Московский государственный технический университет
имени Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

доцент И.С. Папшев
доцент М.В. Черненький
доцент Ю.Е. Гапанюк

Методические указания
к лабораторным работам по курсу
«Базовые компоненты интернет-технологий»
(3 семестр)

Москва
2014

СОДЕРЖАНИЕ

1 Введение

Дисциплина «Базовые компоненты интернет-технологий» предназначена для подготовки студентов к изучению дисциплины «Разработка интернет - приложений». Так как дисциплина «Разработка интернет - приложений» основывается на использовании языка программирования «C#», то основной задачей дисциплины «Базовые компоненты интернет-технологий» является изучение языка программирования «C#».

Лабораторный практикум по курсу «базовые компоненты интернет-технологий» предназначен для формирования у студентов компетенций, связанных с программированием на языке «C#».

Лабораторный практикум содержит 6 лабораторных работ:

1. **Разработка программы для решения квадратного уравнения (2 часа).**

Лабораторная работа является вводной. Она помогает студентам освоить основные управляющие конструкции языка C#, а также основы консольного ввода-вывода в C#.

2. **Разработка программы, реализующей работу с классами (2 часа).**

Лабораторная работа предназначена для практического освоения работы с классами и интерфейсами, наследования классов, реализации абстрактных классов и виртуальных функций.

3. **Разработка программы, реализующей работу с коллекциями (4 часа).**

Лабораторная работа предназначена для практического освоения работы со стандартными коллекциями в обобщенном и необобщенном вариантах, создания собственных коллекций.

4. **Разработка программы, реализующей работу с файлами и технологией Windows Forms (2 часа).**

Лабораторная работа предназначена для практического освоения работы с файлами в C#, а также для оконных приложений с использованием технологии Windows Forms.

5. **Разработка программы, реализующей вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера (4 часа).**

Лабораторная работа базируется на предыдущей работе и предназначена для практического освоения алгоритмов нечеткого сравнения строк.

6. **Разработка программы для работы с делегатами и рефлексией (3 часа).**

Лабораторная работа предназначена для практического освоения работы с делегатами, лямбда-выражениями, рефлексией в языке программирования C#.

2 Цель лабораторного практикума

Целью лабораторного практикума является содействие в формировании следующих компетенций:

- способен разрабатывать и отлаживать компоненты аппаратно-программных комплексов с помощью современных автоматизированных средств проектирования (ПК-7);
- умеет разрабатывать интерфейсы «человек - ЭВМ» (ПК-12);

В результате выполнения лабораторного практикума студент должен уметь:

- разрабатывать консольные и оконные приложения с использованием языка программирования «C#»;
- разрабатывать интерфейсы «человек - ЭВМ» с использованием технологии Windows Forms.

3 Краткая характеристика объекта изучения, исследования

Объектом изучения лабораторного практикума является язык программирования C#.

В частности, в рамках лабораторного практикума изучаются такие аспекты языка C#, как:

- основы объектно-ориентированного программирования на C#, работа с классами и интерфейсами;
- работа со стандартными коллекциями C#, основы разработки собственных коллекционных классов;

- работа с файлами;
- разработка оконных приложений с использованием Windows Forms;
- работа с делегатами и лямбда-выражениями;
- работа с рефлексией.

Язык программирования C# является современным языком программирования, полное описание которого в силу большого объема не может быть представлено в рамках настоящих методических указаний.

Фрагменты программ, которые могут быть использованы для самостоятельного анализа студентами и применены в лабораторных работах, представлены в разделе «Вспомогательные материалы для выполнения лабораторных работ».

Для изучения языка и выполнения лабораторных работ можно рекомендовать источники [1-3].

4 Схема и описание лабораторной установки

В качестве лабораторной установки используется компьютер со следующим программным обеспечением:

- операционная система Windows 7 и выше;
- среда разработки Visual Studio 2010 и выше.

Все программное обеспечение является лицензионным и предоставляется компанией Microsoft в рамках академической программы сотрудничества с МГТУ им. Н.Э. Баумана.

5 Содержание отчета по лабораторным работам

Отчеты разрабатываются отдельно по каждой лабораторной работе. Отчет по каждой лабораторной работе должен включать:

- титульный лист;
- описание задания лабораторной работы;
- тексты программ на языке C#;
- диаграмму классов;

- результаты выполнения программы, экранные формы.

6 Задачи и порядок выполнения работ

6.1 Лабораторная работа 1

Разработать программу для решения квадратного уравнения.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Программа осуществляет ввод с клавиатуры коэффициентов A, B, C, вычисляет дискриминант и корни уравнения (в зависимости от дискриминанта).
3. Если коэффициент A, B, C введен некорректно, то необходимо проигнорировать некорректное значение и ввести коэффициент повторно.

6.2 Лабораторная работа 2

Разработать программу, реализующую работу с классами.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Абстрактный класс «Геометрическая фигура» содержит виртуальный метод для вычисления площади фигуры.
3. Класс «Прямоугольник» наследуется от класса «Геометрическая фигура». Ширина и высота объявляются как свойства (property). Класс должен содержать конструктор по параметрам «ширина» и «высота».
4. Класс «Квадрат» наследуется от класса «Прямоугольник». Класс должен содержать конструктор по длине стороны.
5. Класс «Круг» наследуется от класса «Геометрическая фигура». Радиус объявляется как свойство (property). Класс должен содержать конструктор по параметру «радиус».

6. Для классов «Прямоугольник», «Квадрат», «Круг» переопределить виртуальный метод Object.ToString(), который возвращает в виде строки основные параметры фигуры и ее площадь.
7. Разработать интерфейс IPrint. Интерфейс содержит метод Print(), который не принимает параметров и возвращает void. Для классов «Прямоугольник», «Квадрат», «Круг» реализовать наследование от интерфейса IPrint. Переопределяемый метод Print() выводит на консоль информацию, возвращаемую переопределенным методом ToString().

6.3 Лабораторная работа 3

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
6. Модифицировать класс разреженной матрицы Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (представлен в разделе

«Вспомогательные материалы для выполнения лабораторных работ»). Необходимо добавить в класс методы:

- public void Push(T element) – добавление в стек;
- public T Pop() – чтение с удалением из стека.

8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

6.4 Лабораторная работа 4

Разработать программу, реализующую работу с файлами.

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF (Windows Presentation Foundation).
2. Добавить кнопку, реализующую функцию чтения текстового файла в список слов List<string>.
3. Для выбора имени файла используется класс OpenFileDialog, который открывает диалоговое окно с выбором файла. Ограничить выбор только файлами с расширением «.txt».
4. Для чтения из файла рекомендуется использовать статический метод ReadAllText() класса File (пространство имен System.IO). Содержимое файла считывается методом ReadAllText() в виде одной строки, далее делится на слова с использованием метода Split() класса string. Слова сохраняются в список List<string>.
5. При сохранении слов в список List<string> дубликаты слов не записываются. Для проверки наличия слова в списке используется метод Contains().
6. Вычислить время загрузки и сохранения в список с использованием класса Stopwatch (пространство имен System.Diagnostics). Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).
7. Добавить на форму поле ввода для поиска слова и кнопку поиска. При нажатии на кнопку поиска осуществлять поиск введенного слова в списке.

Слово считается найденным, если оно входит в элемент списка как подстрока (метод Contains() класса string).

8. Добавить на форму список (ListBox). Найденные слова выводить в список с использованием метода «название_списка.Items.Add()». Вызовы метода «название_списка.Items.Add()» должны находится между вызовами методов «название_списка.BeginUpdate()» и «название_списка. EndUpdate()».
9. Вычислить время поиска с использованием класса Stopwatch. Вычисленное время вывести на форму в поле ввода (TextBox) или надпись (Label).

6.5 Лабораторная работа 5

Разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера.

1. Программа должна быть разработана в виде библиотеки классов на языке C#.
2. Использовать самый простой вариант алгоритма без оптимизации.
3. Дополнительно возможно реализовать вычисление расстояния Дамерау-Левенштейна (с учетом перестановок соседних символов).
4. Модифицировать предыдущую лабораторную работу, вместо поиска подстроки используется вычисление расстояния Левенштейна.
5. Предусмотреть отдельное поле ввода для максимального расстояния. Если расстояние Левенштейна между двумя строками больше максимального, то строки считаются несовпадающими и не выводятся в список результатов.

6.6 Лабораторная работа 6

Часть 1. Разработать программу, использующую делегаты.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.

4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
    - метод, разработанный в пункте 3;
    - лямбда-выражение.
5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

Часть 2. Разработать программу, реализующую работу с рефлексией.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
6. Вызовите один из методов класса с использованием рефлексии.

7 Вспомогательные материалы для выполнения лабораторных работ

7.1 Лабораторная работа 1

7.1.1 Фрагмент программы, реализующий консольный ввод-вывод

```
class Program
    {
        static void Main(string[] args)
        {
            float f;

            Console.Write("Введите число: ");
            string str = Console.ReadLine();

            //Преобразование строки в число 1
```

```
            Console.WriteLine("Преобразование строки в число 1");
            bool ConvertResult = float.TryParse(str, out f);
            if (ConvertResult)
            {
                Console.WriteLine("Вы ввели число " + f.ToString("F5"));
            }
            else
            {
                Console.WriteLine("Вы ввели не число");
            }

            //Преобразование строки в число 2
            Console.WriteLine("Преобразование строки в число 2");
            try
            {
                f = float.Parse(str);
                Console.WriteLine("Вы ввели число " + f.ToString("F5"));
            }
            catch (Exception e)
            {
                Console.WriteLine("Вы ввели не число: " + e.Message);
                Console.WriteLine("\nПодробное описание ошибки: " + e.StackTrace);
            }

            //Вывод параметров командной строки
            CommandLineArgs(args);


Console.WriteLine("++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
            double a = ReadDouble("Введите коэффициент A: ");
            Console.WriteLine("Вы ввели коэффициент A = " + a);

            Console.ReadLine();
        }

        /// <summary>
        /// Вывод параметров командной строки
        /// </summary>
        /// <param name="args"></param>
        static void CommandLineArgs(string[] args)
        {

            /*
             Необходимо установить параметры командной строки (несколько слов
через пробел)
             в пункте меню "Project", "название проекта Properties"
             вкладка "Debug", поле ввода "Command Line Arguments"
             */

            //Вывод параметров командной строки 1
            Console.WriteLine("\nВывод параметров командной строки 1:");
            for (int i = 0; i < args.Length; i++)
            {
                Console.WriteLine("Параметр [{0}] = {1}", i, args[i]);
            }
```

```
            //Вывод параметров командной строки 2
            Console.WriteLine("\nВывод параметров командной строки 2:");
            int i2 = 0;
            foreach (string param in args)
            {
                Console.WriteLine("Параметр [{0:F5}] = {1}", i2, param);
                i2++;
            }
        }

        /// <summary>
        /// Ввод вещественного числа с проверкой корректности ввода
        /// </summary>
        /// <param name="message">Подсказка при вводе</param>
        /// <returns></returns>
        static double ReadDouble(string message)
        {
            string resultString;
            double resultDouble;
            bool flag;

            do
            {
                Console.Write(message);
                resultString = Console.ReadLine();

                //Первый способ преобразования строки в число
                flag = double.TryParse(resultString, out resultDouble);

                //Второй способ преобразования строки в число
                /*
                try
                {
                    resultDouble = double.Parse(resultString);
                    //resultDouble = Convert.ToDouble(resultString);
                    flag = true;
                }
                catch
                {
                    //Необходимо присвоить значение по умолчанию из-за ошибки
компилятора
                    resultDouble = 0;
                    flag = false;
                }
                */

                if (!flag)
                {
                    Console.WriteLine("Необходимо ввести вещественное число");
                }
            }
            while (!flag);

            return resultDouble;
        }
```

```
    }
```

7.2 Лабораторная работа 2

7.2.1 Абстрактный класс «Геометрическая фигура»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    /// <summary>
    /// Класс фигура
    /// </summary>
    abstract class Figure : IComparable
    {
        /// <summary>
        /// Тип фигуры
        /// </summary>
        public string Type
        {
            get
            {
                return this._Type;
            }
            protected set
            {
                this._Type = value;
            }
        }
        string _Type;

        /// <summary>
        /// Вычисление площади
        /// </summary>
        /// <returns></returns>
        public abstract double Area();

        /// <summary>
        /// Приведение к строке, переопределение метода Object
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            return this.Type + " площадью " + this.Area().ToString();
        }

        /// <summary>
        /// Сравнение элементов (для сортировки списка)
        /// </summary>
        /// <param name="obj"></param>
        /// <returns></returns>
        public int CompareTo(object obj)
```

```
        {
            Figure p = (Figure)obj;

            if (this.Area() < p.Area()) return -1;
            else if (this.Area() == p.Area()) return 0;
            else return 1; //(this.Area() > p.Area())
        }
    }
}
```

7.2.2 Интерфейс IPrint

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    interface IPrint
    {
        void Print();
    }
}
```

7.2.3 Класс «Прямоугольник»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    class Rectangle : Figure, IPrint
    {
        /// <summary>
        /// Высота
        /// </summary>
        double height;

        /// <summary>
        /// Ширина
        /// </summary>
        double width;

        /// <summary>
        /// Основной конструктор
        /// </summary>
        /// <param name="ph">Высота</param>
        /// <param name="pw">Ширина</param>
        public Rectangle(double ph, double pw)
        {
            this.height = ph;
            this.width = pw;
```

```
            this.Type = "Прямоугольник";
        }

        /// <summary>
        /// Вычисление площади
        /// </summary>
        public override double Area()
        {
            double Result = this.width * this.height;
            return Result;
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

7.2.4 Класс «Квадрат»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    class Square : Rectangle, IPrint
    {
        public Square(double size)
            : base(size, size)
        {
            this.Type = "Квадрат";
        }
    }
}
```

7.2.5 Класс «Круг»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    class Circle : Figure, IPrint
    {
        /// <summary>
        /// Ширина
        /// </summary>
        double radius;

        /// <summary>
```

```
        /// Основной конструктор
        /// </summary>
        /// <param name="ph">Высота</param>
        /// <param name="pw">Ширина</param>
        public Circle(double pr)
        {
            this.radius = pr;
            this.Type = "Круг";
        }

        public override double Area()
        {
            double Result = Math.PI * this.radius * this.radius;
            return Result;
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
    }
}
```

7.3 Лабораторная работа 3

7.3.1 Фрагмент основной программы

```
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(5, 4);
            Square square = new Square(5);
            Circle circle = new Circle(5);

            Console.WriteLine("\nArrayList");
            ArrayList al = new ArrayList();
            al.Add(circle);
            al.Add(rect);
            al.Add(square);

            foreach (var x in al) Console.WriteLine(x);

            Console.WriteLine("\nArrayList - сортировка");
            al.Sort();
            foreach (var x in al) Console.WriteLine(x);

            Console.WriteLine("\nList<Figure>");
            List<Figure> fl = new List<Figure>();
            fl.Add(circle);
            fl.Add(rect);
            fl.Add(square);

            foreach (var x in fl) Console.WriteLine(x);

            Console.WriteLine("\nList<Figure> - сортировка");
            fl.Sort();
```

```
            foreach (var x in fl) Console.WriteLine(x);

            Console.WriteLine("\nМатрица");
            Matrix3D<Figure> cube = new Matrix3D<Figure>(3, 3, 3, null);
            cube[0, 0, 0] = rect;
            cube[1, 1, 1] = square;
            cube[2, 2, 2] = circle;
            Console.WriteLine(cube.ToString());

            Console.WriteLine("\nСписок");
            SimpleList<Figure> list = new SimpleList<Figure>();
            list.Add(square);
            list.Add(rect);
            list.Add(circle);

            foreach (var x in list) Console.WriteLine(x);

            list.Sort();
            Console.WriteLine("\nСортировка списка");
            foreach (var x in list) Console.WriteLine(x);

            Console.WriteLine("\nСтек");
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(rect);
            stack.Push(square);
            stack.Push(circle);

            while (stack.Count > 0)
            {
                Figure f = stack.Pop();
                Console.WriteLine(f);
            }

            Console.ReadLine();
        }
```

7.3.2 Класс «разреженная матрица» (Matrix)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SparseMatrix
{
    public class Matrix<T>
    {
        /// <summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// <summary>
        /// Количество элементов по горизонтали (максимальное количество столбцов)
        /// </summary>
        int maxX;
```

```
        /// <summary>
        /// Количество элементов по вертикали (максимальное количество строк)
        /// </summary>
        int maxY;

        /// <summary>
        /// Пустой элемент, который возвращается если элемент с нужными
координатами не был задан
        /// </summary>
        T nullElement;

        /// <summary>
        /// Конструктор
        /// </summary>
        public Matrix(int px, int py, T nullElementParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.nullElement = nullElementParam;
        }

        /// <summary>
        /// Индексатор для доступа к данных
        /// </summary>
        public T this[int x, int y]
        {
            get
            {
                CheckBounds(x, y);
                string key = DictKey(x, y);
                if (this._matrix.ContainsKey(key))
                {
                    return this._matrix[key];
                }
                else
                {
                    return this.nullElement;
                }
            }
            set
            {
                CheckBounds(x, y);
                string key = DictKey(x, y);
                this._matrix.Add(key, value);
            }
        }

        /// <summary>
        /// Проверка границ
        /// </summary>
        void CheckBounds(int x, int y)
        {
            if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит
за границы");
```

```
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит
за границы");
        }

        /// <summary>
        /// Формирование ключа
        /// </summary>
        string DictKey(int x, int y)
        {
            return x.ToString() + "_" + y.ToString();
        }

        /// <summary>
        /// Приведение к строке
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            //Класс StringBuilder используется для построения длинных строк
            //Это увеличивает производительность по сравнению с созданием и
склеиванием
            //большого количества обычных строк

            StringBuilder b = new StringBuilder();

            for (int j = 0; j < this.maxY; j++)
            {
                b.Append("[");
                for (int i = 0; i < this.maxX; i++)
                {
                    if (i > 0) b.Append("\t");
                    b.Append(this[i, j].ToString());
                }
                b.Append("]\n");
            }

            return b.ToString();
        }

    }
}
```

7.3.3 Класс «простой односвязный список» (SimpleList)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FigureCollections
{
    /// <summary>
    /// Элемент списка
    /// </summary>
    public class SimpleListItem<T>
    {
```

```
    /// <summary>
    /// Данные
    /// </summary>
    public T data { get; set; }
    /// <summary>
    /// Следующий элемент
    /// </summary>
    public SimpleListItem<T> next { get; set; }

    ///конструктор
    public SimpleListItem(T param)
    {
        this.data = param;
    }
}

/// <summary>
/// Список
/// </summary>
public class SimpleList<T> : IEnumerable<T>
    where T : IComparable
{
    /// <summary>
    /// Первый элемент списка
    /// </summary>
    protected SimpleListItem<T> first = null;

    /// <summary>
    /// Последний элемент списка
    /// </summary>
    protected SimpleListItem<T> last = null;

    /// <summary>
    /// Количество элементов
    /// </summary>
    public int Count
    {
        get { return _count; }
        protected set { _count = value; }
    }
    int _count;

    /// <summary>
    /// Добавление элемента
    /// </summary>
    /// <param name="element"></param>
    public void Add(T element)
    {
        SimpleListItem<T> newItem = new SimpleListItem<T>(element);
        this.Count++;

        //Добавление первого элемента
        if (last == null)
        {
            this.first = newItem;
            this.last = newItem;
```

```
        }
        //Добавление следующих элементов
        else
        {
            //Присоединение элемента к цепочке
            this.last.next = newItem;
            //Просоединенный элемент считается последним
            this.last = newItem;
        }
    }

    /// <summary>
    /// Чтение контейнера с заданным номером
    /// </summary>
    public SimpleListItem<T> GetItem(int number)
    {
        if ((number < 0) || (number >= this.Count))
        {
            //Можно создать собственный класс исключения
            throw new Exception("Выход за границу индекса");
        }

        SimpleListItem<T> current = this.first;
        int i = 0;

        //Пропускаем нужное количество элементов
        while (i < number)
        {
            //Переход к следующему элементу
            current = current.next;
            //Увеличение счетчика
            i++;
        }

        return current;
    }

    /// <summary>
    /// Чтение элемента с заданным номером
    /// </summary>
    public T Get(int number)
    {
        return GetItem(number).data;
    }


    /// <summary>
    /// Для перебора коллекции
    /// </summary>
    public IEnumerator<T> GetEnumerator()
    {
        SimpleListItem<T> current = this.first;

        //Перебор элементов
        while (current != null)
        {
```

```
                //Возврат текущего значения
                yield return current.data;
                //Переход к следующему элементу
                current = current.next;
            }
        }

        //Реализация обощенного IEnumerator<T> требует реализации необобщенного
интерфейса
        //Данный метод добавляется автоматически при реализации интерфейса
        System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }

        /// <summary>
        /// Сортировка
        /// </summary>
        public void Sort()
        {
            Sort(0, this.Count - 1);
        }

        /// <summary>
        /// Реализация алгоритма быстрой сортировки
        /// </summary>
        /// <param name="low"></param>
        /// <param name="high"></param>
        private void Sort(int low, int high)
        {
            int i = low;
            int j = high;
            T x = Get((low + high) / 2);
            do
            {
                while (Get(i).CompareTo(x) < 0) ++i;
                while (Get(j).CompareTo(x) > 0) --j;
                if (i <= j)
                {
                    Swap(i, j);
                    i++; j--;
                }
            } while (i <= j);

            if (low < j) Sort(low, j);
            if (i < high) Sort(i, high);
        }

        /// <summary>
        /// Вспомогательный метод для обмена элементов при сортировке
        /// </summary>
        private void Swap(int i, int j)
        {
            SimpleListItem<T> ci = GetItem(i);
            SimpleListItem<T> cj = GetItem(j);
```

```
            T temp = ci.data;
            ci.data = cj.data;
            cj.data = temp;
        }
    }
}
```

7.4 Лабораторная работа 4

7.4.1 Фрагмент кода формы

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using EditDistanceProject;

namespace WindowsFormsFiles
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Список слов
        /// </summary>
        List<string> list = new List<string>();

        private void buttonLoadFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog fd = new OpenFileDialog();
            fd.Filter = "текстовые файлы|*.txt";

            if (fd.ShowDialog() == DialogResult.OK)
            {
                Stopwatch t = new Stopwatch();
                t.Start();

                //Чтение файла в виде строки
                string text = File.ReadAllText(fd.FileName);

                //Разделительные символы для чтения из файла
                char[] separators = new char[] {'
','.',',','!','?','/','\t','\n'};

                string[] textArray = text.Split(separators);
```

```
                foreach (string strTemp in textArray)
                {
                    //Удаление пробелов в начале и конце строки
                    string str = strTemp.Trim();
                    //Добавление строки в список, если строка не содержится в
списке
                    if (!list.Contains(str)) list.Add(str);
                }

                t.Stop();
                this.textBoxFileReadTime.Text = t.Elapsed.ToString();
                this.textBoxFileReadCount.Text = list.Count.ToString();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл");
            }



        }

        private void buttonExact_Click(object sender, EventArgs e)
        {
            //Слово для поиска
            string word = this.textBoxFind.Text.Trim();

            //Если слово для поиска не пусто
            if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)
            {
                //Слово для поиска в верхнем регистре
                string wordUpper = word.ToUpper();

                //Временные результаты поиска
                List<string> tempList = new List<string>();

                Stopwatch t = new Stopwatch();
                t.Start();

                foreach (string str in list)
                {
                    if (str.ToUpper().Contains(wordUpper))
                    {
                        tempList.Add(str);
                    }
                }

                t.Stop();
                this.textBoxExactTime.Text = t.Elapsed.ToString();

                this.listBoxResult.BeginUpdate();

                //Очистка списка
                this.listBoxResult.Items.Clear();
```

```
                //Вывод результатов поиска
                foreach (string str in tempList)
                {
                    this.listBoxResult.Items.Add(str);
                }
                this.listBoxResult.EndUpdate();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для
поиска");
            }
        }

        private void buttonApprox_Click(object sender, EventArgs e)
        {
            //Слово для поиска
            string word = this.textBoxFind.Text.Trim();

            //Если слово для поиска не пусто
            if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)
            {
                int maxDist;
                if(!int.TryParse(this.textBoxMaxDist.Text.Trim(), out maxDist))
                {
                    MessageBox.Show("Необходимо указать максимальное расстояние");
                    return;
                }

                if (maxDist < 1 || maxDist > 5)
                {
                    MessageBox.Show("Максимальное расстояние должно быть в
диапазоне от 1 до 5");
                    return;
                }

                //Слово для поиска в верхнем регистре
                string wordUpper = word.ToUpper();

                //Временные результаты поиска
                List<Tuple<string, int>> tempList = new List<Tuple<string,
int>>();

                Stopwatch t = new Stopwatch();
                t.Start();

                foreach (string str in list)
                {
                    //Вычисление расстояния Дамерау-Левенштейна
                    int dist = EditDistance.Distance(str.ToUpper(), wordUpper);

                    //Если расстояние меньше порогового, то слово добавляется в
результат
                    if (dist <= maxDist)
                    {
                        tempList.Add(new Tuple<string, int>(str, dist));
```

```
                    }
                }

                t.Stop();
                this.textBoxApproxTime.Text = t.Elapsed.ToString();

                this.listBoxResult.BeginUpdate();

                //Очистка списка
                this.listBoxResult.Items.Clear();

                //Вывод результатов поиска
                foreach (var x in tempList)
                {
                    string temp = x.Item1 + "(расстояние=" + x.Item2.ToString() +
")";
                    this.listBoxResult.Items.Add(temp);
                }
                this.listBoxResult.EndUpdate();
            }
            else
            {
                MessageBox.Show("Необходимо выбрать файл и ввести слово для
поиска");
            }

        }

        private void buttonExit_Click(object sender, EventArgs e)
        {
            this.Close();
            //Application.Exit();
        }

        private void buttonSaveReport_Click(object sender, EventArgs e)
        {
            //Имя файла отчета
            string TempReportFileName = "Report_" +
DateTime.Now.ToString("dd_MM_yyyy_hhmmss");

            //Диалог сохранения файла отчета
            SaveFileDialog fd = new SaveFileDialog();
            fd.FileName = TempReportFileName;
            fd.DefaultExt = ".html";
            fd.Filter = "HTML Reports|*.html";

            if (fd.ShowDialog() == DialogResult.OK)
            {
                string ReportFileName = fd.FileName;

                //Формирование отчета
                StringBuilder b = new StringBuilder();
                b.AppendLine("<html>");

                b.AppendLine("<head>");
```

```
                b.AppendLine("<meta http-equiv='Content-Type' content='text/html;
charset=UTF-8'/>");
                b.AppendLine("<title>" + "Отчет: " + ReportFileName + "</title>");
                b.AppendLine("</head>");

                b.AppendLine("<body>");

                b.AppendLine("<h1>" + "Отчет: " + ReportFileName + "</h1>");
                b.AppendLine("<table border='1'>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Время чтения из файла</td>");
                b.AppendLine("<td>" + this.textBoxFileReadTime.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Количество уникальных слов в файле</td>");
                b.AppendLine("<td>" + this.textBoxFileReadCount.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Слово для поиска</td>");
                b.AppendLine("<td>" + this.textBoxFind.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Максимальное расстояние для нечеткого
поиска</td>");
                b.AppendLine("<td>" + this.textBoxMaxDist.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Время четкого поиска</td>");
                b.AppendLine("<td>" + this.textBoxExactTime.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr>");
                b.AppendLine("<td>Время нечеткого поиска</td>");
                b.AppendLine("<td>" + this.textBoxApproxTime.Text + "</td>");
                b.AppendLine("</tr>");

                b.AppendLine("<tr valign='top'>");
                b.AppendLine("<td>Результаты поиска</td>");
                b.AppendLine("<td>");
                b.AppendLine("<ul>");

                foreach (var x in this.listBoxResult.Items)
                {
                    b.AppendLine("<li>" + x.ToString() + "</li>");
                }

                b.AppendLine("</ul>");
                b.AppendLine("</td>");
                b.AppendLine("</tr>");
```

```
                b.AppendLine("</table>");

                b.AppendLine("</body>");
                b.AppendLine("</html>");

                //Сохранение файла
                File.AppendAllText(ReportFileName, b.ToString());

                MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);
            }
        }
    }
}
```

7.5 Лабораторная работа 5

7.5.1 Вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EditDistanceProject
{
    public static class EditDistance
    {
        /// <summary>
        /// Вычисление расстояния Дамерау-Левенштейна
        /// </summary>
        public static int Distance(string str1Param, string str2Param)
        {
            if ((str1Param == null) || (str2Param == null)) return -1;

            int str1Len = str1Param.Length;
            int str2Len = str2Param.Length;

            //Если хотя бы одна строка пустая, возвращается длина другой строки
            if ((str1Len == 0) && (str2Len == 0)) return 0;
            if (str1Len == 0) return str2Len;
            if (str2Len == 0) return str1Len;

            //Приведение строк к верхнему регистру
            string str1 = str1Param.ToUpper();
            string str2 = str2Param.ToUpper();

            //Объявление матрицы
            int[,] matrix = new int[str1Len + 1, str2Len + 1];

            //Инициализация нулевой строки и нулевого столбца матрицы
            for (int i = 0; i <= str1Len; i++) matrix[i, 0] = i;
            for (int j = 0; j <= str2Len; j++) matrix[0, j] = j;
```

```
            //Вычисление расстояния Дамерау-Левенштейна
            for (int i = 1; i <= str1Len; i++)
            {
                for (int j = 1; j <= str2Len; j++)
                {
                    //Эквивалентность символов, переменная symbEqual соответствует
m(s1[i],s2[j])
                    int symbEqual = ((str1.Substring(i - 1, 1) == str2.Substring(j
- 1, 1)) ? 0 : 1);

                    int ins = matrix[i, j - 1] + 1; //Добавление
                    int del = matrix[i - 1, j] + 1; //Удаление
                    int subst = matrix[i - 1, j - 1] + symbEqual; //Замена

                    //Элемент матрицы вычисляется как минимальный из трех случаев
                    matrix[i, j] = Math.Min(Math.Min(ins, del), subst);

                    //Дополнение Дамерау по перестановке соседних символов
                    if ((i > 1) && (j > 1) &&
                        (str1.Substring(i - 1, 1) == str2.Substring(j - 2, 1)) &&
                        (str1.Substring(i - 2, 1) == str2.Substring(j - 1, 1)))
                    {
                        matrix[i, j] = Math.Min(matrix[i, j], matrix[i - 2, j - 2]
+ symbEqual);
                    }
                }
            }
            //Возвращается нижний правый элемент матрицы
            return matrix[str1Len, str2Len];
        }


    }
}
```

7.6 Лабораторная работа 6

7.6.1 Фрагмент программы, реализующей работу с делегатами

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Delegates
{

    //Делегаты - аналог процедурного типа в Паскале.
    //Делегат - это не тип класса, а тип метода.
    //Делегат определяет сигнатуру метода (типы параметров и возвращаемого
значения).
    //Если создается метод типа делегата, то у него должна быть сигнатура как у
делегата.
    //Метод типа делегата можно передать как параметр другому методу.
```

```
    //Название делегата при объявлении указывается "вместо" названия метода
    delegate int PlusOrMinus(int p1, int p2);


    class Program
    {
        //Методы, реализующие делегат (методы "типа" делегата)
        static int Plus(int p1, int p2) { return p1 + p2; }
        static int Minus(int p1, int p2) { return p1 - p2; }

        /// <summary>
        /// Использование обощенного делегата Func<>
        /// </summary>
        static void PlusOrMinusMethodFunc(string str, int i1, int i2, Func<int,
int, int> PlusOrMinusParam)
        {
            int Result = PlusOrMinusParam(i1, i2);
            Console.WriteLine(str + Result.ToString());

            // Func<int, string, bool> - делегат принимает параметры типа int и
string и возвращает bool

            // Если метод должен возвращать void, то используется делегат Action
            // Action<int, string> - делегат принимает параметры типа int и string
и возвращает void
            // Action как правило используется для разработки групповых делегатов,
которые используются в событиях

        }

        /// <summary>
        /// Использование делегата
        /// </summary>
        static void PlusOrMinusMethod(string str, int i1, int i2, PlusOrMinus
PlusOrMinusParam)
        {
            int Result = PlusOrMinusParam(i1, i2);
            Console.WriteLine(str + Result.ToString());
        }

        static void Main(string[] args)
        {
            int i1 = 3;
            int i2 = 2;

            PlusOrMinusMethod("Плюс: ", i1, i2, Plus);
            PlusOrMinusMethod("Минус: ", i1, i2, Minus);

            //Создание экземпляра делегата на основе метода
            PlusOrMinus pm1 = new PlusOrMinus(Plus);
            PlusOrMinusMethod("Создание экземпляра делегата на основе метода: ",
i1, i2, pm1);

            //Создание экземпляра делегата на основе 'предположения' делегата
            //Компилятор 'пердполагает' что метод Plus типа делегата
            PlusOrMinus pm2 = Plus;
```

```
            PlusOrMinusMethod("Создание экземпляра делегата на основе
'предположения' делегата: ", i1, i2, pm2);

            //Создание анонимного метода
            PlusOrMinus pm3 = delegate(int param1, int param2)
            {
                return param1 + param2;
            };
            PlusOrMinusMethod("Создание экземпляра делегата на основе анонимного
метода: ", i1, i2, pm2);

            PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-
выражения 1: ", i1, i2,
                (int x, int y) =>
                {
                    int z = x + y;
                    return z;
                }
                );

            PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-
выражения 2: ", i1, i2,
                (x, y) =>
                {
                    return x+y;
                }
                );

            PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-
выражения 3: ", i1, i2, (x, y) => x+y );



            ////////////////////////////////////////////////////////////
            Console.WriteLine("\n\nИспользование обощенного делегата Func<>");

            PlusOrMinusMethodFunc("Создание экземпляра делегата на основе метода:
", i1, i2, Plus);

            string OuterString = "ВНЕШНЯЯ ПЕРЕМЕННАЯ";

            PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-
выражения 1: ", i1, i2,
                (int x, int y) =>
                {
                    Console.WriteLine("Эта переменная объявлена вне лямбда-
выражения: " + OuterString);
                    int z = x + y;
                    return z;
                }
                );

            PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-
выражения 2: ", i1, i2,
                (x, y) =>
                {
```

```
                return x + y;
            }
            );

        PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-
выражения 3: ", i1, i2, (x, y) => x + y);


        ////////////////////////////////////////////////////////
        //Групповой делегат всегда возвращает значение типа void
        Console.WriteLine("Пример группового делегата");
        Action<int, int> a1 = (x, y) => { Console.WriteLine("{0} + {1} = {2}",
x, y, x + y); };
        Action<int, int> a2 = (x, y) => { Console.WriteLine("{0} - {1} = {2}",
x, y, x - y); };
        Action<int, int> group = a1 + a2;
        group(5, 3);

        Action<int, int> group2 = a1;
        Console.WriteLine("Добавление вызова метода к групповому делегату");
        group2 += a2;
        group2(10, 5);
        Console.WriteLine("Удаление вызова метода из группового делегата");
        group2 -= a1;
        group2(20, 10);

        Console.ReadLine();
    }
  }
}
```

7.6.2 Фрагмент программы, реализующей работу с рефлексией

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;

namespace Reflection
{
    class Program
    {
        /// <summary>
        /// Проверка, что у свойства есть атрибут заданного типа
        /// </summary>
        /// <returns>Значение атрибута</returns>
        public static bool GetPropertyAttribute(PropertyInfo checkType, Type
attributeType, out object attribute)
        {
            bool Result = false;
            attribute = null;

            //Поиск атрибутов с заданным типом
            var isAttribute = checkType.GetCustomAttributes(attributeType, false);
            if (isAttribute.Length > 0)
```

```
            {
                Result = true;
                attribute = isAttribute[0];
            }

            return Result;
        }


        static void Main(string[] args)
        {
            Type t = typeof(ForInspection);

            Console.WriteLine("Тип " + t.FullName + " унаследован от " +
t.BaseType.FullName);
            Console.WriteLine("Пространство имен " + t.Namespace);
            Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

            Console.WriteLine("\nКонструкторы:");
            foreach (var x in t.GetConstructors())
            {
                Console.WriteLine(x);
            }

            Console.WriteLine("\nМетоды:");
            foreach (var x in t.GetMethods())
            {
                Console.WriteLine(x);
            }

            Console.WriteLine("\nСвойства:");
            foreach (var x in t.GetProperties())
            {
                Console.WriteLine(x);
            }

            Console.WriteLine("\nПоля данных (public):");
            foreach (var x in t.GetFields())
            {
                Console.WriteLine(x);
            }

            Console.WriteLine("\nСвойства, помеченные атрибутом:");
            foreach (var x in t.GetProperties())
            {
                object attrObj;
                if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))
                {
                    NewAttribute attr = attrObj as NewAttribute;
                    Console.WriteLine(x.Name + " - " + attr.Description);
                }
            }

            Console.WriteLine("\nВызов метода:");

            //Создание объекта
```

```
            //ForInspection fi = new ForInspection();

            //Можно создать объект через рефлексию
            ForInspection fi = (ForInspection)t.InvokeMember(null,
BindingFlags.CreateInstance, null, null, new object[] { });

            //Параметры вызова метода
            object[] parameters = new object[] { 3, 2 };

            //Вызов метода
            object Result = t.InvokeMember("Plus", BindingFlags.InvokeMethod,
null, fi, parameters);
            Console.WriteLine("Plus(3,2)={0}", Result);

            Console.ReadLine();
        }
    }
}
```

7.6.3 Реализация атрибута

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Reflection
{
    /// <summary>
    /// Класс атрибута
    /// </summary>
    [AttributeUsage(AttributeTargets.Property, AllowMultiple=false,
Inherited=false)]
    public class NewAttribute : Attribute
    {
        public NewAttribute() {}
        public NewAttribute(string DescriptionParam)
        {
            Description = DescriptionParam;
        }

        public string Description { get; set; }
    }
}
```

7.6.4 Пример инспектируемого класса

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Reflection
{
```

```
    /// <summary>
    /// Класс для исследования с помощью рефлексии
    /// </summary>
    public class ForInspection
    {
        public ForInspection() { }
        public ForInspection(int i) { }
        public ForInspection(string str) { }

        public int Plus(int x, int y) { return x + y; }
        public int Minus(int x, int y) { return x - y; }

        [NewAttribute("Описание для property1")]
        public string property1
        {
            get { return _property1; }
            set { _property1 = value; }
        }
        private string _property1;

        public int property2 { get; set; }

        [NewAttribute(Description = "Описание для property3")]
        public double property3 { get; private set; }


        public int field1;
        public float field2;
    }
}
```

8 Контрольные вопросы

8.1 Лабораторная работа 1

1. Каким образом реализуется консольный ввод-вывод в C#?
2. Какие способы используются для преобразования строковых значений в числовые?
3. Что такое пространство имен (namespace) и для чего используются пространства имен?
4. Какие операторы используются для организации условий?
5. Какие операторы используются для организации циклов?

8.2 Лабораторная работа 2

1. Какие типы элементов может содержать класс в языке C#?

2. В чем особенность абстрактных классов?
3. Что такое интерфейс?
4. В чем особенность виртуальных и абстрактных методов? Их сходства и различия, отличия от обычных методов.
5. Что такое свойство (property)?
6. Что означает ключевое слово «base» при объявлении конструктора класса?
7. От какого количества классов и интерфейсов может быть унаследован класс в языке C#? С чем связана такая схема наследования?

8.3 Лабораторная работа 3

1. Какие типы коллекций в языке C# Вы знаете?
2. Чем отличаются обобщенные и необобщенные коллекции?
3. В чем особенность использования коллекционного класса ArrayList?
4. В чем особенность использования коллекционного класса List?
5. В чем особенность записи и чтения элементов при использовании коллекций ArrayList и List?
6. Что такое индексаторы и как они используются?
7. В чем отличие индексатора от свойства?

8.4 Лабораторная работа 4

1. Как создать приложение Windows Forms в Visual Studio?
2. Как в программе на языке C# прочитать содержимое текстового файла?
3. Как используется класс OpenFileDialog для выбора имени файла?
4. Каким образом производится работа с компонентом TextBox?
5. Каким образом производится работа с компонентом Label?
6. Каким образом производится работа с компонентом ListBox?
7. Как добавить на форму кнопку и прикрепить к ней обработчик события?

8.5 Лабораторная работа 5

1. Что такое расстояние Левенштейна и для чего оно используется?
2. Приведите пример вычисления расстояния Левенштейна для двух строк с использованием добавления, удаления и замены символов.
3. Опишите основные принципы работы алгоритма Вагнера-Фишера на основе вычисления значений матрицы минимальных расстояний.
4. Каким образом можно создать библиотеку классов и сделать ее доступной для использования в других проектах?
5. Каким образом в C# можно измерить время выполнения фрагмента программного кода?

8.6 Лабораторная работа 6

1. Что такое делегат?
2. Как передать в метод параметр типа делегат?
3. Как написать метод, соответствующий делегату?
4. Что такое лямбда-выражение?
5. Как написать лямбда-выражение, соответствующее делегату?
6. Для чего используются обобщенные делегаты Func и Action?
7. Какие возможности предоставляет рефлексия в C#?
8. Каковы основные возможности класса Type?
9. Что такое атрибуты и для чего они используются?
10. Как создать собственный атрибут?
11. Как вызвать метод класса с использованием рефлексии?
12. Для чего используется метод InvokeMember класса Type?

9 Литература

1. Нейгел К., Ивьен Б., Глинн Д., Уотсон К. C# 4.0 и платформа .NET 4 для профессионалов. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2011. – 1440 с.

2. Шилдт Г. C# 4.0: полное руководство. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2013. – 1056 с.
3. Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов. – СПб.: Питер, 2009. – 432 с.
4. Расстояние Левенштейна. [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/%D0%E0%F1%F1%F2%EE%FF%ED%E8%E5_%CB%E5%E2%E5%ED%F8%F2%E5%E9%ED%E0 (дата обращения: 12.12.2014).