

Министерство науки и высшего образования РФ  
ФГБОУ ВО «Томский государственный университет  
систем управления и радиоэлектроники»  
Кафедра безопасности информационных систем (БИС)

**С.С. Харченко**

# **ОСНОВЫ ПРОГРАММИРОВАНИЯ**

*Учебно-методическое пособие по курсовой работе*

*для студентов специальностей и направлений*

*10.03.01 – «Информационная безопасность»,*

*10.05.02 – «Информационная безопасность  
телекоммуникационных систем»,*

*10.05.03 – «Информационная безопасность  
автоматизированных систем»,*

*10.05.04 – «Информационно-аналитические системы безопасности»*

В-Спектр  
Томск, 2019

**УДК 004.42**

**ББК 32.973**

**X 22**

**X 22 Харченко С.С.** Основы программирования: учебно-методическое пособие по курсовой работе. – Томск: В-Спектр, 2019. – 48 с.

ISBN 978-5-91191-429-5

Учебно-методическое пособие по курсовой работе содержит теоретические сведения и методические указания, необходимые для выполнения курсовой работы по дисциплине «Основы программирования», и предназначено для студентов 2-го курса, обучающихся по специальностям: 10.05.03 «Информационная безопасность автоматизированных систем», 10.03.01 «Информационная безопасность», 10.05.02 «Информационная безопасность телекоммуникационных систем», 10.05.04 «Информационно-аналитические системы безопасности». Охарактеризованы требования к курсовой работе, описана структура, методика выполнения и защиты курсовой работы. В начале данного пособия, приводится глава с заданием для выполнения курсовой работы по данной дисциплине, далее в главах рассмотрено поэтапное выполнение курсовой работы на примере.

УДК 004.42

ББК 32.973

**ISBN 978-5-91191-429-5**

© Каф. безопасности информационных систем  
ТУСУР, 2019

© С.С. Харченко, 2019

## СОДЕРЖАНИЕ

Предисловие .....	4
Глава 1. Задание на курсовую работу .....	5
Глава 2. Проектирование разрабатываемого приложения .....	6
Глава 3. Разработка приложения на языке программирования С# .....	11
Глава 4. Использование sqlite в разработке приложений на языке программирования С# .....	27
Глава 5. Тестирование приложения .....	32
Глава 6. Варианты заданий на курсовую работу .....	37
Литература.....	45
Приложение А. Форма задания на курсовую работу. ....	46

## ПРЕДИСЛОВИЕ

Цель дисциплины «Основы программирования» – научить студентов строить алгоритмы и реализовывать их на компьютере в виде программ. Решать различные задачи по обработке информации и моделированию, применяемые при разработке программных и программно-аппаратных компонентов защищенных автоматизированных систем.

Таким образом, согласно ФГОС процесс изучения дисциплины «Основы программирования» направлен на формирование следующих компетенций:

ОПК-3 способностью применять языки, системы и инструментальные средства программирования в профессиональной деятельности;

ОПК-4 способностью понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации;

В результате изучения дисциплины обучающийся должен:

- **знать** язык программирования высокого уровня.
- **уметь** проектировать и кодировать алгоритмы с соблюдением требований к качественному стилю программирования; реализовывать основные структуры данных и базовые алгоритмы средствами языков программирования.
- **владеть** навыками разработки, документирования, тестирования и отладки программного обеспечения в соответствии с современными технологиями и методами программирования; навыками разработки программной документации; навыками программирования с использованием эффективных реализаций структур данных и алгоритмов.

Указанные задачи частично могут быть решены с помощью данного учебного пособия.

## Глава 1

### ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Выполнение курсовой работы по дисциплине «Основы программирования» требует решения одной из перечисленных в разделе 6 задач и, как результат, создания программы на языке программирования C# и написания пояснительной записки к работе. Использование другого языка программирования и выбор произвольной темы допускается только после согласования с преподавателем. Порядок и объем работ, а также требования к содержанию представлены ниже.

На первом занятии необходимо согласовать тему курсовой работы и стек используемых технологий с преподавателем, после чего оформить задание на курсовую работу, пример задания на курсовую работу представлен в приложении А. Пояснительная записка должна содержать следующие разделы:

- титульный лист;
- реферат на русском и английском языках;
- задание на работу;
- содержание;
- обзор;
- проектирование;
- тестирование;
- отладка(опционально);
- заключение;
- список использованных источников;
- приложение А – обязательное(исходные коды программы);

Содержательная часть разделов пояснительной записки рассматривается в разделах 2–5 данного пособия. Основные требования предъявляемые к разрабатываемому программному обеспечению:

- десктопное приложение;
- наличие механизма авторизации и регистрации пользователя;
- взаимодействие с базой данных;
- наличие раздела помощи с краткой инструкцией пользователя;

Особых требований к количеству таблиц базы данных не предъявляется. Перед тем как приступить к выполнению курсовой работы необходимо в полном объеме ознакомиться с данным пособием. В ходе выполнения курсовой работы рекомендуется ознакомиться со списком литературы, который прилагается к данному пособию. Пояснительная записка должна быть оформлена в соответствии ОС ТУСУР 01–2013 или ГОСТ 7.32–2017.

## Глава 2

### ПРОЕКТИРОВАНИЕ РАЗРАБАТЫВАЕМОГО ПРИЛОЖЕНИЯ

Проектирование разрабатываемого программного обеспечения один из самых главных этапов разработки, на этом этапе необходимо определиться с технологиями и алгоритмами, которые будут использованы. Проектирование программного обеспечения регламентируется ГОСТ Р 51904-2002 «Программное обеспечение встроенных систем. Общие требования к разработке и документированию». В рамках курсовой работы этап проектирование должен содержать следующие разделы:

- обоснование выбранных технологий;
- описание алгоритма;
- определение временной сложности алгоритма;
- блок-схема основного алгоритма;
- проектирование структуры программы;
- база данных.

#### Обоснование выбранных технологий

В разделе «Обоснование выбранных технологий» необходимо изучить разнообразные средства разработки, языки программирования и фреймворки, которые позволяют разрабатывать полноценные десктопные и/или мобильные приложения. Провести их сравнительный анализ, для этого выбрать наиболее значимые для Вас критерии и построить сравнительную таблицу рассматриваемых средств. После чего сделать выводы о наиболее подходящих для вас технологий. Пример сравнения языков программирования представлен на табл. 1.

Т а б л и ц а 1

#### Сравнение языков программирования

	Парадигма	Типизация	Управление памятью
C++	ООП, Императив	Статическая, сильная	Руководство
C#	ООП, Императив	Статическая, явная	Автоматическое
Java	ООП, Императив	Статическая, сильная	Автоматическое
Python	ООП, Императив	Динамическая, сильная	Автоматическое

#### Описание алгоритма

В разделе «Описание алгоритма» необходимо дать словесное описание алгоритма по шагам и представить его в виде блок-схемы. Например, перед нами стоит задача: «Написать программу, которая приглашает ввести последовательно дробных чисел и вычисляет их среднее арифметическое», тогда алгоритм решения на естественном языке можно представить в следующем виде:

Алгоритм А:

A1. I ← 1;

А2. Ввод  $a$ ;  
 А3.  $S \leftarrow S + a$ ;  
 А4.  $I \leftarrow I + 1$ ;  
 А5. Если  $I \leq 5$ , то возврат к А2, иначе возврат к А6;  
 А6.  $S \leftarrow S/5$ ;  
 А7. Вывод  $S$ ;  
 Блок-схема алгоритма А представлена на рис. 1.

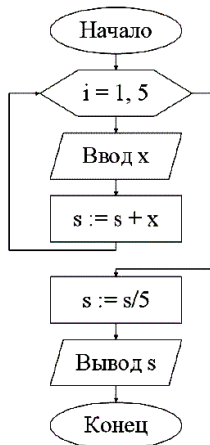


Рис. 1. Блок-схема алгоритма А

Алгоритмы необходимо составить для всех подзадач, которые предстоит решить в ходе выполнения курсовой работы. При составлении блок-схем алгоритмов необходимо руководствоваться – ГОСТ 19.701–90 «Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения».

### **Определение временной сложности алгоритма**

Под временной сложностью алгоритма  $T$  стоит понимать некоторую функцию времени работы алгоритма от объема входных данных [1]. Обозначается как  $O$  (произносится как «о большое»). Существует несколько видов функций временной сложности, они представлены ниже в порядке возрастания времени выполнения алгоритма:

1.  $C$
2.  $\log(\log(N))$
3.  $\log(N)$
4.  $N^C, 0 < C < 1$
5.  $N$
6.  $N \cdot \log(N)$

7.  $N^C, C > 1$
8.  $C^N, C > 1$
9.  $N!$

где  $C$  – некоторая константа, существуют и другие более сложные виды функций временной сложности. Также стоит отметить, что существуют и другие нотации временной сложности кроме  $O$ , в рамках данной работы они рассматриваться не будут.  $O$  показывает временную сложность алгоритма при наихудшем стечении обстоятельств.

Например, перед нами стоит задача сортировки массива из  $N$  чисел, тогда  $N$  – это и есть объем входных данных. Как вам уже известно существует некоторое количество алгоритмов способных решить эту задачу, все они обладают разной временной сложностью. Временная сложность алгоритмов сортировки представлена в табл. 2.

Таблица 2

<b>Временная сложность алгоритмов сортировки</b>	
Алгоритм	Временная сложность
Быстрая сортировка	$O(N^2)$
Сортировка слиянием	$O(N \cdot \log(N))$
Пирамидальная сортировка	$O(N \cdot \log(N))$
Пузырьковая сортировка	$O(N^2)$
Сортировка вставками	$O(N^2)$
Сортировка выбором	$O(N^2)$
Блочная сортировка	$O(N^2)$

Рассмотрим нахождение вида функции временной сложности на примере сортировки пузырьком. Алгоритм сортировки пузырьком использует вложенные циклы. Во внутреннем поочередно сравниваются элементы, если они стоят в неправильном порядке, то выполняется перестановка элементов местами. Внешний цикл выполняется до тех пор, пока в массиве найдется хоть одна пара неотсортированных элементов. Трудоемкость функции перестановки местами элементов в массиве не зависит от объема входных данных ( $N$ ), следовательно, ее можно принять равной 1 (т.е., постоянная). В результате выполнения внутреннего цикла, наибольший элемент смещается в конец массива неупорядоченной части, поэтому через  $N$  таких вызовов массив в любом случае окажется отсортирован. Если же массив отсортирован, то внутренний цикл будет выполнен лишь один раз. Математически это представляется следующим образом:

$$T = O\left(\sum_{i=1}^N \sum_{j=1}^{N-1} 1\right) = O\left(\sum_{i=1}^N N\right) = O(N^2),$$

где  $T$  – временная сложность алгоритма.



## Проектирование структуры программы

В этом разделе необходимо построить две диаграммы – диаграмму прецедентов и диаграмму классов. Диаграмма прецедентов определяет функциональные требования к системе и описывает типичные взаимодействия между пользователями системы и самой системой, а также описывает процесс ее функционирования.

Для построения диаграммы прецедентов, используется язык графического описания моделей – UML [2]. Для построения диаграммы прецедентов используются следующие термины UML – сценарий, актер, прецедент, include, граница системы. Примеры простой диаграммы прецедентов представлены на рис. 2, 3.

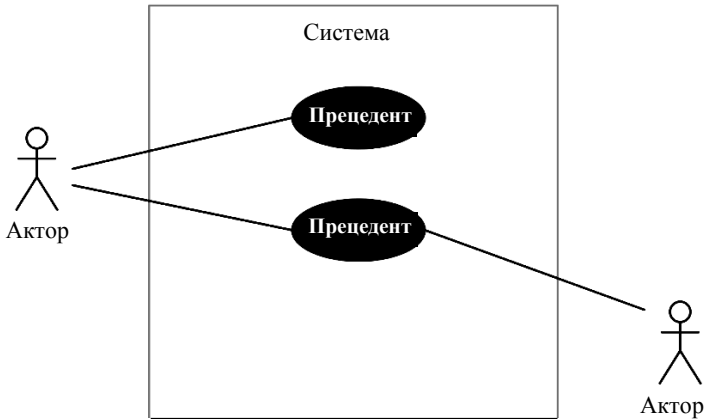


Рис. 2. Общий вид диаграммы прецедентов

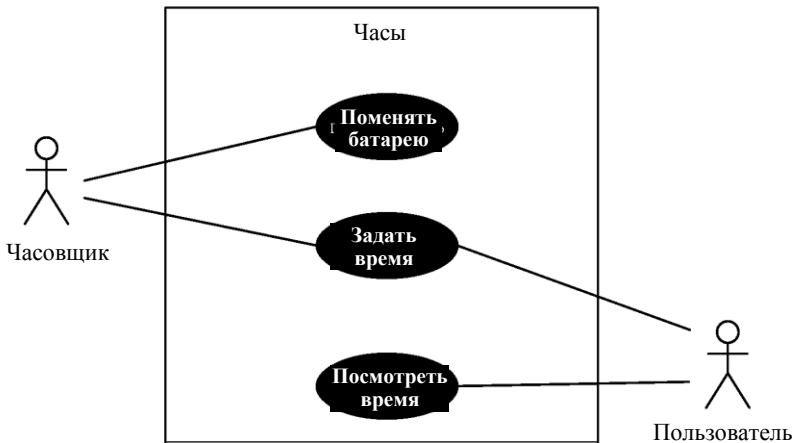


Рис. 3. Пример диаграммы прецедентов для ручных часов

Для построения диаграммы классов также описывается при помощи языка UML и демонстрирует общую структуру иерархии классов системы, их атрибутов методов и интерфейсов. В нотации UML существует 6 основных типов взаимодействия классов, каждый со своим графическим обозначением (рис. 4)



Рис. 4. Обозначение типов взаимодействия классов в языке UML

Для задания видимости членов класса также используются специальные графические обозначения: «+» – public доступ, «-» – private доступ, «#» – protected доступ (рис. 5).

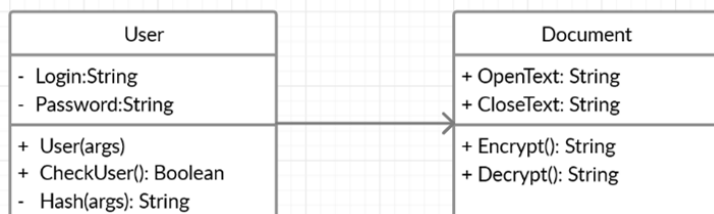


Рис. 5. UML диаграмма взаимодействия классов

### База данных

В разделе «Базы данных» необходимо построить структуру базы данных, описать основные сущности на концептуальном уровне используя ER-модель (модель объект–отношение) [3]. Определить какие таблицы вам необходимы и выделить первичные и внешние ключи планируемой базы данных. Пример простой ER-диаграммы представлен на рис. 6.

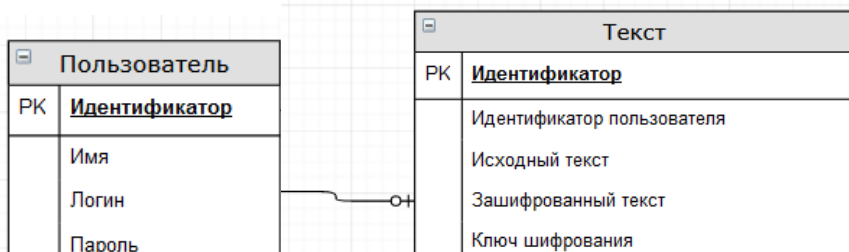


Рис. 6. ER-диаграмма

## Глава 3

### РАЗРАБОТКА ПРИЛОЖЕНИЯ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#

После запуска Visual Studio 2019 при помощи нажатия горячих клавиш Shift+Ctrl+N запускаем меню создания нового проекта, в форме поиска шаблона приложения введите «windows forms»(рис. 7), далее выберите проект «**Приложение Windows Forms(.Net Framework)**».

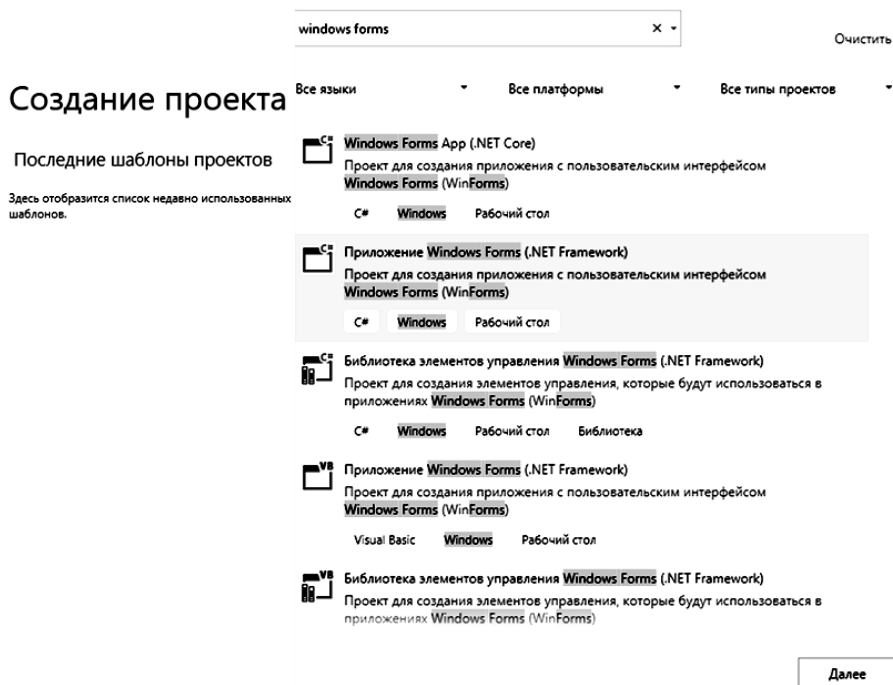


Рис. 7. Меню создания нового проекта

После выбора типа проекта откроется меню настройки проекта, где необходимо выбрать название проекта, его расположение на диске, версию .Net Framework и другие характеристики (рис. 8).

После создания проекта откроется главное окно Visual Studio для работы с проектом (рис. 9). В правой части окна располагается обозреватель решений, в котором можно увидеть всю иерархию файлов проекта. В левой части располагается панель элементов, в которой можно найти элементы

управления, которые можно располагать на формах и управлять ими: кнопки, текстовые поля ввода, контекстные меню и т.д.

## Настроить новый проект

Приложение Windows Forms (.NET Framework) C# Windows Рабочий стол

Имя проекта

Расположение

 ...

Решение

Имя решения ⓘ

Поместить решение и проект в одном каталоге

Платформа

Назад Создать

Рис. 8. Меню настройки проекта

Добавим еще одну форму в проект, для этого вызвав контекстное меню для проекта, выбираем меню «Добавить», далее меню «Форма Windows...» (рис. 10).

После создания формы можно ее переименовать в **Обзревателе решений**, после чего среда предложит переименовать все ссылки в проекте на новое название формы, предпочтительно принять это предложение (рис. 11).

Для изменения настроек формы используется раздел «Свойства», так, например, чтобы задать заголовок формы – необходимо поменять свойство «Text» для нужной формы (рис. 12).

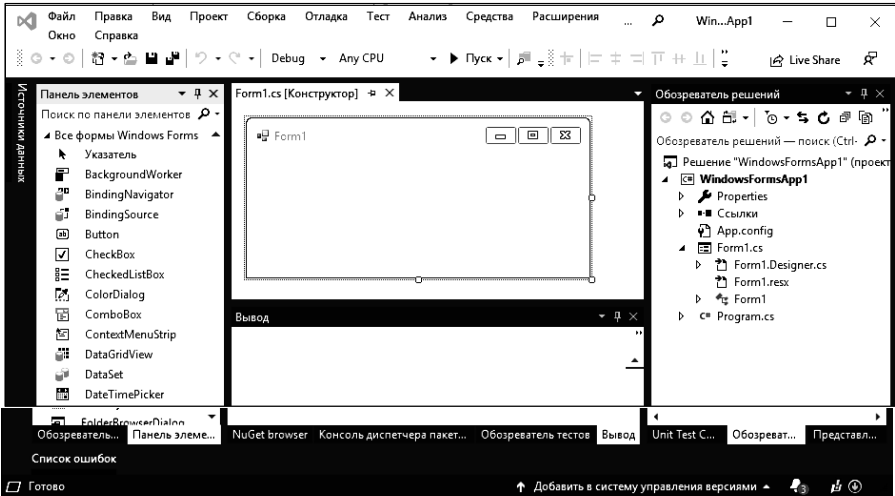


Рис. 9. Главное окно Visual Studio с проектом Windows Forms

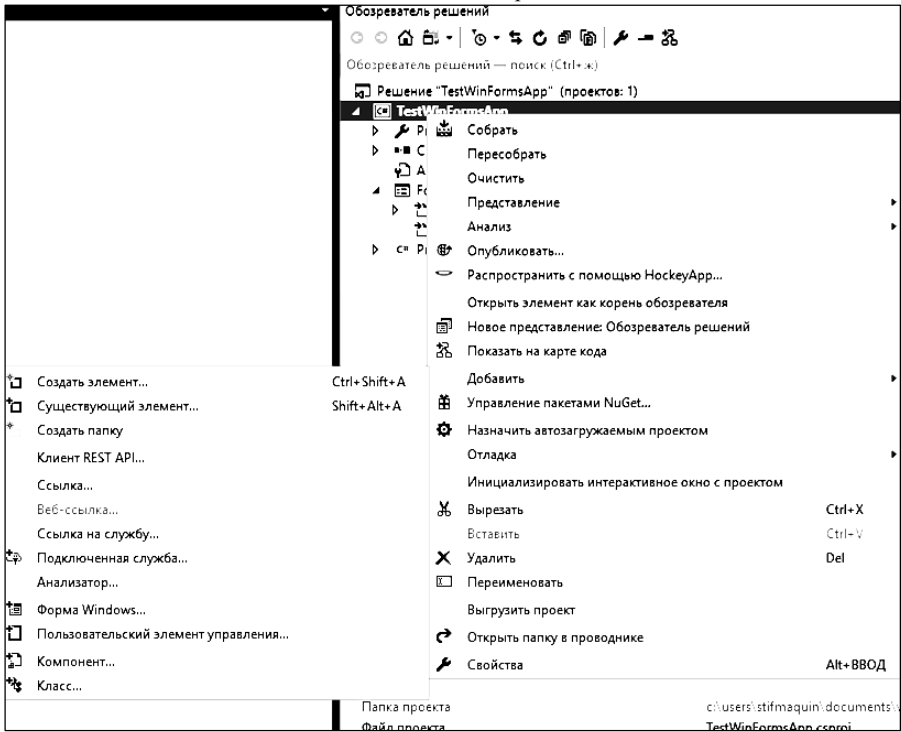


Рис. 10. Контекстное меню добавления элемента в проект

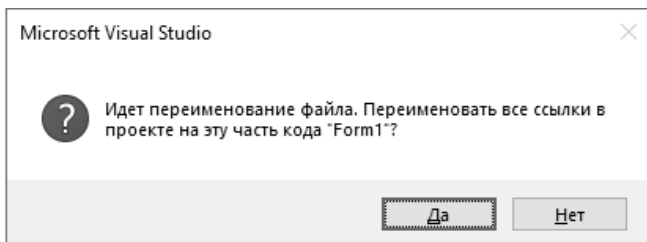


Рис. 11. Уведомление о переименовании ссылок в проекте

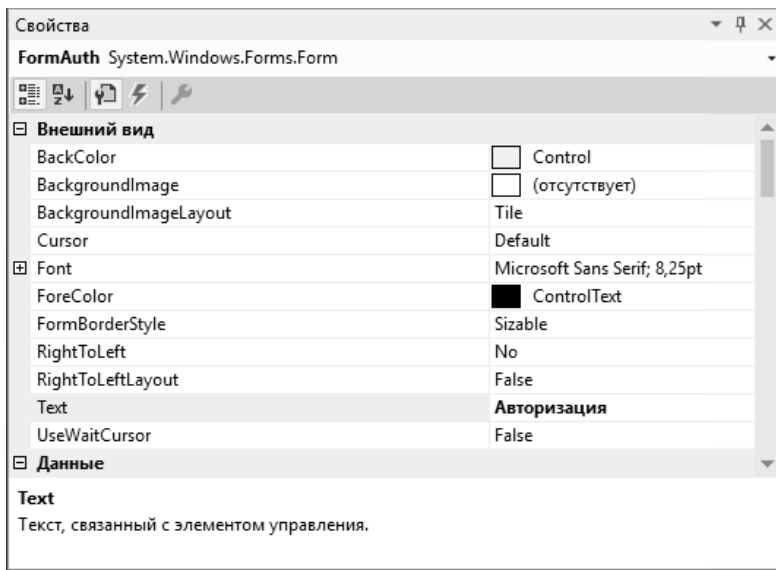


Рис. 12. Панель управления свойствами формы

Порядок запуска приложения задан в файле **Program.cs**, так, если необходимо запускать при старте программы не форму, созданную по умолчанию, например, не `FormMain`, а `FormAuth`, необходимо строку: «`Application.Run(new FormMain());`» заменить на «`Application.Run(new FormAuth());`», в результате код запуска приложения примет следующий вид:

```
using System;
using System.Windows.Forms;

namespace BasicsOfProgramming
{
    static class Program
    {
```

```

/// <summary>
/// Главная точка входа для приложения.
/// </summary>
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new FormAuth());
}
}
}

```

Теперь после запуска приложения будет стартовать форма авторизации, а не основная форма приложения.

Настроим форму авторизации, для этого добавим на нее два элемента типа – «Textbox» для ввода в них логина и пароля, и два элемента типа – «Button». Для добавления элементов необходимо найти необходимый элемент в списке панели элементов, нажать на него мышью и не отпуская ЛКМ перетащить элемент на форму (рис. 13).

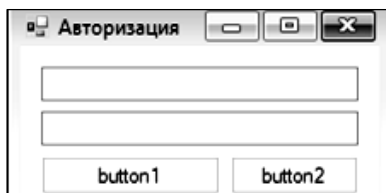


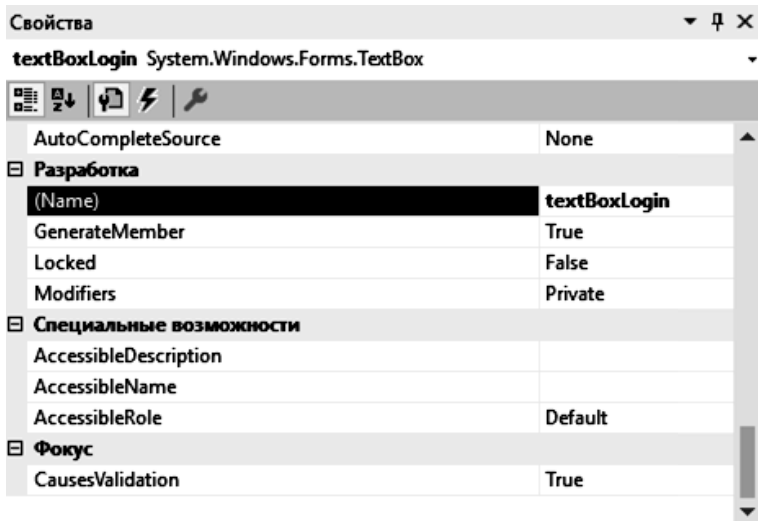
Рис. 13. Конструктор формы авторизации

По умолчанию все элементы именуются по типу и нумеруются по мере добавления, так при добавлении двух элементов типа «Textbox» – они будут именоваться «textBox1» и «textBox2», однако это становится неудобно при добавлении большого количества элементов в проект. Поэтому рекомендуется сразу после добавления переименовывать все элементы на форме (рис. 14–17).

Также необходимо помнить, что свойство «Text» автоматически не меняется при изменении названия элемента. Необходимо его изменить для обоих кнопок, которые расположены на форме (рис. 18, 19).

После изменения свойства «Text» у элементов текст автоматически меняется и в конструкторе форм (рис. 20).

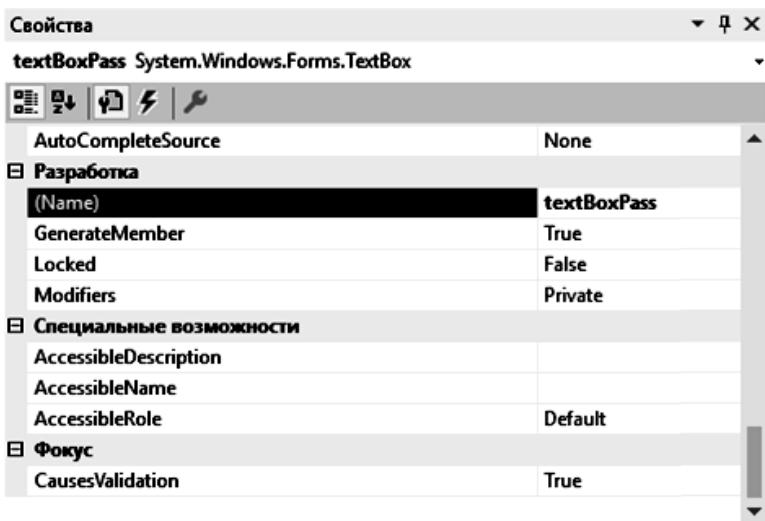
Для того чтобы для формы авторизации убрать кнопки «свернуть» и «развернуть/свернуть окно на весь экран», необходимо свойства – «MaximizeBox» и «MinimizeBox2» установить в **False** для формы FormAuth (рис. 21, 22).



**(Name)**

Указывает имя, используемое в коде для идентификации объекта.

Рис. 14. Свойства элемента ввода логина



**(Name)**

Указывает имя, используемое в коде для идентификации объекта.

Рис. 15. Свойства элемента ввода пароля



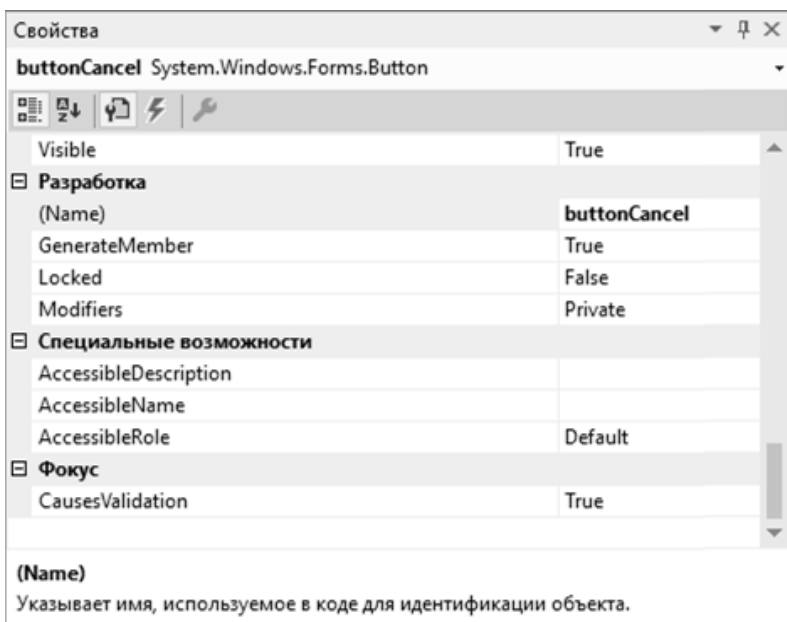


Рис. 16. Свойства элемента кнопки отмены

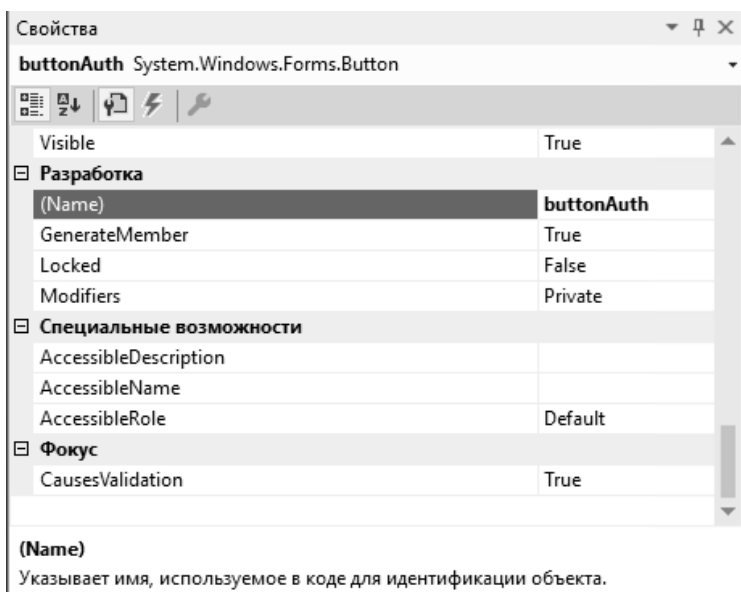


Рис. 17. Свойства элемента кнопки авторизации

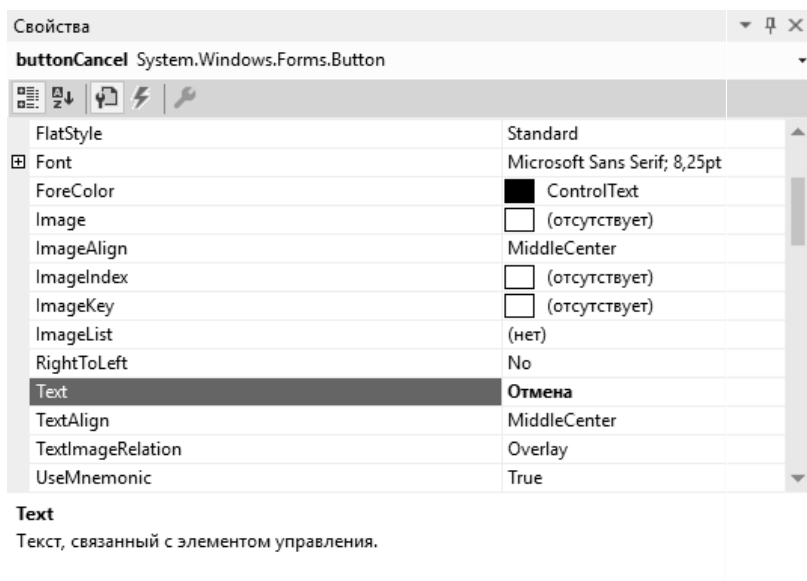


Рис. 18. Изменения свойства Text элемента кнопки авторизации

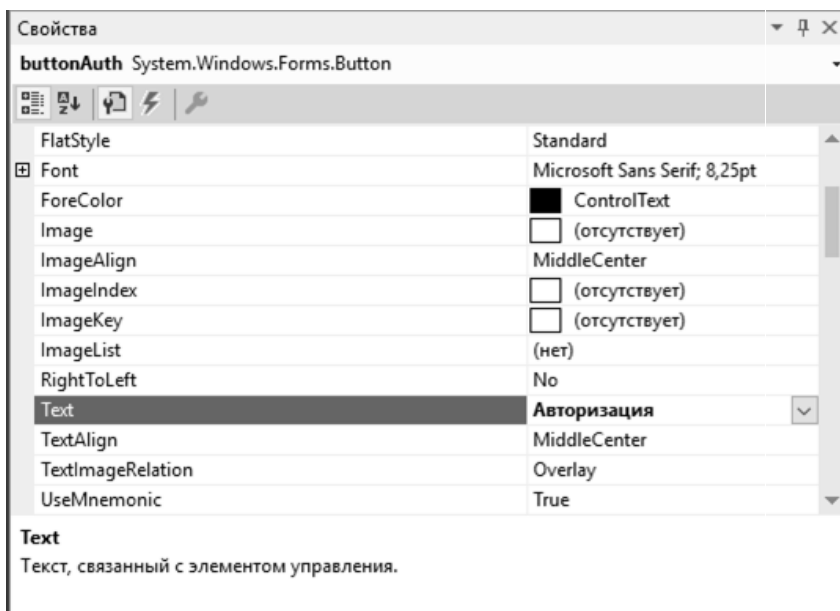


Рис. 19. Изменения свойства Text элемента кнопки авторизации

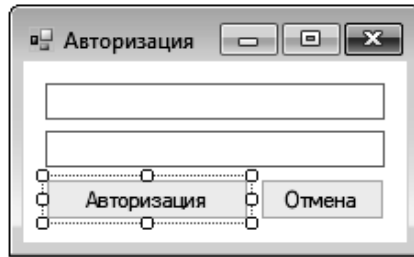



Рис. 20. Форма авторизации после кастомизации элементов

Свойства

**FormAuth** System.Windows.Forms.Form

(Name)	<b>FormAuth</b>
Language	(По умолчанию)
Localizable	False
Locked	False
<b>Специальные возможности</b>	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
<b>Стиль окна</b>	
ControlBox	True
HelpButton	False
Icon	 (Значок)
IsMdiContainer	False
MainMenuStrip	(нет)
MaximizeBox	False
MinimizeBox	False
Opacity	100%
ShowIcon	True
ShowInTaskbar	True
SizeGripStyle	Auto
TopMost	False
TransparencyKey	<input type="checkbox"/>
<b>Фокус</b>	
CausesValidation	True

**MinimizeBox**  
 Определяет, содержит ли форма в правом верхнем углу строки заголовка значок свертывания.

Рис. 21. Удаление возможности минимизации формы

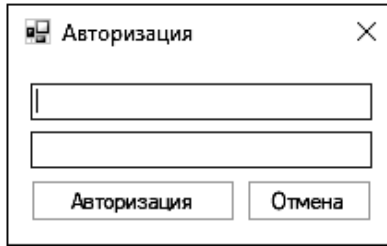


Рис. 22. Форма авторизации после ее кастомизации

Шаблон формы для авторизации готов, но не наделен никаким функционалом, так если мы будем нажимать кнопки – «Авторизация» или «Отмена» – ничего не произойдет, а при вводе текста в поле для пароля ввод будет осуществляться в открытом виде, что недопустимо (рис. 23).

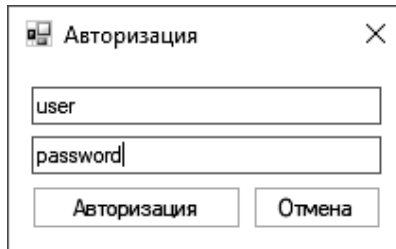


Рис. 23. Взаимодействие с элементами управления формы авторизации

Для того чтобы скрывать символы, вводимые в поле для пароля, зададим свойство **PasswordChar** равное «\*» (рис. 24).

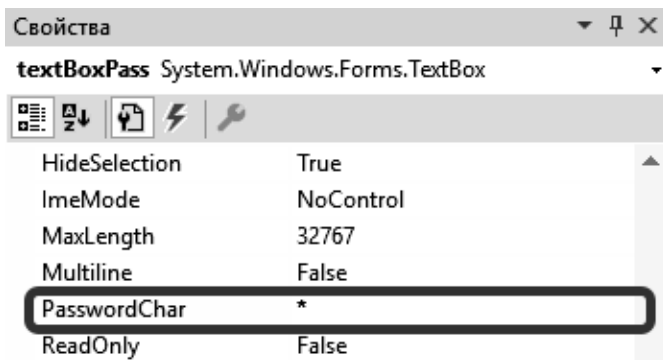


Рис. 24. Указание символа для отображения пароля при вводе однострочном поле редактирования

Для того чтобы запустить приложение и убедиться, что символы вводимые в поле пароль не будут отображаться пользователю можно воспользоваться горячими клавишами **Ctrl+F5** (рис. 25).

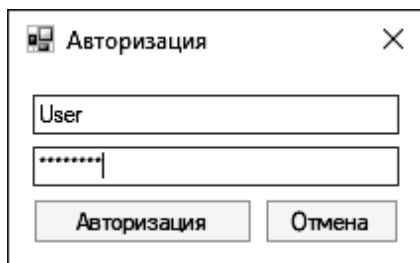


Рис. 25. Ввод пароля на форме авторизации

Для того чтобы задать функционал нажатия кнопки «Отмена» необходимо дважды кликнуть на кнопку в конструкторе форм, это сгенерирует обработчик нажатия на кнопку:

```
using System.Windows.Forms;
namespace BasicsOfProgramming
{
    public partial class FormAuth : Form
    {
        public FormAuth()
        {
            InitializeComponent();
        }

        private void buttonCancel_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Для того чтобы закрыть форму, необходимо вызвать соответствующий метод – **Close()**.

Таким образом код события нажатия на кнопку примет следующий вид:

```
private void buttonCancel_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Ключевое слово – «this» используется для обращения к текущему объекту (экземпляру класса) формы FormAuth, его использование не является обязательным, однако добавляет понимание кода при прочтении. Для реализации механизма авторизации создадим класс User, для этого через контекстное меню проекта добавим одноименный файл (рис. 26, 27).

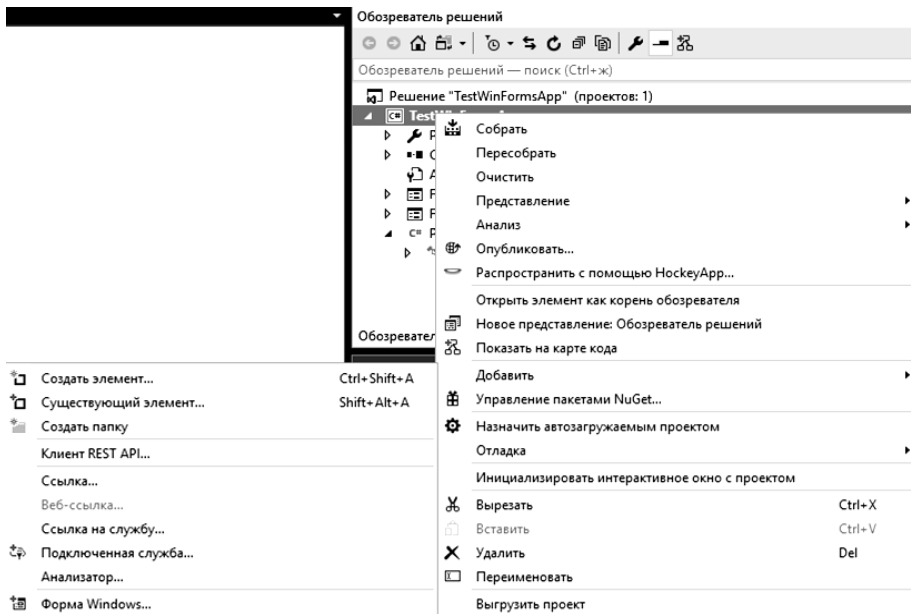


Рис. 26. Добавление элемента в проект

Добавление нового элемента - BasicsOfProgramming

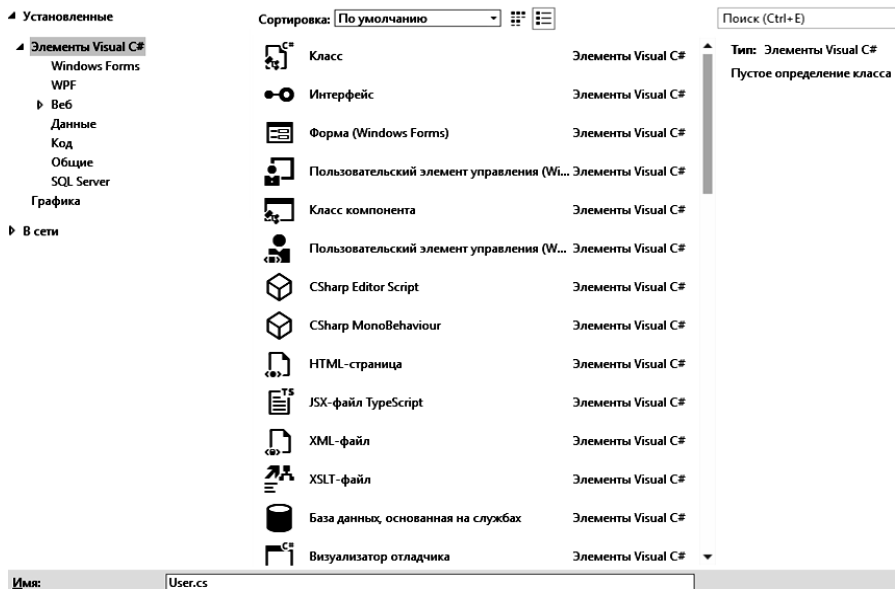


Рис. 27. Добавление класса к проекту

После создания класса отредактируем его, создадим два поля – для хранения логина и пароля, а также метод проверки существования пользователя:

```
public class User
{
    public string Login { get; }
    public string Password { get; }

    public User(string login, string password)
    {
        Login = login;
        Password = password;
    }
    public bool CheckUser()
    {
        return true;
    }
}
```

Метод **CheckUser** в дальнейшем необходимо переопределить, чтобы он проверял по базе данных наличия пользователя с заданными логином и паролем. Теперь можно вернуться к определению функционала кнопки «Авторизация»:

```
private void buttonAuth_Click(object sender, EventArgs e)
{
    if (new User(textBoxLogin.Text, textBoxPass.Text).CheckUser())
    {
        this.Hide();
        var formMain = new FormMain();
        formMain.Closed += (s, args) => this.Close();
        formMain.Show();
    }
    else
    {
        this.Close();
    }
}
```

Теперь после нажатия кнопки «Авторизация» происходит следующее:

- скрывается форма **FormAuth**;
- создается новая форма **FormMain**;
- для события **Closed** основной формы добавляется вызов функции **Close** формы авторизации;
- созданная новая форма отображается на экране.

Поскольку правильнее пароль в базе данных хранить в виде хэш-строки, понадобится метод, который будет делать преобразование пароля в хэш-

строку. Например, для использования хеширования алгоритмом SHA1 метод будет иметь следующий вид:

```
private string Hash(string input)
{
    using (SHA1Managed sha1 = new SHA1Managed())
    {
        var hash = sha1.ComputeHash(Encoding.UTF8.GetBytes(input));
        var sb = new StringBuilder(hash.Length * 2);

        foreach (byte b in hash)
        {
            sb.Append(b.ToString("x2"));
        }

        return sb.ToString();
    }
}
```

В приведенном методе используется класс **SHA1Managed**, чтобы компиляция прошла успешно нужно подключить соответствующую библиотеку – **System.Security.Cryptography**, кроме того, необходимо подключить библиотеку **System.Text**, чтобы была возможность использовать класс **StringBuilder**.

Допустим перед нами стоит задача создания приложения, которое позволяет для двух заданных пользователем чисел, выполнять простейшие операции над числами и выводить ответ. Для этого добавим на форму элементы управления, которые позволяют манипулировать данными (рис. 28).

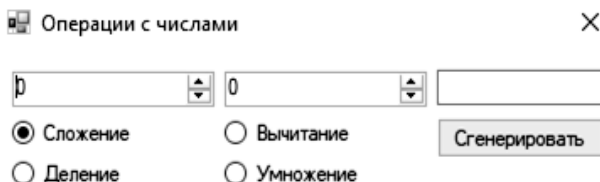


Рис. 28. Форма «Операции с числами»

Для того чтобы работать с числами создадим класс **Calculator**, конструктор которого принимает два целых числа, и имеет 4 метода, которые описывают простые арифметические операции и добавим его к нашему проекту.

```
public class Calculator
{
    private int firstNumber, secondNumber;

    public Calculator(int firstNumber, int secondNumber)
    {
        this.firstNumber = firstNumber;
    }
}
```



```

        this.secondNumber = secondNumber;
    }
    public int GetAdd()
    {
        return firstNumber + secondNumber;
    }
    public double GetDiv()
    {
        return firstNumber / secondNumber;
    }
    public int GetSub()
    {
        return firstNumber - secondNumber;
    }
    public int GetMult()
    {
        return firstNumber * secondNumber;
    }
}

```

Для того чтобы использовать этот класс и вычислять значения выбранные в элементах numericUpDownA и numericUpDownB, необходимо написать обработчик события – изменения значения radioButton, расположенных на форме, для этого в конструкторе формы, дважды кликните по элементу radioButtonAdd, который отвечает за сложение. После этого автоматически будет сгенерирован пустой обработчик события изменения значения для этого элемента, который необходимо заполнить кодом.

```

private void radioButtonAdd_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        int a = Convert.ToInt32(numericUpDownA.Value);
        int b = Convert.ToInt32(numericUpDownB.Value);
        textBoxResult.Text = new Calculator(a, b).GetAdd().ToString();
    }
}

```

Аналогично этому, необходимо наделить функционалом и другие элементы radioButton расположенные на форме.

```

private void radioButtonDiv_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {

```

```

        int a = Convert.ToInt32(numericUpDownA.Value);
        int b = Convert.ToInt32(numericUpDownB.Value);
        textBoxResult.Text = new Calculator(a, b).GetDiv().ToString();
    }
}

private void radioButtonSub_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        int a = Convert.ToInt32(numericUpDownA.Value);
        int b = Convert.ToInt32(numericUpDownB.Value);
        textBoxResult.Text = new Calculator(a, b).GetSub().ToString();
    }
}

private void radioButtonMult_CheckedChanged(object sender, EventArgs e)
{
    if (((RadioButton)sender).Checked)
    {
        int a = Convert.ToInt32(numericUpDownA.Value);
        int b = Convert.ToInt32(numericUpDownB.Value);
        textBoxResult.Text = new Calculator(a, b).GetMult().ToString();
    }
}

```

Также необходимо наделить функционалом кнопку «Сгенерировать», для этого можно использовать событие нажатия на кнопку и класс Random.

```

private void buttonnRandomize_Click(object sender, EventArgs e)
{
    numericUpDownA.Value = new Random().Next();
    numericUpDownB.Value = new Random().Next();
}

```

## Глава 4

### ИСПОЛЬЗОВАНИЕ SQLITE В РАЗРАБОТКЕ ПРИЛОЖЕНИЙ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C#

Простейший вариант добавления библиотек SQLite в свой проект – использовать пакетный менеджер NuGet. Для этого кликаем правой кнопкой мыши на разделе Ссылки в проекте и в появившемся контекстном меню выбираем пункт Управление пакетами Nuget. Далее в вкладке «Обзор» ищем нужный нам пакет и устанавливаем его в проект.

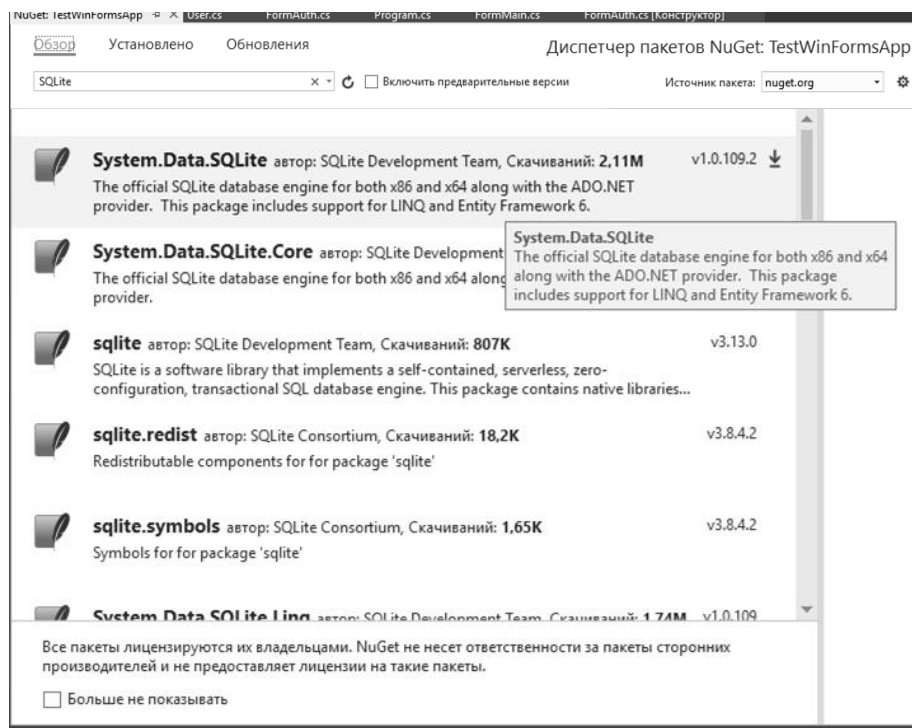


Рис. 28. Поиск пакетов Nuget в среде Visual Studio

В отличие от подавляющего большинства других баз данных, SQLite не имеет строгой типизации хранимых данных. Это сделано в целях обеспечения максимальной совместимости синтаксиса SQL-запросов с другими SQL-движками. Однако определенный набор правил все же есть.

SQLite имеет пять классов хранимых данных:

1. NULL – null
2. INTEGER – целое число
3. REAL – вещественное число
4. TEXT – текст
5. BLOB – блок данных

В SQLite нет логического типа данных, поэтому значение TRUE и FALSE предполагается храниться в виде 1 и 0 класса INTEGER.

Также SQLite не содержит отдельного класса для хранения даты и времени. На усмотрение разработчика предлагается три варианта хранения:

1. INTEGER – в виде целого числа секунд, прошедших от точки отчета 1970-01-01 00:00:00 UTC. Это наиболее популярный и быстрый вариант, знакомый поклонникам Unix.

2. TEXT – в виде строкового значения по стандарту ISO8601 («YYYY-MM-DD HH:MM:SS.SSS»)

3. REAL – число дней, прошедших с полудня по Гринвичу 24 ноября 4714 года до н.э. по Юлианскому календарю.

За исключением колонки, выделенной для ключей записей, любые другие колонки таблиц могут хранить данные любого типа. Однако разработчик может задать для колонок predetermined типы сопоставлений, которые будут определять тип и преобразование к нему вводимых данных. В колонке типа TEXT могут храниться данные классов NULL, TEXT и BLOB, при записи числового значения оно преобразуется в текст. При создании таблиц их колонки получают свои типы сопоставления в следующем порядке:

Если описание колонки содержит подстроку «INT», то ей присваивается тип сопоставления INTEGER. Если описание колонки содержит подстроки «CHAR», «CLOB» или «TEXT», то ей присваивается тип сопоставления TEXT. Если описание колонки содержит подстроку «BLOB», то ей присваивается тип сопоставления NONE. Если описание колонки содержит подстроки «REAL», «FLOAT» или «DOUBLE», то ей присваивается тип сопоставления REAL. Во всех остальных случаях колонка получает тип NUMERIC. По умолчанию, каждая запись таблицы SQLite содержит ключ – 64-битное целое уникальное число, позволяющее однозначно идентифицировать запись. В тексте SQL-запроса это значение можно считать с помощью predetermined названий «rowid», «oid», «\_rowid\_». Данные значения ключей не генерируются движком, если при создании таблицы был указан параметр WITHOUT ROWID.

Если при создании таблицы одна из колонок имеет тип INTEGER и параметр PRIMARY KEY в том или ином виде, то эта колонка автоматически становится псевдонимом для значений ключей записей таблицы:

```
CREATE TABLE t(id INTEGER PRIMARY KEY, title);  
CREATE TABLE t(id INTEGER, title, PRIMARY KEY(id));
```

В этом случае значения колонок, полученных в результате выполнения запроса «SELECT id, rowid FROM t» будут идентичными.

По умолчанию, если значения ROWID достигли максимального значения INTEGER, то новые значения будут подбираться среди ранее использованных значений ключей удаленных записей. Если разработчику необходимо, чтобы ключевые значения всегда оставались уникальными, то при создании таблицы в описании ключевой колонки следует указать параметр AUTOINCREMENT: «CREATE TABLE t(id INTEGER PRIMARY KEY AUTOINCREMENT, title)». Значения ключей могут задаваться вручную непосредственно в командах INSERT и UPDATE: «INSERT INTO t(rowid, title) VALUES(10, 'One')».

Для примера работы объектами класса SQLiteCommand создадим класс Database.cs и добавим в него функцию инициализации базы данных:

```
public class Database
{
    private readonly string dataSource;
    public Database(string dataSource)
    {
        this.dataSource = dataSource;
    }
    public bool InitializeDatabase()
    {
        SQLiteConnection conn = new SQLiteConnection(dataSource);
        try
        {
            conn.Open();
            if (conn.State == ConnectionState.Open)
            {
                SQLiteCommand cmd = conn.CreateCommand();
                string sql_command = "DROP TABLE IF EXISTS users;"
                    + "CREATE TABLE users("
                    + "id INTEGER PRIMARY KEY AUTOINCREMENT, "
                    + "login TEXT, "
                    + "password TEXT, "
                    + "role TEXT; ";
                cmd.CommandText = sql_command;
                cmd.ExecuteNonQuery();
            }
        }
        catch
        {
            return false;
        }
        finally
        {
            conn.Dispose();
        }
        return true;
    }
}
```

В приведенном примере новый объект класса `SQLiteCommand` создается вызовом метода `CreateCommand()` ранее созданного объекта класса `SQLiteConnection`. В зависимости от сценария работы также можно использовать конструкции:

- `SQLiteCommand cmd = new SQLiteCommand(conn);`
- `SQLiteCommand cmd = new SQLiteCommand();`
- `cmd.Connection = conn;`

Текст SQL-команды записывается в свойство `CommandText`. Строку с командой или содержащую ее переменную также можно передать прямо в конструктор, например:

- `SQLiteCommand cmd = new SQLiteCommand(«SELECT * FROM users»);`
- `SQLiteCommand cmd = new SQLiteCommand(sql_command, conn);`

Выполнение SQL-команды, не предполагающей возвращения данных, выполняется вызовом метода `ExecuteNonQuery()`. Выполнение SQL-запроса, предполагающего возвращение единственного значения, выполняется вызовом метода `ExecuteScalar()`, используя этот метод добавим в класс `Database` метод, который возвращает количество записей в таблице пользователи в базе данных:

```
public int UsersCount()
{
    SQLiteConnection conn = new SQLiteConnection(dataSource);
    try
    {
        conn.Open();
        if (conn.State == ConnectionState.Open)
        {
            SQLiteCommand cmd = conn.CreateCommand();
            string sql_command = "SELECT count(id) FROM users";
            cmd.CommandText = sql_command;
            return (int)cmd.ExecuteScalar();
        }
    }
    catch
    {
        return -1;
    }
    finally
    {
        conn.Dispose();
    }
    return -1;
}
```

Для выполнения SQL-команды, предполагающей возврат множества данных, используются два подхода, каждый из которых имеют как плюсы, так

и минусы. Первый – построчное считывание результатов выполнения команды с помощью объекта класса `SQLiteDataReader`. Реализация этого подхода требует большее, по сравнению со вторым, количество программного кода, однако дает разработчику несравнимо более богатый набор возможностей для контроля над процессом считывания и обработки данных из результата выполнения SQL-запроса.

Для определения имеется ли в базе данных пользователь с определенным логином и паролем создадим метод аналогичный методу `UsersCount`:

```
public bool ValidUser(string username, string password)
{
    SQLiteConnection conn = new SQLiteConnection(dataSource);
    try
    {
        conn.Open();
        if (conn.State == ConnectionState.Open)
        {
            SQLiteCommand cmd = conn.CreateCommand();
            cmd.CommandText = string.Format("SELECT login, password, role"
                + "FROM users"
                + "where login = '{0}' AND"
                + "password = '{1}'",
                username, password);
            var usersCount = (int)cmd.ExecuteScalar();
            return usersCount > 0;
        }
    }
    catch
    {
        return false;
    }
    finally
    {
        conn.Dispose();
    }
    return false;
}
```

## ГЛАВА 5

### ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Тестирование программного обеспечения – это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе контрольных примеров. Количество и сами контрольные примеры определяются тестировщиком исходя из его знаний и квалификации [4, 5]. Контрольный пример – это совокупность предварительных условий, входов (включая действия, где это применимо) и ожидаемых результатов, разработанных для управления выполнением элемента тестирования для достижения целей тестирования, включая корректную реализацию, идентификацию ошибок, проверку качества и получение другой значимой информации.

Существуют различные подходы и разновидности тестирования:

- функциональное тестирование;
- тестирование пользовательского интерфейса;
- тестирование безопасности;
- тестирование взаимодействия;
- нагрузочное тестирование;
- стрессовое тестирование;
- тестирование стабильности;
- объемное тестирование;
- тестирование установки;
- тестирование удобства пользования;
- тестирование на отказ и восстановление;
- конфигурационное тестирование;
- дымовое тестирование;
- регрессионное тестирование;
- повторное тестирование;
- тестирование сборки;
- санитарное тестирование.

В рамках курсовой работы необходимо провести функциональное тестирование, путем написания автоматических тестов и ручного тестирования пользовательского интерфейса. Для написания автоматических тестов в зависимости от используемых технологий существуют готовые библиотеки, которые позволяют ускорить и автоматизировать процесс тестирования функционала разрабатываемого приложения. Для .Net Framework популярна библиотека NUnit, чтобы добавить автоматические тесты к вашему приложению создайте новый проект в вашем решении (рис. 29) из шаблона «Тестовый проект NUnit» (рис. 30).



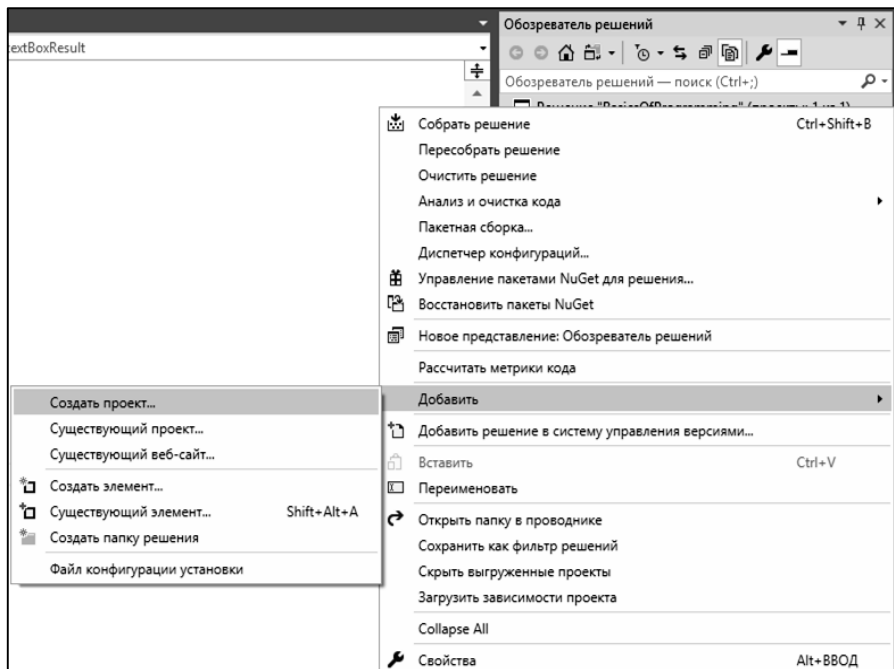


Рис. 29. Добавление нового проекта в решение

## Добавить новый проект

Последние шаблоны проектов

Приложение Windows Forms (.NET Framework)

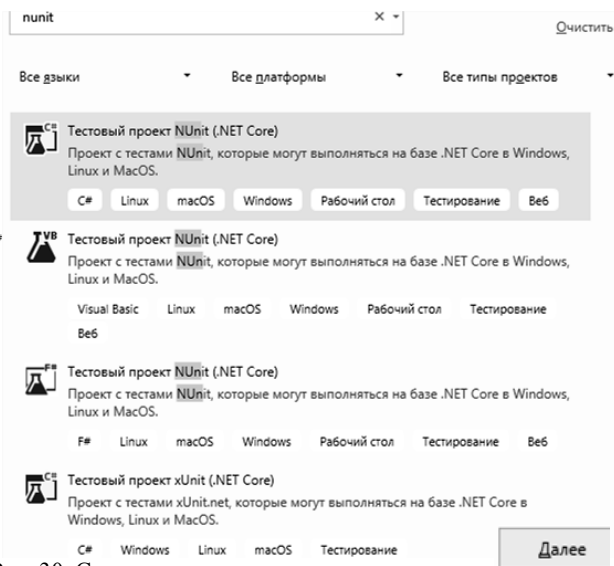


Рис. 30. Создание тестового проекта

Для того чтобы в новом проекте была возможность обращаться к классам тестируемого проекта, необходимо добавить на него ссылку (рис. 31, 32) и в файле теста добавить строку подключения зависимости «using BasicsOfProgramming», после этого можно приступать к непосредственному написанию автоматических тестов.

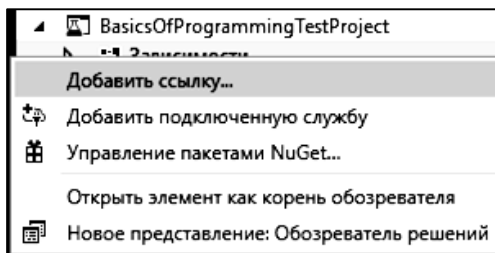


Рис. 31. Добавление ссылок в проект

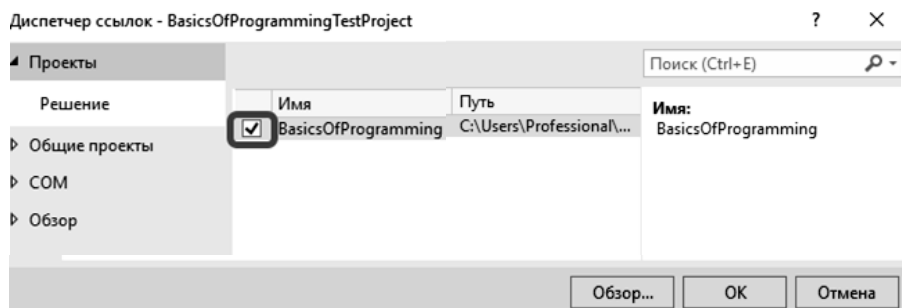


Рис. 32. Поиск проекта в диспетчере ссылок

Добавим несколько простых тестов на методы класса Calculator:

```
[Test]
public void TestCalculatorGetAdd()
{
    var calculator = new Calculator(4, 3);
    Assert.AreEqual(7, calculator.GetAdd());
}

[Test]
public void TestCalculatorGetDiv()
{
    var calculator = new Calculator(6, 4);
    Assert.AreEqual(1.5, calculator.GetDiv());
}
```

```
[Test]
public void TestCalculatorGetMult()
{
    var calculator = new Calculator(4, 3);
    Assert.AreEqual(12, calculator.GetMult());
}
```

```
[Test]
public void TestCalculatorGetSub()
{
    var calculator = new Calculator(4, 3);
    Assert.AreEqual(1, calculator.GetSub());
}
```

В приведенном коде, используется класс Assert, он содержит методы для написания различного рода тестов. В приведенном примере используется метод AreEqual, который принимает два операнда на входе и проверяет что переданные операнды одинаковы. Для того чтобы запустить написанные тесты в меню «Тест» выберите «Выполнить все тесты» (рис. 33).

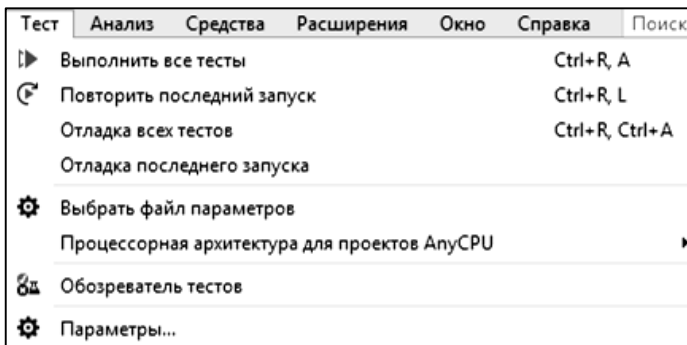


Рис. 33. Запуск тестов

После запуска тестов откроется оснастка «Обозреватель тестов» (рис. 34).

Как видно из рис. 34 один из тестов «провалился», в результате выполнения теста TestCalculatorGetDiv мы ожидали, поделив 6 на 4, получить значение 1,5, а в результате вызова метода calculator.GetDiv() было получено значение 1,0, следовательно необходимо провести отладку этого метода чтобы все тесты связанные с ним проходили. Представленная ошибка возникла из-за округления целочисленного операнда при делении, для решения этой проблемы достаточно сделать приведение типов к double:

```
public double GetDiv()
{
```

```

double firstDouble = Convert.ToDouble(firstNumber);
double secondDouble = Convert.ToDouble(secondNumber);
return firstDouble / secondDouble;
}

```

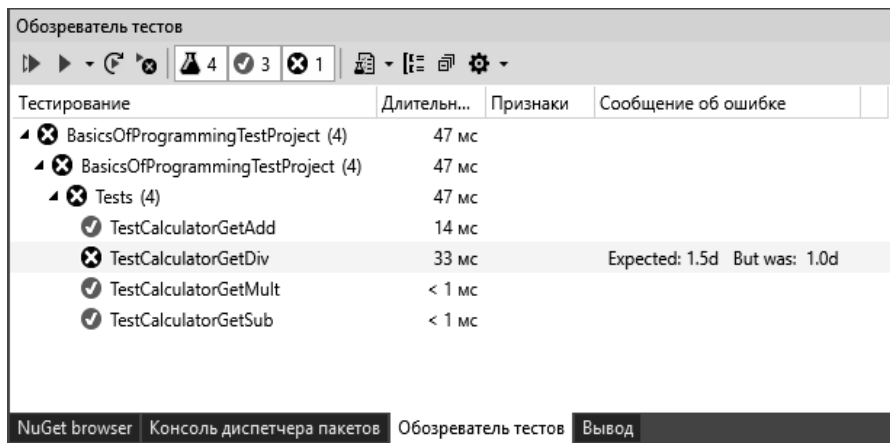


Рис. 34. Оснастка «Обозреватель тестов»

Аналогичным способом необходимо покрыть тестами все методы разработанных классов.

Основными целями ручного тестирования интерфейса являются выявления ошибок функциональности, необработанных исключений, несоответствия проектной документации. Для того чтобы провести ручное тестирование интерфейса необходимо составить тест планы в виде сценариев работы пользователя на естественном языке, пример тест плана на естественном языке представлен ниже:

1. Пользователь не заполняет поле «Логин».
2. Пользователь заполняет поле «Пароль».
3. Пользователь заполняет поле «Подтвердите пароль» аналогично полю «Пароль».
4. Пользователь нажимает кнопку «Зарегистрировать».
5. Пользователю отображается информационное окно о том, что необходимо заполнить поле «Логин».

После составления тест планов для покрытия всей функциональности разрабатываемого приложения необходимо по каждому тест плану провести тестирование с предоставлением результатов в виде скриншотов и заключения о его выполнении или невыполнении. После выявления провалов тест планов, необходимо произвести отладку и снова выполнить тестирование, до тех пор, пока все тест планы не будут выполнены.

## Глава 6

### ВАРИАНТЫ ЗАДАНИЙ НА КУРСОВУЮ РАБОТУ

#### 1. Шифр Гронсфельда

Этот шифр сложной замены, называемый шифром Гронсфельда, представляет собой модификацию шифра Цезаря числовым ключом. Для этого под буквами исходного сообщения записывают цифры числового ключа. Если ключ короче сообщения, то его запись циклически повторяют. Шифртекст получают примерно, как в шифре Цезаря, но отсчитывают по алфавиту не третью букву (как это делается в шифре Цезаря), а выбирают ту букву, которая смещена по алфавиту на соответствующую цифру ключа. Например, применяя в качестве ключа группу из четырех начальных цифр числа  $e$  (основания натуральных логарифмов), а именно 2718, получаем для исходного сообщения ВОСТОЧНЫЙ ЭКСПРЕСС шифртекст представленный в табл. 3.

Таблица 3

Пример использования шифра Гронсфельда

Сообщение	ВОСТОЧНЫЙЭКСПРЕСС
Ключ	27182718271827100
Шифртекст	ДХТЬРЮОГЛДЛЩСЧЖЩУ

Чтобы зашифровать первую букву сообщения В, используя первую цифру ключа 2, нужно отсчитать вторую по порядку букву от В в алфавите. Получается первая буква шифртекста Д.

#### 2. Шифрование квадратом Полибия

В Древней Греции (II в. до н.э.) был известен шифр, называемый «квадрат Полибия». Шифровальная таблица представляла собой квадрат с пятью столбцами и пятью строками, которые нумеровались цифрами от 1 до 5. В каждую клетку такого квадрата записывалась одна буква. В результате каждой букве соответствовала пара чисел, и шифрование сводилось к замене буквы парой чисел. Для латинского алфавита квадрат Полибия имеет вид, представленный в табл. 4.

Таблица 4

Квадрат Полибия для латинского алфавита

	1	2	3	4	5	6
1	A	B	C	D	E	F
2	G	H	I	J	K	L
3	M	N	O	P	Q	R
4	S	T	U	V	X	Y
5	Z	!	,	.	:	—
6	?	;	(	)	%	/

### 3. Шифр Хилла

Шифр Хилла является полиграммным шифром, который может использовать большие блоки с помощью линейной алгебры. Каждой букве алфавита сопоставляется число по модулю 26. Для латинского алфавита часто используется простейшая схема:  $A = 0, B = 1, \dots, Z = 25$ , но это не является существенным свойством шифра. Блок из  $n$  букв рассматривается как  $n$ -мерный вектор и умножается по модулю 26 на матрицу размера  $n \times n$ .

### 4. Шифр Атбаш

Атбаш – простой шифр подстановки для алфавитного письма. Правило шифрования состоит в замене  $i$ -й буквы алфавита буквой с номером  $n-i+1$ , где  $n$  – число букв в алфавите.

### 5. Шифр Виженера(для латинского алфавита)

Шифр Виженера – метод полиалфавитного шифрования буквенного текста с использованием ключевого слова. Шифр Виженера «размывает» характеристики частот появления символов в тексте, но некоторые особенности появления символов в тексте остаются. Главный недостаток шифра Виженера состоит в том, что его ключ повторяется. Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания необходимо использовать таблицу для латинского алфавита.

### 6. Шифр Виженера(для кириллицы)

Для зашифровывания необходимо использовать таблицу с кириллицей.

### 7. Шифр Плейфера

Шифр Плейфера использует матрицу  $5 \times 5$  (для латинского алфавита, для кириллического алфавита необходимо увеличить размер матрицы до  $4 \times 8$ ), содержащую ключевое слово или фразу. Для создания матрицы и использования шифра достаточно запомнить ключевое слово и четыре простых правила. Чтобы составить ключевую матрицу, в первую очередь нужно заполнить пустые ячейки матрицы буквами ключевого слова (не записывая повторяющиеся символы), потом заполнить оставшиеся ячейки матрицы символами алфавита, не встречающимися в ключевом слове, по порядку (в английских текстах обычно опускается символ «Q», чтобы уменьшить алфавит, в других версиях «I» и «J» объединяются в одну ячейку). Ключевое слово может быть записано в верхней строке матрицы слева направо, либо по спирали из левого верхнего угла к центру. Ключевое слово, дополненное алфавитом, составляет матрицу  $5 \times 5$  и является ключом шифра.

### 8. Шифр Скитала

Для шифрования сообщения использовались пергаментная лента и палочка цилиндрической формы с фиксированными длиной и диаметром. Пергаментная лента наматывалась на палочку так, чтобы не было ни просветов, ни нахлестов. Написание сообщения производилось по намотанной пергаментной ленте по длинной стороне цилиндра. После того, как достигался конец намотанной ленты, палочка поворачивалась на часть оборота и написание со-

общения продолжалось. После разматывания ленты на ней оказывалось зашифрованное сообщение. Расшифрование выполнялась с использованием палочки таких же типоразмеров.

### 9. Простая табличная перестановка

Можно записать сообщение в прямоугольную таблицу по маршруту: по горизонтали, начиная с верхнего левого угла, поочередно слева направо. Сообщение будем списывать по маршруту: по вертикалям, начиная с верхнего правого угла, поочередно сверху вниз. Полученная строка и есть зашифрованный текст.

### 10. Двойная табличная перестановка

Аналогично простой табличной перестановки для обеспечения дополнительной скрытности можно повторно зашифровать сообщение, которое уже прошло шифрование. Такой метод шифрования называется двойной перестановкой. В случае двойной перестановки столбцов и строк таблицы перестановки определяются отдельно для столбцов и отдельно для строк. Сначала в таблицу записывается текст сообщения, а потом поочередно переставляются столбцы, а затем строки. При расшифровании порядок перестановок должен быть обратным.

**Для вариантов 1–10** Разработать программу, которая зашифрует введенный с клавиатуры текст и сохранит его в файл и базу данных и при необходимости считает зашифрованный текст из файла или базы данных и расшифрует данный текст.

### 11. Магический квадрат

Магическим квадратом порядка  $N$  называется квадратная таблица размера  $N \times N$ , составленная из чисел  $1, 2, \dots, N^2$  так, что суммы по каждому столбцу, каждой строке и каждой из двух диагоналей равны между собой. Составить программу построит магический квадрат для заданного  $N$ , введенного с клавиатуры, сохранит его в виде изображения в файл.

### 12. Сложение и вычитание дробей с помощью метода «бабочка»

Метод бабочки заключается в умножении чисел по диагонали. Если дроби нужно сложить, получившиеся числа также нужно сложить, аналогично при вычитании. Это будет наш числитель. После умножаем числа в знаменателе – получаем ответ! Пример применения этого метода представлен на рис. 35.

Для введенных с клавиатуры дробей построить «бабочки» и вычислить результат, с сохранением в виде картинки в файл.

$$\frac{3}{4} + \frac{2}{5} \rightarrow \frac{3 \cdot 5 + 2 \cdot 4}{4 \cdot 5} = \frac{15 + 8}{20} = \frac{23}{20} = 1 \frac{3}{20}$$

$$\frac{3}{4} - \frac{2}{5} \rightarrow \frac{3 \cdot 5 - 2 \cdot 4}{4 \cdot 5} = \frac{15 - 8}{20} = \frac{7}{20}$$

Рис. 35. Пример использования метода «бабочка»

### **13. Решето Эратосфена**

Решето Эратосфена – алгоритм нахождения всех простых чисел до некоторого целого числа  $n$ , который приписывают древнегреческому математику Эратосфену Киренскому. Решето подразумевает фильтрацию, в данном случае фильтрацию всех чисел за исключением простых. По мере прохождения списка нужные числа остаются, а ненужные исключаются.

### **14. Решето Сундарамы**

Решето Сундарамы – детерминированный алгоритм нахождения всех простых чисел до некоторого целого числа  $n$ . Разработан индийским студентом Сундарамом в 1934 году.

### **15. Решето Аткина**

Решето Аткина – алгоритм нахождения всех простых чисел до заданного целого числа  $N$ . Алгоритм был создан А.О. Аткином и Д.Ю. Бернштайном. Заявленная авторами асимптотическая скорость работы алгоритма соответствует скорости лучших ранее известных алгоритмов просеивания, но в сравнении с ними решето Аткина требует меньше памяти.

Для вариантов 13–15 разработать программу, позволяющую пользователю с клавиатуры ввести натуральное число  $N$ , от 1 до 50 и получить  $N$  – первых натуральных чисел. Составленное решето должно быть сохранено в виде изображения в файл.

### **16. Алгоритм ближайшего соседа в задаче коммивояжера**

Коммивояжер желает посетить ряд городов и вернуться в исходный город, минимизируя суммарную длину поездок. Эта задача в математической форме формулируется как задача нахождения во взвешенном графе гамильтонова цикла минимальной длины и называется задачей коммивояжера. Алгоритм ближайшего соседа – один из простейших эвристических алгоритмов решения задачи коммивояжера. Относится к категории «жадных» алгоритмов. Формулируется следующим образом: пункты обхода плана последовательно включаются в маршрут, причем каждый очередной включаемый пункт должен быть ближайшим к последнему выбранному пункту среди всех остальных, еще не включенных в состав маршрута.

### **17. Метод ветвей и границ в задаче коммивояжера**

Для решения задачи коммивояжера методом ветвей и границ необходимо выполнить следующий алгоритм:

- Построение матрицы с исходными данными.
- Нахождение минимума по строкам.
- Редукция строк.
- Нахождение минимума по столбцам.
- Редукция столбцов.
- Вычисление оценок нулевых клеток.
- Редукция матрицы.



- Если полный путь еще не найден, переходим к пункту 2, если найден к пункту 9

- Вычисление итоговой длины пути и построение маршрута.

Для вариантов 16, 17 необходимо предоставить пользователю интерфейс для указания вершин графа, после чего задача коммивояжера должна быть решена и решение выведено на экран, результат необходимо сохранить в виде изображения в файл.

### **18. Алгоритм Блюм–Блюма–Шуба**

Основная рекуррентная формула алгоритма:  $x_n = x_{n-1}^2 \bmod pq$ , где  $p$  и  $q$  два больших простых числа. Два простых числа,  $p$  и  $q$ , должны быть оба сравнимы с 3 по модулю 4 ( $p \equiv 3 \pmod{4}$ ,  $q \equiv 3 \pmod{4}$ ) и НОД должен быть мал (это увеличивает длину цикла). Для повышения качества получаемой последовательности на очередном шаге выбираются не все биты, а только младшие, или даже только бит четности. Из полученных «случайных битов» формируются двоичные псевдослучайные числа произвольной разрядности.

### **19. Запаздывающие генераторы Фибоначчи**

В отличие от генераторов, использующих линейный конгруэнтный алгоритм, фибоначиевы генераторы можно использовать в статистических алгоритмах, требующих высокого разрешения. В связи с этим линейный конгруэнтный алгоритм постепенно потерял свою популярность и его место заняло семейство фибоначиевых алгоритмов, которые могут быть рекомендованы для использования в алгоритмах, критичных к качеству случайных чисел.

Для вариантов 18, 19 разработать программу, которая позволяет для введенных пользователем исходных данных сгенерировать последовательность случайных чисел, отобразить их на графике и сохранить в виде изображения в файл.

### **20. Алгоритм Кнута–Морриса–Пратта**

Алгоритм Кнута–Морриса–Пратта – эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объема входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Разработать программу, которая в заранее определенном тексте найдет и подсветит слово, введенное пользователем с клавиатуры, в случае отсутствия такого слова – выведет об этом сообщение. Предусмотреть возможность изменения изначального текста.

### **21. Расстояние Левенштейна**

Расстояние Левенштейна – минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Измеряется для двух строк, широко используется в теории информации и компьютерной лингвистике.

## **22. Расстояние Дамерау–Левенштейна**

Расстояние Дамерау–Левенштейна (названо в честь ученых Фредерика Дамерау и Владимира Левенштейна) – это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определенных в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

Для вариантов 21, 22 для двух введенных пользователем строк рассчитать расстояние Левенштейна/Дамерау–Левенштейна. Результаты вычислений сохранить в базу данных и предусмотреть возможность их просмотра.

## **23. Алгоритм волновой трассировки**

Алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е. при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещен, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно – указание пары ячеек, между которыми нужно найти кратчайший путь. Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно.

Разработать программу, которая позволяет пользователю заполнить поле непроходимыми ячейками, указать начальное и конечное поле, после этого программа должна найти кратчайший путь, отрисовать его на поле, и сохранить результат в виде изображения в файл, в случае отсутствия пути – вывести об этом сообщение.

## **24. Двоичный поиск**

Двоичный (бинарный) поиск – классический алгоритм поиска элемента в отсортированном массиве, использующий дробление массива на половины. Используется в информатике, вычислительной математике и математическом программировании. Широкое распространение в информатике применительно к поиску в структурах данных. Например, поиск в массивах данных осуществляется по ключу, присвоенному каждому из элементов массива. Также его применяют в качестве численного метода для нахождения приближенного решения уравнений.

Разработать программу телефонный справочник, которая используя двоичный поиск, ищет абонента с введенными пользователем фамилией и именем. Данные всех абонентов должны храниться в базе данных и при выборке сортироваться по алфавиту.

## **25. Пирамидальная сортировка**

Пирамидальная сортировка – алгоритм сортировки, работающий в худшем, в среднем и в лучшем случае за  $\Theta(n \log n)$  операций при сортировке  $n$  элементов. Количество применяемой служебной памяти не зависит от размера массива (т.е.,  $O(1)$ ).

## **26. Сортировка расческой**

Сортировка расческой улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея – устранить черепахи, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком.

## **27. Гномья сортировка**

Гномья сортировка основана на технике, используемой обычным голландским садовым гномом. Это метод, которым садовый гном сортирует линию цветочных горшков. По существу он смотрит на текущий и предыдущий садовые горшки: если они в правильном порядке, он шагает на один горшок вперед, иначе он меняет их местами и шагает на один горшок назад. Граничные условия: если нет предыдущего горшка, он шагает вперед; если нет следующего горшка, он закончил.

**Для вариантов 25–27** предоставить возможность пользователю задать размер массива, после чего заполнить массив случайными числами (использовать готовую библиотеку) и отсортировать алгоритмом в соответствии вариантом. Процесс сортировки должен сопровождаться изменением гистограммы отображения значений массива на форме приложения

## **28. Метод Куайна**

Метод Куайна – способ представления функции в ДНФ или КНФ с минимальным количеством членов и минимальным набором переменных. Преобразование функции можно разделить на два этапа: на первом этапе осуществляется переход от канонической формы к так называемой сокращенной форме; на втором этапе – переход от сокращенной формы к минимальной форме.

## **29. Метод Куайна–Мак-Класки**

Метод Куайна–Мак-Класки – табличный метод минимизации булевых функций, предложенный Уиллардом Куайном и усовершенствованный Эдвардом Мак-Класки. Представляет собой попытку избавиться от недостатков метода Куайна. Специфика метода Куайна–Мак-Класки по сравнению с методом Куайна в сокращении количества попарных сравнений на предмет их склеивания. Сокращение достигается за счет исходного разбиения термов на группы с равным количеством единиц (нулей). Это позволяет исключить сравнения, заведомо не дающие склеивания. Несмотря на большую возможность практиче-

ского применения чем у карт Карно когда речь идет о более чем четыре переменных, метод Куайна–Мак-Класки тоже имеет ограничения области применения, так как время работы метода растет экспоненциально с увеличением входных данных.

Для вариантов 28-29 для введенного с клавиатуры логического выражения провести минимизацию функции. Все логические функции необходимо сохранять в базе данных, в случае уже имеющийся в базе данных изначально функции результат выводить из базы данных.

### **30. Карты Карно**

Карта Карно – графический способ представления переключательных булевых функций с целью наглядной и удобной их минимизации, обеспечивающий упрощение сложных логических функций многих переменных и устранение потенциальных логических гонок. Является одним из эквивалентных способов описания или задания логических функций наряду с таблицей истинности или выражениями булевой алгебры. Преобразование представления логической функции, заданной в виде карты Карно в таблицу истинности или в булеву формулу и обратное преобразование элементарно по простоте.

Составить программу, которая для введенного логического выражения построить карту Карно, отобразит ее на форме и предложит решение ввести пользователю. Булеву функцию до минимизации и после минимизации сохранять в базе данных, карту Карно сохранять в виде изображения в файле.

## ЛИТЕРАТУРА

1. Кнут Д.Э. Искусство программирования. – Т. 1: Основные алгоритмы. – 3-е изд.; пер. с англ. – М.: ИД «Вильямс», 2016. – 720 с.
2. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: учеб. пособие для вузов. – СПб.: Питер, 2002. – 464 с.
3. Шустова Л.И., Тараканов О.В. Базы данных: учебник. – М.: НИЦ ИНФРА-М, 2016. – 304 с.
4. Старолетов С.М. Основы тестирования и верификации программного обеспечения: учеб. пособие. – СПб.: Лань, 2018. – 344 с.
5. Бек К. Экстремальное программирование: разработка через тестирование: пер. с англ. П. Анджан. – СПб.: Питер, 2003. – 224 с.

## ПРИЛОЖЕНИЕ А

### Форма задания на курсовую работу

Министерство науки и высшего образования РФ  
ФГБОУ ВО «Гомский государственный университет  
систем управления и радиоэлектроники»  
Кафедра безопасности информационных систем (БИС)

УТВЕРЖДАЮ

Зав. кафедрой БИС

\_\_\_\_\_ Е.Ю. Костюченко

«\_\_» \_\_\_\_\_ 2019 г.

### Задание

на курсовую работу по дисциплине «Основы программирования» студенту группы 728-1 факультета безопасности А.А. Иванову.

Тема работы: «Решето Аткина».

Цель работы: Получение навыков программирования на языках высоко уровня и применения парадигмы объектно-ориентированного программирования на практике.

Срок сдачи студентом законченной работы «\_\_» декабря 2019 г.

Исходные данные к работе: решето Аткина – алгоритм нахождения всех простых чисел до заданного целого числа  $N$ . Алгоритм был создан А.О. Аткином и Д.Ю. Бернштайном. Заявленная авторами асимптотическая скорость работы алгоритма соответствует скорости лучших ранее известных алгоритмов просеивания, но в сравнении с ними решето Аткина требует меньше памяти.

Задание: разработать программу позволяющую пользователю с клавиатуры ввести натуральное число  $N$ , от 1 до 50 и получить  $N$  – первых натуральных чисел при помощи решета Аткина. Составленное решето должно быть сохранено в виде изображения в файл.

Требования к используемым технологиям: язык программирования C#, .Net Framework, СУБД SQLite.

СОГЛАСОВАНО

Доц. кафедры БИС

\_\_\_\_\_ С.С. Харченко

«\_\_» \_\_\_\_\_ 2019 г.

ПРИНЯЛ К ИСПОЛНЕНИЮ

Студент гр. 728-1

\_\_\_\_\_ А.А. Иванов

«\_\_» \_\_\_\_\_ 2019 г.

Для заметок

Учебное издание

*Сергей Сергеевич Харченко*

## **ОСНОВЫ ПРОГРАММИРОВАНИЯ**

*Учебно-методическое пособие по курсовой работе*

*для студентов специальностей и направлений*

*10.03.01 – «Информационная безопасность»,*

*10.05.02 – «Информационная безопасность  
телекоммуникационных систем»,*

*10.05.03 – «Информационная безопасность  
автоматизированных систем»,*

*10.05.04 – «Информационно-аналитические системы безопасности»*

Верстка – В.М. Бочкаревой

Текст дан в авторской редакции, без корректуры

---

Издательство «В-Спектр»

Подписано к печати 05.12.2019

Формат 60×84<sup>1/16</sup>. Печать трафаретная.

Печ. л. 3. Тираж 100 экз. Заказ 31.

---

Тираж отпечатан ИП В.М. Бочкаревой

ИНН 701701817754

634055, г. Томск, пр. Академический 13-24

Т. 8-905-089-92-40. Эл. почта: [bvm@sibmail.com](mailto:bvm@sibmail.com)