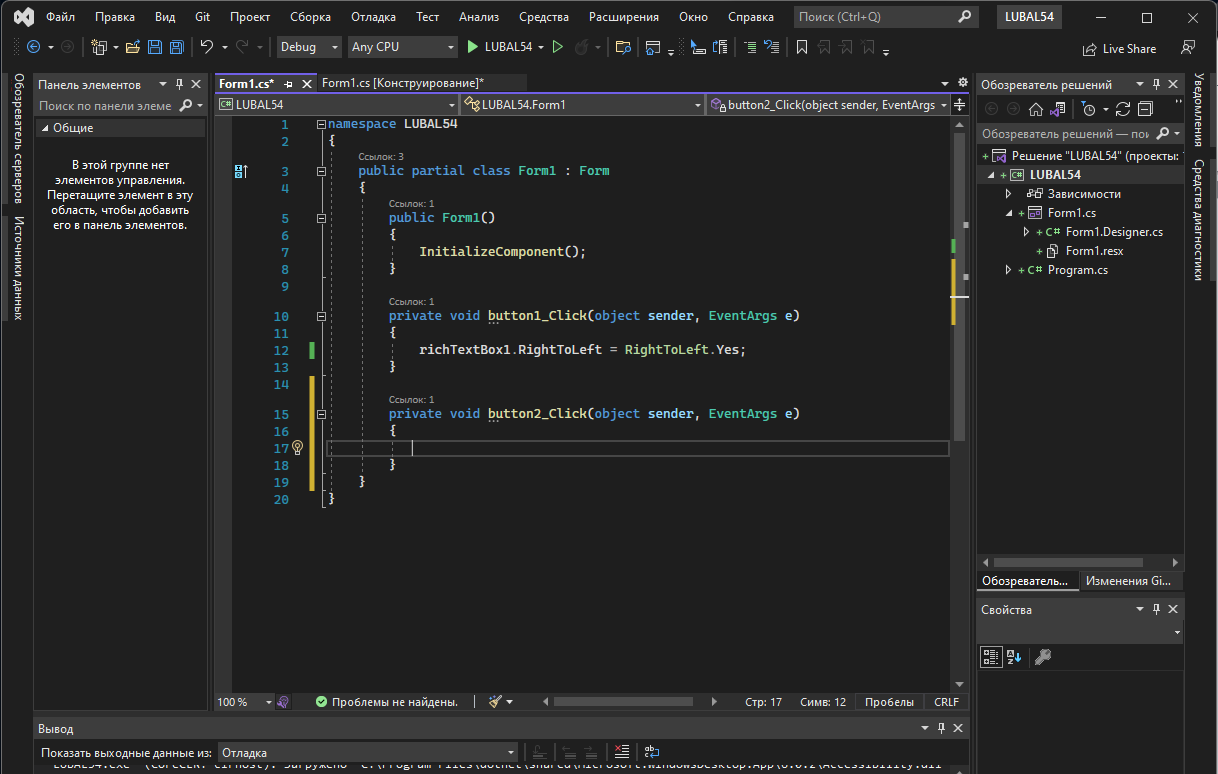


Рисунок 1. Вид окна приложения Visual Studio

Справа находится окно Обозреватель решений, в котором можно увидеть структуру нашего проекта. В данном случае у нас сгенерированная по умолчанию структура:

* Зависимости - это узел содержит сборки dll, которые добавлены в проект по умолчанию. Эти сборки как раз содержат классы библиотеки .NET, которые будет использовать C#
* Далее идет файл единственной в проекте формы - Form1.cs, который по умолчанию открыт в центральном окне:

|  |  |
| --- | --- |
|  | namespace LUBAL54  {      public partial class Form1 : Form      {          public Form1()          {              InitializeComponent();          }      }  } |

* Класс формы - Form1 представляет графическую форму - фактически то окно, которое мы увидим на экране при запуске проекта.
* Этот класс определяется как частичный (с модификатором partial) и наследуется от встроенного класса Form, который содержит базовую функциональность форм.
* В самом классе Form1 определен по умолчанию только конструктор, где вызывается метод InitializeComponent(), который выполняет инициализацию компонентов формы из файла дизайнера.
* Рядом с этим элементом можно заметить другой файл формы - Form1.Designer.cs. Это файл дизайнера - он содержит определение компонентов формы, добавленных на форму в графическом дизайнере и именно его код по сути передается выше через вызов InitializeComponent()
* Program.cs определяет точку входа в приложение:

|  |  |
| --- | --- |
|  | namespace LUBAL54  {      internal static class Program      {          [STAThread]          static void Main()          {              ApplicationConfiguration.Initialize();              Application.Run(new Form1());          }      }  } |

* Данный файл содержит класс Program. Выполнение программы на языке C# начинается с метода Main. И в классе Program как раз определен подобный метод.
* Метод Main снабжен атрибутом [STAThread]. Этот атрибут, грубого говоря, необходим для корректной работы компонентов Windows. В самом методе сначала вызывается метод

|  |  |
| --- | --- |
|  | ApplicationConfiguration.Initialize() |

* который устанавливает некоторую базовую конфигурацию приложения
* Затем вызывается метод

|  |  |
| --- | --- |
|  | Application.Run(new Form1()); |

* в который передается объект отображаемой по умолчанию на экране формы.
* То есть, когда мы запустим приложение, сработает метод Main, в котором будет вызван метод Application.Run(new Form1()), благодаря чему мы увидим форму Form1 на экране (рисунок 2).

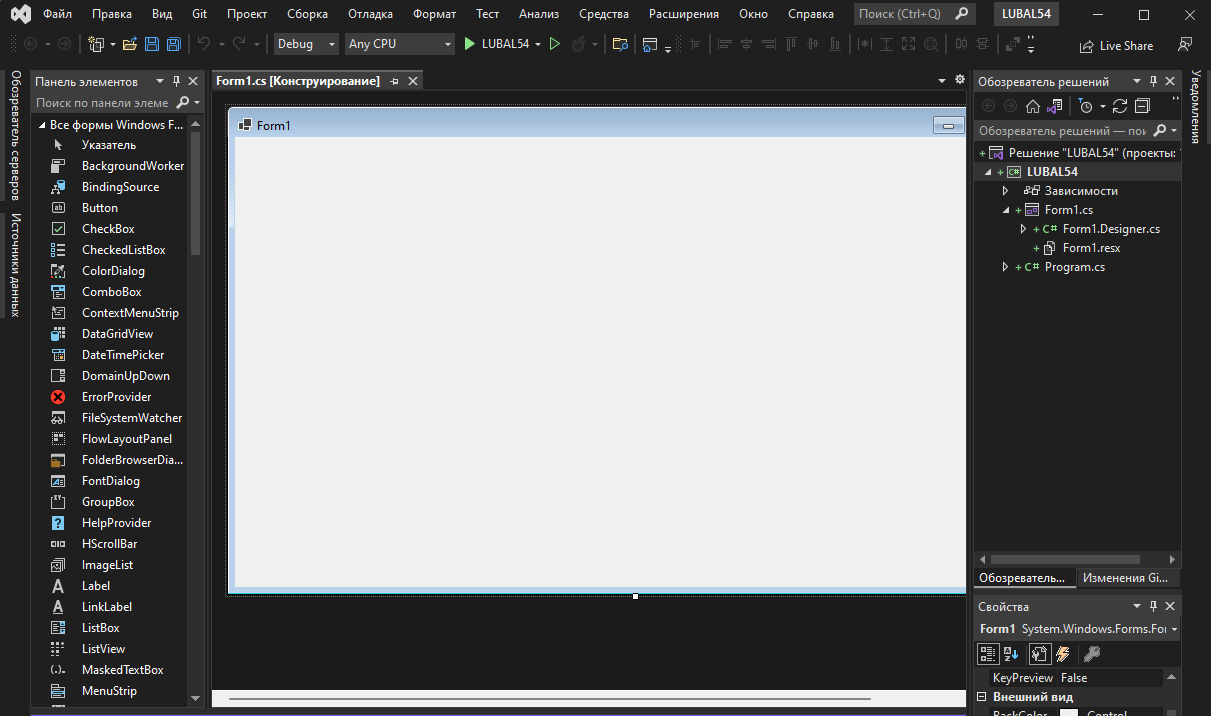


Рисунок 2 Форма Form1 на экране

Одним из преимуществ разработки в Visual Studio приложений Windows Forms является наличие графического редактора, который позволяет в графическом виде представить создаваемую форму и в принципе упрощает работу с графическими компонентами.

**54 Создайте приложение для изменения основных свойств компонента RichTextBox. Изменения должны отображаться**

Создадим приложение и разместим на форме компонент RichTextBox и кнопки для изменения его свойств. На рисунке 3 представлена форма в режиме редактирования – для размещения элементов управления.

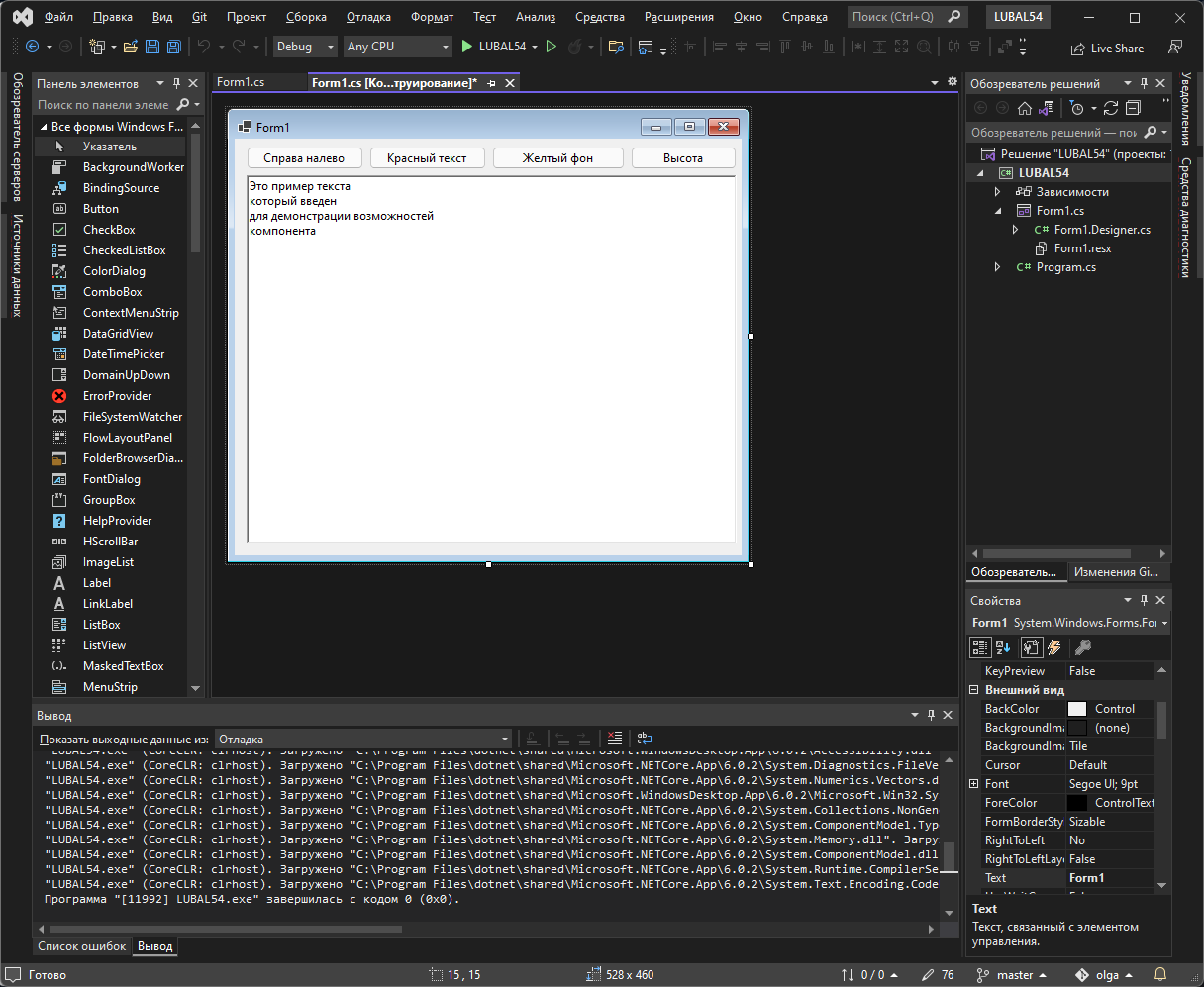


Рисунок 3 Форма приложения в режиме редактирования

Для кнопок по двойному щелчку назначим обработчики события в котором будем изменять свойства основного компонента RichTextBox.

Форма работающего приложения представлена на рисунке 4.

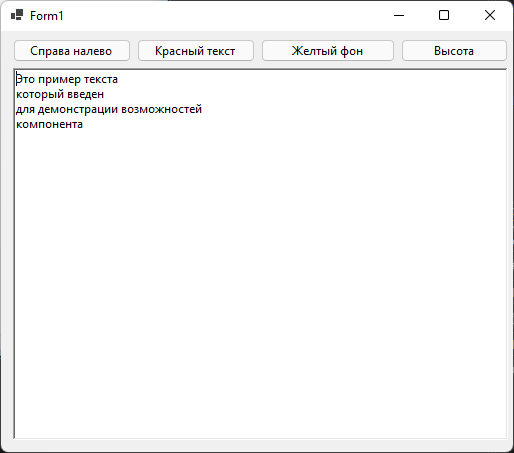


Рисунок 4. Форма работающего приложения.

Изменение свойства компонента – выравнивание текста по нажатию на кнопку представлено на рисунке 5.

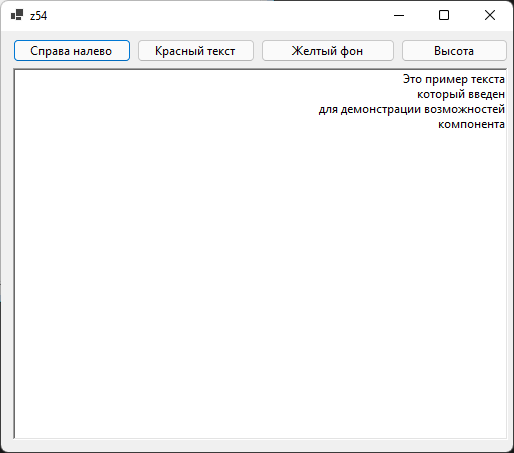


Рисунок 5. Изменение свойства компонента – выравнивание текста.

**Исходный текст программы:**

namespace LUBAL54

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

richTextBox1.RightToLeft = RightToLeft.Yes;

}

private void button2\_Click(object sender, EventArgs e)

{

richTextBox1.ForeColor = Color.Red;

}

private void button3\_Click(object sender, EventArgs e)

{

richTextBox1.BackColor = Color.Yellow;

}

private void button4\_Click(object sender, EventArgs e)

{

richTextBox1.Height = 150;

}

}

}

Блок-схема алгоритма работы программы представлена на рисунке 6.

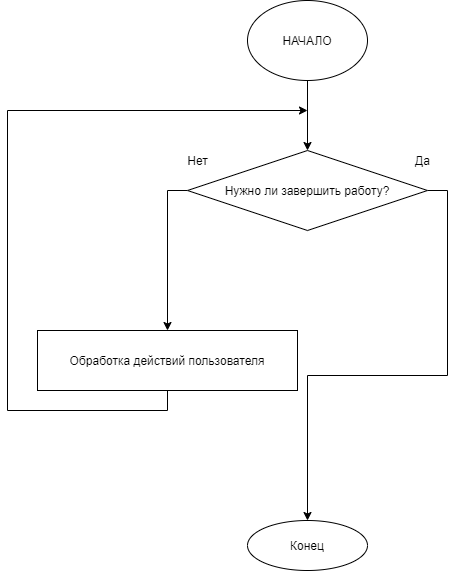


Рисунок 6. Блок-схема алгоритма работы программы.

**76 Создайте приложение для построения диаграммы отражающей число читателей в трех библиотеках. Тип диаграммы: круговая или гистограмма выбирается пользователем. Число читателей в каждой библиотеке вводите с формы**

Создадим форму и разместим на ней диаграмму, кнопки управления и элементы ввода данных. Вид формы в режиме редактирования представлен на рисунке 7.

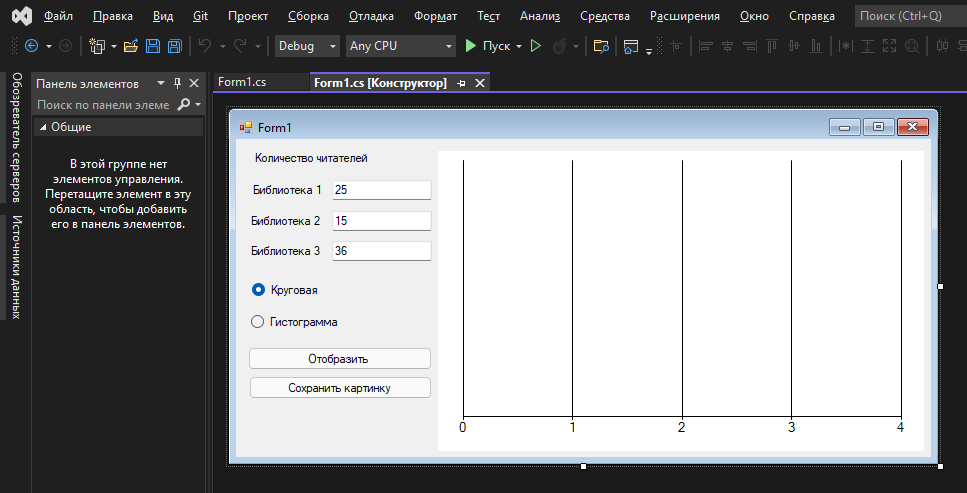


Рисунок 7. Вид формы в режиме редактирования.

Двойным нажатием на кнопки заполним их обработчики событий. Для кнопки отображения будем заполнять значения диаграммы а для кнопки сохранения организуем сохранение диаграммы на диск для дальнейшего использования.

Вид формы с круговой диаграммой в процессе работы представлен на рисунке 8.

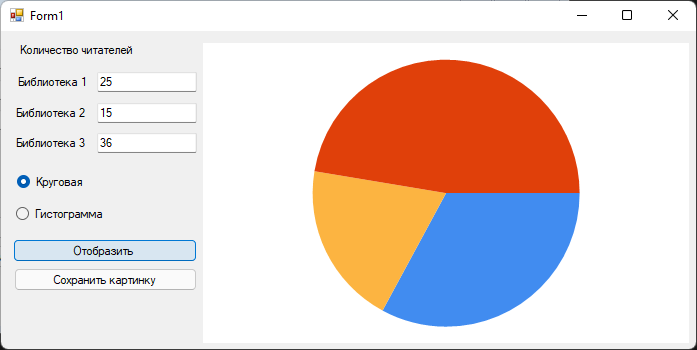


Рисунок 8. Вид формы с круговой диаграммой

Вид формы с гистограммой представлен на рисунке 9.

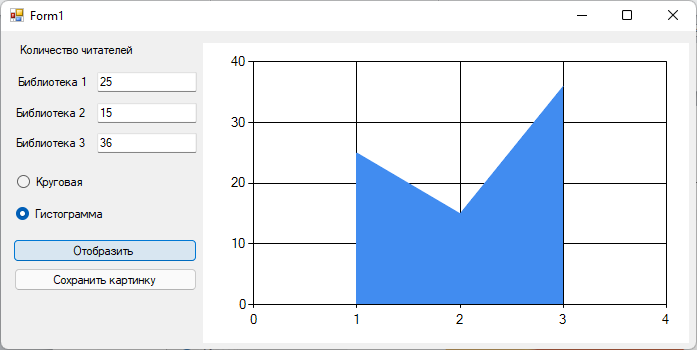


Рисунок 9. Вид формы с гистограммой.

**Исходный код программы**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace chart76

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void chart1\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

if (radioButton1.Checked)

{

chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Pie;

}

if (radioButton2.Checked)

{

chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Area;

}

chart1.Series[0].Points[0].YValues[0] = Convert.ToDouble(textBox1.Text);

chart1.Series[0].Points[1].YValues[0] = Convert.ToDouble(textBox2.Text);

chart1.Series[0].Points[2].YValues[0] = Convert.ToDouble(textBox3.Text);

}

private void button2\_Click(object sender, EventArgs e)

{

chart1.SaveImage("D:/image.bmp", ChartImageFormat.Bmp);

}

}

}

Блок-схема алгоритма работы программы представлена на рисунке 10.

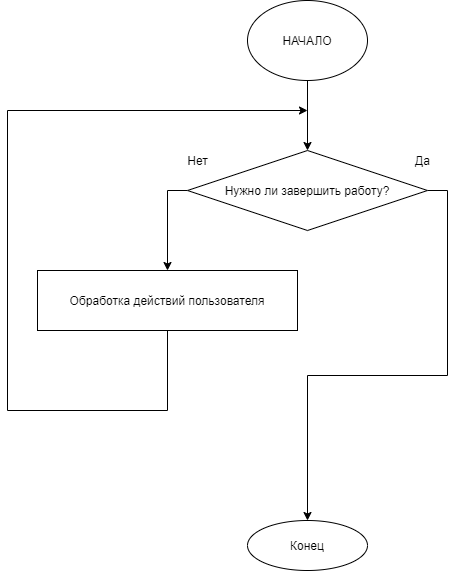


Рисунок 10. Блок-схема алгоритма работы программы.

**Список использованных источников**

1. Албахари, Д. C# 5.0. Справочник. Полное описание языка / Д.Албахари. – М.: Вильямс, 2014.
2. Лабор, В.В. Си Шарп: Создание приложений для Windows / В.В.Лабор. - Мн.: Харвест, 2003.
3. Павловская, Т.А С#. Программирование на языке высокого уровня / Т.А.Павловская. - СПб: Питер, 2014.
4. Фролов, А.В. Визуальное проектирование приложений С# / А.В.Фролов. - М: КУДИЦ - ОБРАЗ, 2003.
5. Фленов, М. Библия С# / М.Фленов. - СПб.: Питер, 2011.