

Разработка консольных и оконных приложений на языке C#.

А.Г. Пискунов

16 июня 2020 г.

СОДЕРЖАНИЕ	5
1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА	6
1.1 Требования к приложениям и данным	7
2 МОДУЛЬ: ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ C#	7
2.1 ЛАБОРАТОРНАЯ РАБОТА: ВЫЧИСЛЕНИЕ ПРОСТЫХ ЧИСЕЛ	7
2.2 ЛАБОРАТОРНАЯ РАБОТА: РЕШЕТО ЭРАТОСФЕНА	8
2.3 ЛАБОРАТОРНАЯ РАБОТА: ПЕРЕДАЧА ФАКТИЧЕСКИХ ПАРАМЕТРОВ В МЕТОДЫ	8
2.4 ЛАБОРАТОРНАЯ РАБОТА: ОБРАБОТКА СТРОЧЕК	9
2.5 ЛАБОРАТОРНАЯ РАБОТА: ВЫБОР ДВУХ СТОЛБЦОВ (ШИРОТА И ДОЛГОТА) ИЗ ТЕКСТОВОГО ФАЙЛА	10
2.6 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ	11
3 МОДУЛЬ: ПРОГРАММИРОВАНИЕ НА ПЛАТФОРМЕ .Net FRAMEWORK	11
3.1 ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК	11
3.2 ЛАБОРАТОРНАЯ РАБОТА: СОРТИРОВКИ И КОЛЛЕКЦИИ	12
3.3 ЛАБОРАТОРНАЯ РАБОТА: ЗАКРЫТИЕ ОКНА	12
3.4 ЛАБОРАТОРНАЯ РАБОТА: ДВОИЧНЫЕ ФАЙЛЫ	13
3.5 ЛАБОРАТОРНАЯ РАБОТА: ВЫВОД ТАБЛИЦЫ В XML ФОРМАТЕ	13
3.6 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ	14
4 ВОПРОСЫ ДЛЯ ДИФФЕРЕНЦИАЛЬНОГО ЗАЧЕТА	15
4.1 Простые вопросы	15
4.2 Базовые вопросы	19
4.3 Дополнительные вопросы	20
5 МОДУЛЬ: СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ	22
5.1 ЛАБОРАТОРНАЯ РАБОТА: УПРАВЛЯЮЩИЕ ЭЛЕМЕНТЫ	22
5.2 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛОГОВОЕ ОКНО OkCancel	23
5.3 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛОГОВОЕ ОКНО ВВОДА ДАННЫХ	23
5.4 ЛАБОРАТОРНАЯ РАБОТА: МЕНЮ В ГЛАВНОМ ОКНЕ ПРИЛОЖЕНИЯ	24
5.5 ЛАБОРАТОРНАЯ РАБОТА: ПАНЕЛЬ ИНСТРУМЕНТОВ ДОЧЕРНЕГО ОКНА С ПРЕДСТАВЛЕНИЕМ ТАБЛИЧНЫХ ДАННЫХ	24
5.6 ЛАБОРАТОРНАЯ РАБОТА: ОКНО РЕДАКТИРОВАНИЯ ТАБЛИЦ	25
5.7 ЛАБОРАТОРНАЯ РАБОТА: МНОГОДОКУМЕНТНЫЙ ИНТЕРФЕЙС	26
5.8 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ	26
6 МОДУЛЬ: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММИРОВАНИЯ В .Net	27
6.1 ЛАБОРАТОРНАЯ РАБОТА: РЕСУРСЫ В ПРИЛОЖЕНИЯХ	27
6.2 ЛАБОРАТОРНАЯ РАБОТА: ПРИМИТИВЫ ГРАФИКИ	28

6.3	ЛАБОРАТОРНАЯ РАБОТА: МАСШТАБИРОВАНИЕ ЛОМАНОЙ	28
6.4	ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ МЫШИ	29
6.5	ЛАБОРАТОРНАЯ РАБОТА: СКАЧИВАНИЕ ФАЙЛОВ	30
6.6	ЛАБОРАТОРНАЯ РАБОТА: ЖУРНАЛИРОВАНИЕ РАБОТЫ ПРИЛОЖЕНИЯ	31
6.7	ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ	31
7	ВОПРОСЫ ДЛЯ ЭКЗАМЕНА	32
7.1	Создание Оконных Приложений	32
7.2	Дополнительные Возможности Программирования В .Net	33
8	ТЕМАТИКА КУРСОВЫХ ПРОЕКТОВ	35
9	ТРЕБОВАНИЯ К СДАВАЕМЫМ РАБОТАМ	37
10	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ВВЕДЕНИЕ В ЯЗЫК ПРО- ГРАММИРОВАНИЯ C#	40
10.1	ВОПРОСЫ КОМПИЛЯЦИИ ПРИЛОЖЕНИЙ В СРЕДЕ МОНО Для *NIX	40
10.2	ЛАБОРАТОРНАЯ РАБОТА: ВЫЧИСЛЕНИЕ ПРОСТЫХ ЧИСЕЛ	40
10.3	ЛАБОРАТОРНАЯ РАБОТА: РЕШЕТО ЭРАТОСФЕНА	41
10.4	ЛАБОРАТОРНАЯ РАБОТА: ПЕРЕДАЧА ФАКТИЧЕСКИХ ПА- РАМЕТРОВ В МЕТОДЫ	42
10.5	ЛАБОРАТОРНАЯ РАБОТА: ОБРАБОТКА СТРОЧЕК	44
10.6	ЛАБОРАТОРНАЯ РАБОТА: ВЫБОР ДВУХ СТОЛБЦОВ (ШИ- РОТА И ДОЛГОТА) ИЗ ТЕКСТОВОГО ФАЙЛА	45
11	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ПРОГРАММИРОВАНИЕ НА ПЛАТФОРМЕ .Net FRAMEWORK	49
11.1	ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕ- СКИ ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК	49
11.2	ЛАБОРАТОРНАЯ РАБОТА: СОРТИРОВКИ И КОЛЛЕКЦИИ	52
11.3	ЛАБОРАТОРНАЯ РАБОТА: ЗАКРЫТИЕ ОКНА	54
11.4	ЛАБОРАТОРНАЯ РАБОТА: ДВОИЧНЫЕ ФАЙЛЫ	56
11.5	ЛАБОРАТОРНАЯ РАБОТА: ВЫВОД ТАБЛИЦЫ В XML ФОР- МАТЕ	59
12	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ	62
12.1	ЛАБОРАТОРНАЯ РАБОТА: УПРАВЛЯЮЩИЕ ЭЛЕМЕНТЫ	63
12.1.1	Законченный пример конструктора окна	63
12.2	ЛАБОРАТОРНАЯ РАБОТА: ДИАЛОГОВОЕ ОКНО OkCancel	64
12.3	ЛАБОРАТОРНАЯ РАБОТА: ДИАЛОГОВОЕ ОКНО ВВОДА ДАН- НЫХ	66
12.4	ЛАБОРАТОРНАЯ РАБОТА: МЕНЮ В ГЛАВНОМ ОКНЕ ПРИ- ЛОЖЕНИЯ	70
12.5	ЛАБОРАТОРНАЯ РАБОТА: ПАНЕЛЬ ИНСТРУМЕНТОВ ДО- ЧЕРНЕГО ОКНА С ПРЕДСТАВЛЕНИЕМ ТАБЛИЧНЫХ ДАННЫХ	71
12.5.1	Окно с представлением табличных данных	71
12.5.2	Панель инструментов дочернего окна	74

12.6	ЛАБОРАТОРНАЯ РАБОТА: ОКНО РЕДАКТИРОВАНИЯ ТАБЛИЦ	77
12.6.1	Редактирование таблицы, часть 1	77
12.6.2	Редактирование таблицы, часть 2	80
12.7	ЛАБОРАТОРНАЯ РАБОТА: МНОГОДОКУМЕНТНЫЙ ИНТЕРФЕЙС	84
13	МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММИРОВАНИЯ В .Net	88
13.1	ЛАБОРАТОРНАЯ РАБОТА: РЕСУРСЫ В ПРИЛОЖЕНИЯХ	88
13.1.1	Построение приложения с ресурсами	89
13.1.2	Рисование и доступ к ресурсам приложения	90
13.1.3	Таймер для поочередного отображения картинок	90
13.2	ЛАБОРАТОРНАЯ РАБОТА: ПРИМИТИВЫ ГРАФИКИ	91
13.3	ЛАБОРАТОРНАЯ РАБОТА: МАСШТАБИРОВАНИЕ ЛОМОНОЙ	95
13.3.0.1	Использование Doxygen для документирования кода.	105
13.4	ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ МЫШИ	105
13.4.1	Введение в использование мыши	106
13.4.2	Контекстное меню	107
13.5	ЛАБОРАТОРНАЯ РАБОТА: СКАЧИВАНИЕ ФАЙЛОВ	108
13.6	ЛАБОРАТОРНАЯ РАБОТА: ЖУРНАЛИРОВАНИЕ РАБОТЫ ПРИЛОЖЕНИЯ	109
14	ОТВЕТЫ	112
14.1	Простые Вопросы	112
14.2	Базовые Вопросы	116
14.3	Дополнительные Вопросы	130
14.4	Создание Оконных Приложений	146
14.5	Дополнительные Возможности Программирования В .Net	152
15	НЕКОТОРЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И ПРИМЕРЫ	160
15.1	Исходные текст всех примеров и лабораторных	161
15.2	Конвертация файлов и абстрактные классы	161
15.3	Пример использования лямбда выражений	164
15.4	Явно заданная продолжительность тика	165
15.5	Явно заданное кодирование файла	166
15.6	Контекстное меню	168
15.7	Скачивание файла	170
15.8	Клик снаружи или внутри многоугольника?	171
15.9	Теннис для двух игроков	173
15.10	Построение графиков при помощи управляющего элемента Chart	177
15.11	Использование трехмерной графики	179
15.12	Порождение дочернего процесса	179
15.13	Операция инкремента в двух нитях	180
15.14	Пример мертвой блокировки	182
15.15	Пример для генерации CSV файла из GPX - файла	182
15.16	Скачивание карт Яндексa и ОпенСтритМап	185
15.17	Утилита для генерации ресурсов	192
15.18	Примеры файла AssemblyInfo.cs	196

15.19 Проблема с разделителем дробной части	196
15.20 Сериализация и десериализация	197
16 СПИСКИ ЛАБОРАТОРНЫХ РАБОТ ДЛЯ СОКРАЩЕННЫХ КУРСОВ	203
16.1 Методические указания для курса Технологии .NET В Разработке ИС	203
16.1.1 Вычисление простых чисел	203
16.1.2 Передача фактических параметров в методы	203
16.1.3 Обработка строчек	203
16.1.4 Передача аргументов командной строки в приложение	204
16.1.4.1 Законченный пример, написанный только средствами языка	205
16.1.4.2 Законченный пример 2	206
16.1.5 Выбор двух столбцов CSV файла	209
16.1.6 Чтение координат трека	210
16.1.7 Представление табличных данных	210
16.1.7.1 Законченный пример	210
16.1.8 Рисование немасштабированной ломаной	212
16.2 Методические указания для курса Проектирование Управляющих, Информационных и Интеллектуальных систем	213
16.2.1 Конвертация текстовых файлов с записями из целых чисел в двоичные и наоборот	214
16.2.1.1 Законченные примеры конвертации файлов	214
16.2.2 Обработка события FormClosing	216
16.2.2.1 Законченный пример	218
16.2.3 Создание диалогового окна OkCancel	219
16.2.3.1 Законченный пример	219
16.2.4 Создание диалогового окна ввода данных	221
16.2.4.1 Законченный пример	221
16.2.5 Меню в главном окне приложения	223
16.2.5.1 Законченный пример	223
16.2.6 Представление табличных данных	225
16.2.6.1 Законченный пример	225
16.2.7 Открытие нескольких дочерних окон	228
16.2.7.1 Класс главного окна приложения для законченного примера	228
16.2.8 Экспорт таблицы	231
16.2.9 Окно редактирования таблицы	233
16.2.9.1 Законченный пример	233
16.2.10 Рисование масштабированной ломаной	237
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ	242
СПИСОК ЛИТЕРАТУРЫ	245

1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Упражнения в лабораторный практикум подбирались для обучения именно программированию. Их все можно выполнить ни разу не запуская дизайнер форм в частности и MS Visual Studio как таковую. Вполне достаточно выполнять компилирование приложений из настроенного (при установке студии) окна интерпретатора cmd.exe командной строки. Что-то вроде такого:

```

Visual Studio Command Prompt (2)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.

G:\agp\96\praktikums\smp\cs>csc *.cs
Microsoft (R) Visual C# 2010 Compiler version 4.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.

G:\agp\96\praktikums\smp\cs>dir *.exe *.cs
Том в устройстве G имеет метку WORK
Серийный номер тома: 8E15-8000

Содержимое папки G:\agp\96\praktikums\smp\cs
03.04.2019  20:31                4 096 app.exe
Содержимое папки G:\agp\96\praktikums\smp\cs
03.04.2019  20:23                2 246 app.cs
                2 файлов             6 342 байт
                0 папок            14 749 822 976 байт свободно
G:\agp\96\praktikums\smp\cs>_
  
```

Компиляция одного из примеров

Замечание

В случае MS Visual Studio 2010 настроенное окно интерпретатора командной строки появлялось в случае установки C++. Так что, устанавливался C++ и C#.

В крайнем случае, не слишком тяжело настроить командный файл для работы из обычного окна командного интерпретатора:

```

---- File:./cs/cs2.cmd

SET X=c:\WINDOWS\Microsoft.NET\Framework\v3.0
SET X=c:\WINDOWS\Microsoft.NET\Framework\v3.5
SET X=C:\Windows\Microsoft.NET\Framework\v4.0.30319

SET R=/r:System.Data.SQLite.dll
SET R=

%X%/csc /out:a.exe %R% /unsafe *.cs /nologo >.errors.txt

---- End Of File:./cs/cs2.cmd
  
```

Замечание 2

Собственно компиляция не слишком усложняется даже в случае операционной системы из семейства unix (*NIX). Например, для Ubuntu 18.x подготовка к компиляции и компиляция состояла из следующих шагов:

[10.1](#)

Кроме того, считается, что читатель либо освоил материал [29] и [30], либо такой язык программирования как Java или pascal и ему не требуется слишком подробных объяснений, например, по операторам цикла или операции применения функции. Последнюю версию документа можно найти [28]. А сам курс является основой для курса [18].

Основную литературу для курса см. [41, 36, 15, 39] и очень рекомендуется по каждому вопросу заглядывать в [3].

Сдача каждой работы (все равно лабораторной, модульной или домашнего задания) предполагает передачу файлов исходного кода приложения, файлов с тестами, файлов командного интерпретатора и файла с отчетом о работе. Кроме того, отчеты по модульным и курсовым работам сдаются в еще и в напечатанном виде.

1.1 Требования к приложениям и данным

Любое приложение на языке C# должно:

- небольшие порции (одно, два, три значения) входных данных брать из аргументов командной строки.
- большие порции данных (текст или координаты перемещения транспортного средства) брать из текстового файла, если не оговорено противное, то через стандартный ввод.
- начиная с лабораторной 2.4 по аргументам командной строки (ключам) '-?', '/?', '-h', '/h', '-help', '/help' выдавать подсказку для использования.
- если не оговорено противное, результаты работы выводить в стандартный вывод.
- по ключу '-v' в стандартный вывод ошибок выводить дополнительную информацию о ходе работы приложения.
- начиная со второго модуля каждый проект должен содержать файл метаописания сборки AssemblyInfo.cs с упоминанием автора и версии приложения (см. раздел 15.18).
- Во всех текстовых файлах, которые будут использоваться для ввода данных, символ '#' является комментарием, сам символ и строка справа от него должны игнорироваться приложениями.

2 МОДУЛЬ: ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ C#

2.1 ЛАБОРАТОРНАЯ РАБОТА: ВЫЧИСЛЕНИЕ ПРОСТЫХ ЧИСЕЛ

Тема:

Введение в язык программирования C#.

Цель:

Создание консольного приложения для вычисления всех простых чисел до заданного по определению.

Задание:

- Познакомиться с описанием переменных встроенных типов (int, bool).
- Познакомиться с чтением целых чисел из стандартного входа.
- Познакомиться с выражениями, операторами - выражение и операторами цикла for и while.
- Познакомиться с возможностями форматирования данных и выводом в стандартный выход.

2.2 ЛАБОРАТОРНАЯ РАБОТА: РЕШЕТО ЭРАТОСФЕНА

Тема:

Использование массивов в языке C#.

Цель:

Вычисления всех простых чисел до заданного алгоритмом Решето Эратосфена.

Задание:

- Познакомиться с описанием, захватом памяти для массивов и использованием массивов.
- Познакомиться с типами для измерения времени (DateTime, TimeSpan).
- Сравнить время работы алгоритмов по определению и Решето Эратосфена. Время выводить секундами, то есть, не 1 минута 1 секунда, а 61 секунда.
- В усложненной версии приложения требуется не просто вывести невычеркнутые числа массива (то есть, простые), а сдвинуть простые числа в начало массива и уменьшить его длину, так что бы массив не содержал нули (вычеркнутые значения).

2.3 ЛАБОРАТОРНАЯ РАБОТА: ПЕРЕДАЧА ФАКТИЧЕСКИХ ПАРАМЕТРОВ В МЕТОДЫ

Тема:

Введение в язык программирования C#.

Цель:

Создание консольного приложения для использования статических методов.

Задание:

- Выучить три группы типов C# (значимые типы, ссылочные типы и строки).
- Познакомиться со способами применения статических и нестатических методов.
- Познакомиться с правилами передачи фактических параметров в методы, и особенностями их (параметров) изменения.
- Познакомиться с описанием своих классов и структур, Создать новый класс и новую структуру для объявления и передачи параметров этого типа в метод.
- Создать статические методы для изменения значений переменных из всех трех групп типов: целого и структуры, массива и класса, строки.
- Приложение должно содержать примеры перегрузки методов, вызова статических методов из своего класса (из метода Program.Main вызывать метод Program.f); вызова статического метода из чужого класса; вызова нестатического метода.

2.4 ЛАБОРАТОРНАЯ РАБОТА: ОБРАБОТКА СТРОЧЕК

Тема:

Основы работы с текстовыми данными.

Цель:

Использование методов класса string и обработка аргументов командной строки.

Задание:

- Познакомиться с информационным сервисом для разработчиков программного обеспечения MSDN ([3]).
- Познакомиться с передачей аргументов командной строки в C#.
- Познакомиться с типами для работы со текстами (char, string, StringBuilder).
- Познакомиться с таблицами кодирования символов национальных алфавитов (кириллицы) (1251, 866, UTF-16, UTF-8).
- Познакомиться с методами кодирования символов национальных алфавитов (кириллицы) (Encoding).

2.5 ЛАБОРАТОРНАЯ РАБОТА: ВЫБОР ДВУХ СТОЛБЦОВ (ШИРОТА И ДОЛГОТА) ИЗ ТЕКСТОВОГО ФАЙЛА

Тема:

Обработка исключительных состояний.

Цель:

Использование методов конвертации текстовых представлений чисел в числа и наоборот (TryParse, Parse, ToString).

Задание:

- Научиться выдавать осмысленную подсказку по использованию в ответ на ключи '-?' или '/?'.
- Научиться получать номера колонок для ввода через аргументы командной строки с полной диагностикой ввода. Например:

```
c:/tmp>a.exe -l1t NN1 -lng NN2
```

где ключ '-l1t' задает номер колонки для широты, а '-lng' - для долготы. Целые вводить через метод TryParse, то есть, без возникновения исключительных состояний.

- Использовать необходимые методы классов char и string для обработки текстов и выдачи диагностики об ошибках, возникающих в процессе ввода (номер строки и позиция в строке, в котором возникла ошибка).
- Освоить использование блоков try-catch-finally и вывод текста ошибки из класса Exception;
- Познакомиться со спецификаторами места вывода методов Console.WriteLine, Console.ReadLine, String.Format. Использовать эти возможности для вывода градусов широты и долготы с заданной точностью в 6 или 8 знаков после запятой.

Замечание о компиляции

Использование ключа /debug в строке компиляции проекта

```
c:/tmp>csc /debug /out:a.exe *.cs
```

приведет к тому, что сообщения о необработанных исключительных состояниях будут содержать номер строки файла, в которой случилось ошибка. Эта информация хранится в файле a.pdb, находящуюся рядом со сборкой a.exe.

2.6 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ

Разработать консольное приложение для

1. символьного сложения двух целых чисел произвольной длины.
2. символьного умножения двух целых чисел произвольной длины.
3. символьного деления двух целых чисел произвольной длины.
4. простого редактирования текстового файла (требуется убирать пробелы и табуляции между словами и знаками препинания, вставлять отсутствующий пробел между знаком препинания и словом).
5. изменения кодировки текстового файла.

```
a.exe -866to1251 | -1251to866 | -1251toutf8
```

6. спиралевидной нумерации ячеек двумерного массива (int [,] arr;)
7. изменения одной из трех форм представления градусов (создав свой класс для конвертации):
 - градусы с дробной частью;
 - целые градусы, минуты с дробной частью;
 - целые градусы, целые минуты, секунды с дробной частью.
8. вычисления расстояния (или степени похожести) между двумя отдельными словами. Например, между словами 'ошибка' и 'ошибку' расстояние 1.
9. удаления однострочных и многострочных комментариев из программ.
10. замены макросов в тексте программы (на Си) на их значения.
11. построчного сравнения двух текстовых файлов (см. утилиту diff, см. <https://ru.wikipedia.org/wiki/Diff>).

3 МОДУЛЬ: ПРОГРАММИРОВАНИЕ НА ПЛАТФОРМЕ .Net FRAMEWORK

3.1 ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК

Тема:

Приложения, именованные области видимости и динамически подключаемые библиотеки.

Цель:

Разработать консольное приложение:

- либо для демонстрации диалогового окна,
- либо для демонстрации использования библиотеки обработки командной строки [19].

Задание:

- Познакомится со сборками .Net.
- Познакомится с пространствами имен (областями видимости) и правилами использования типов из сборки.
- Подключить одну из предлагаемых сборок к консольному приложению и продемонстрировать использование типов этой сборки.

3.2 ЛАБОРАТОРНАЯ РАБОТА: СОРТИРОВКИ И КОЛЛЕКЦИИ

Тема:

Основы использования Framework .Net.

Цель:

Разработать консольное приложение для сортировки текстового CSV файла (фамилия и возраст студента) по фамилии или по возрасту (см. 1.1). Данные получить из заданного в командной строке файла (не стандартного ввода).

Задание:

- Познакомиться с классами File, StreamReader;
- Познакомиться со списками.
- Познакомиться с шаблонами.
- Познакомиться с возможностями .Net для сортировок коллекций (интерфейс IComparable).

3.3 ЛАБОРАТОРНАЯ РАБОТА: ЗАКРЫТИЕ ОКНА

Тема:

Основы использования Framework .Net.

Цель:

Разработать оконное приложение, которое уточняет намерение оператора закрыть окно. Студенты с четными номерами зачетов показывают окно вопроса с кнопками Да - Нет, с нечетными номерами зачетов показывают окно вопроса с кнопками Ок - Cancel.

Задание:

- Познакомиться со классом Form.
- Познакомиться со способами демонстрации окон.
- Научиться задавать вопросы оператору при помощи стандартного окна MessageBox и перечисления MessageBoxButtons.
- Научиться получать ответ оператора приложения при помощи стандартного окна MessageBox и перечисления DialogResult.
- Познакомиться с делегатами, событием FormClosing и обработчиками событий.

Окно или иначе форма - поверхность экрана компьютера (прямоугольник) для взаимодействия приложения с пользователем, достаточно самостоятельна в том смысле, что может появляться на экране как нечто отдельное.

3.4 ЛАБОРАТОРНАЯ РАБОТА: ДВОИЧНЫЕ ФАЙЛЫ

Тема:

Потоки ввода - вывода данных.

Цель:

Разработать консольное приложение, которое преобразует таблицу с числами из текстового файла в двоичный и наоборот, с возможной любой сортировкой. Усложненная реализация лабораторной должна предполагать возможность задавать вид таблицы (таблица из двух колонок целых чисел, таблица из трех колонок целых чисел, таблица из двух колонок вещественных чисел, таблица из трех колонок вещественных чисел) и, желательно использование абстрактного класса как родителя класса записи для всех четырех предлагаемых версий таблицы.

Задание:

- Познакомиться со классами File и Directory.
- Познакомиться с интерфейсом IDisposable и третьей формой использования using.
- Познакомиться со абстрактными классами TextReader, TextWriter.
- Познакомиться со классами BinaryReader, BinaryWriter.
- Познакомиться со шаблонами списков.

3.5 ЛАБОРАТОРНАЯ РАБОТА: ВЫВОД ТАБЛИЦЫ В XML ФОРМАТЕ

Тема:

XML документы, сериализация и десериализация.

Цель:

Разработать консольное приложение, которое преобразует таблицу из CSV-файла в XML - файл. Разделителем полей будет считаться символ точки с запятой - ';', два разделителя не считаются одним (как в случае пробелов), а задают пустое поле. Входной CSV-файл может иметь различную кодировку, выходной XML - файл должен иметь кодировку UTF-8.

Задание:

- Познакомиться с пространством имен (областью видимости) System.Xml;
- Познакомиться с XML форматом.
- Номер строки в исходном CSV - файле выводить в качестве атрибута соответствующей записи в XML - файле.
- Познакомиться с шаблонами словарей, предусмотреть возможность явного именования одного или более полей в записи;
- Познакомиться с кодировками .Net. Использовать для ввода следующие кодировки: 866, 1251, UTF-8;
- Подготовить несколько тестов для демонстрации работоспособности приложения в виде командного файла.

3.6 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ

Разработать консольное приложение

1. и класс для вычисления проекции Меркатора, (см. [8, с. 58]), требуется разработать методы, для отображения градусов сферы в пиксели изображения и, наоборот, пикселей в градусы.
2. для вывода координат движения транспортного средства в виде XML - файла.
3. для работы приложения, следящего за состоянием кассы магазина на основе двоичного файла.
4. для работы приложения, следящего за состоянием кассы магазина на основе текстового файла.
5. для выполнения машины Тьюринга ([7]).
6. для транслитерации имен файлов в заданном каталоге. Требуется переименовать файлы, обработка ошибок, журнал работы приложения. По ключу -demo демонстрация - не переименовывать файлы, а только вывести список, какие файлы будут переименованы и новые имена.
7. для включения-отключения автозапуска для сменных накопителей (USB-флеш-накопителей, оптических дисков (CD-ROM, DVD-ROM)).
8. для использования парсера математических выражений ([2]).

9. выдачи сдачи. Через стандартный ввод приложение получает состояние кассы, то есть, не сортированный список пар

(номинал_монеты_или_банкноты, количество_монет) по одной паре в строке, через аргументы командной строки - общую сумму сдачи. Приложение подбирает монеты для заданной суммы, и, дополнительно, выводит отсортированное состояние кассы (по ключу -v).

10. утилиту для нахождения разницы в двух текстовых файлах произвольной кодировки. В качестве примера использовать утилиту diff, см. <https://ru.wikipedia.org/wiki/Diff>.
11. утилиту для нахождения разницы в двух каталогах.
12. утилиту для подсчета площади многоугольника на эллипсоиде.
13. утилиту для подсчета площади сфероидической трапеции.

Разработать публичную сборку (динамически подключаемую библиотеку) со Строгими Именами (см. https://professorweb.ru/my/csharp/assembly/level1/1_8.php).

4 ВОПРОСЫ ДЛЯ ДИФФЕРЕНЦИАЛЬНОГО ЗАЧЕТА

4.1 Простые вопросы

Для выполнения кода из этого раздела разработано специальное приложение (см. раздел 14.1) с примером условной компиляции. Символы для условной компиляции (макросы) могут передаваться из аргумента командной строки /define. Пример командной строки для построения ответа на 12-й вопрос:

```
csc /define:Q12 /out:q12.exe *.cs
```

1. Чему равна переменная i?

```
int i = 0;
int [,] a = new int [3,4];
i = a.Length;
```

2. Чему равна переменная i?

```
int i = 0;
int [,] a = new int [3,4];
i = a.Rank;
```

3. Чему равна переменная i?

```
int i = 0;
int [,] a = new int [3,4];
i = a.GetLength(0);
```

4. Чему равна переменная i?

```
int    i    = 0;
int [,] a = new int [3,4];
i = a.GetLength(1);
```

5. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i = b++;
```

6. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i = ++b;
```

7. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i = b + b++;
```

8. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i = b++ + b;
```

9. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i += b;
```

10. Чему равна переменная i?

```
int    b    = 2;
int    i    = 0;
i -= b;
```

11. Чему равна переменная i?

```
int    i    = 0;
i = 1/2;
```

12. Чему равна переменная i?

```
int    i    = 0;
i = 1%2;
```


13. Чему равна переменная f?

```
double f = 0.0;
f = -1.0;
f = Math.Abs(f);
```

14. Чему равна переменная f?

```
double f = 0.0;
f = -3.14159;
f = Math.Round(f);
```

15. Чему равна переменная f?

```
double f = -2.0;
f = Math.Pow(f, 2);
```

16. Чему равна переменная f?

```
double f = 0.0;
f = Math.PI;
```

17. Чему равна переменная f?

```
double f = 4.0;
f = Math.Sqrt(f);
```

18. Чему равна переменная f?

```
double f = 4.0;
f = Math.Sin(0.0);
```

19. Чему равна переменная f?

```
double f = 4.0;
f = Math.Cos(0.0);
```

20. Чему равна переменная f?

```
double f = 0.0;
f = true?2.0:0.0;
```

21. Чему равна переменная f?

```
double f = 0.0;
f = false?2.0:0.0;
```

22. Чему равна переменная f?

```
double f = 0.0;
int z[]=null;
f = z==null?2.0:0.0;
```

23. Чему равна переменная f?

```
double f = 0.0;
int z[]=new int[2];
f = z==null?2.0:0.0;
```

24. Чему равна переменная f?

```
double f = 0.0;
f = double.Parse("3");
```

25. Чему равна переменная f?

```
double f = 0.0;
f = double.Parse(" 3 + 2");
```

26. Чему равна переменная f?

```
double f = 0.0;
f = 1.0 + 2.0 * 3;
```

27. Чему равна переменная f?

```
double f = 0.0;
f = (1.0 + 2.0) * 3;
```

28. Чему равна переменная i?

```
int b = 2;
int i = 0;
i = b++ + b++ + b++;
```

29. Чему равна переменная i?

```
int b = 2;
int i = 0;
i = ++b + ++b + ++b;
```

30. Приведите примеры констант для основных встроенных типов.

4.2 Базовые вопросы

1. Особенности процесса построения приложения для .Net.
2. Что такое сборка в .Net? Перечислите основные типы файлов MS Visual Studio для построения приложения.
3. Перечислите области видимости.
4. Из чего состоит программа на языке C#? Охарактеризуйте язык C#.
5. Перечислите и охарактеризуйте встроенные типы языка C#. На какие группы делятся все типы C#?
6. Перечислите и охарактеризуйте операторы языка C#.
7. Перечислите и охарактеризуйте инструкции языка C#.
8. Приведите примеры инструкций объявлений для основных встроенных типов.
9. Перечислите и охарактеризуйте инструкции выбора (ветвления).
10. Перечислите и охарактеризуйте инструкции итерации (цикла).
11. Перечислите и охарактеризуйте инструкции безусловного перехода.
12. Перечислите и охарактеризуйте ключевые слова, используемые для обработки исключительных состояний.
13. Что такое класс и структура. Различия между классами и структурами. Из чего состоит класс или структура?
14. Статические и не статические члены классов или структур.
15. Как выполняется передача фактических параметров в методы? Ключевые слова ref и out.
16. Какой класс в .Net предоставляет математические функции. Примеры функций.
17. Какие структуры в C# используются для работы со временем?
18. Методы и свойства, которые предоставляет класс string?
19. Методы и свойства, которые предоставляет структура char?
20. Какие escape-последовательности символов доступны в C# ?
21. Какие методы, не приводящие и приводящие к исключительным состояниям, используются для преобразования строчек в числовые типы? И наоборот, числовых типов в строчки?
22. Какие типы и методы используются для ввода - вывода текстовых файлов?
23. Какие типы и методы используются для ввода - вывода двоичных файлов?

24. Что такое метод класса? Примеры переопределение методов класса.
25. Что такое конструктор класса? Как вызывается конструктор родительского класса. Примеры перегрузок конструкторов класса. Конструктор по умолчанию.
26. Что такое свойство класса?
27. Что такое индексатор класса?
28. Что такое коллекция в .Net? Привести два примера коллекции, и их основные методы и свойства.
29. Что такое инструкция using?
30. Массивы в языке C#, инициализация массивов, одномерные и многомерные массивы. Основные свойства и методы массивов.

4.3 Дополнительные вопросы

1. Из каких разделов состоит сборка?
2. Основные директивы препроцессора C#. Для чего они используются?
3. Что такое абстрактный класс?
4. Приведите и охарактеризуйте примеры нескольких типов, которые используются для обработки исключительных состояний.
5. Реализуйте метод CompareTo в следующем классе

```
class S : IComparable {  
    bool first;  
    string fio;  
    uint age;  
}
```

для сортировки коллекций или по полю fio, или по полю age.

6. Добавьте в класс

```
class S {  
    string fio;  
    uint age;  
}
```

конструктор и статическое свойство, для подсчета количества созданных объектов.

7. Что такое перегрузка операторов? Охарактеризуйте перегрузку операторов в C#.
8. Для класса

```
class coor {  
    double ltt;  
    double lng;  
}
```

Перегрузите оператор преобразования в строку.

9. Какой механизм используется для реализации полиморфизма?
10. Позднее и раннее связывание. Виртуальные функции и переопределение функций. Ключевые слова `virtual` и `override`.
11. Что такое атрибуты (`System.Attribute`)? Примеры часто используемых атрибутов.
12. Как из типа перечисление (`Enum`) получить текстовые представления всех его значений и строчку преобразовать в значение из типа перечисление?
13. Какой класс в `.Net` используется для создания окон? Какие методы в `.Net` используются для демонстрации окон оператору? Типы окон.
14. Какой класс в `.Net` используется для создания управляющих элементов окон? Каким способом управляющие элементы связываются с окном?
15. Что такое файловая система? Какие классы используются для управлением файловой системы (создание, копирование, удаление файлов и директорий)?
16. Ключевое слово `yield` и пример его использования.
17. Для чего используется класс `File`.
18. Получение случайных чисел в `.Net`.
19. Что такое интерфейс?
20. Для чего используется класс `Directory`?
21. Для чего используется класс `DirectoryInfo`?
22. Что такое автоматически реализуемые свойства (`auto-implemented properties`)? Приведите пример.
23. Что такое класс `StringBuilder`?
24. Что такое лямбда - выражение? Примеры использования.
25. Какие типы и методы используются для низкоуровневого ввода - вывода файлов?
26. Что такое регулярное выражение?
27. Привести пример и объяснить использования ключевых слов `checked`, `unchecked`.

28. Что такое динамический тип (dynamic)? Приведите отличия от типа object и типа var.
29. Примеры интерфейсов в C#.
30. Что такое класс CultureInfo и как она влияет на работу приложения?

5 МОДУЛЬ: СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ

5.1 ЛАБОРАТОРНАЯ РАБОТА: УПРАВЛЯЮЩИЕ ЭЛЕМЕНТЫ

Тема:

Оконные приложения.

Цель:

Создать приложение для демонстрации заданного управляющего элемента из списка (Button, TextBox, Label) в заданном месте, с заданным текстом

Требования раздела 1.1 остаются в силе. Поэтому задавать управляющие элементы и их положение в окне надо при помощи аргументов командной строки. Должно быть что - то вроде:

```
Programma dlya vstavki v formu knopki, nadpisi ili tekstovogo polya.  
a.exe [-?] [-x ***] [-y ***] [-w ***] -k <button/label/textbox>;  
-?   справка;  
-x   отступ объекта от левого края;  
-y   отступ объекта от верхнего края;  
-w   text na objekte ;  
-k   vibor objekta(button,label ili textbox);
```

Управляющий элемент (окна) - поверхность экрана компьютера (прямоугольник) для взаимодействия приложения с пользователем и, в отличие от окна, не самостоятельное в том смысле, что может появляться на экране только в содержащем его окне.

Задание:

- Познакомиться со возможностями .Net по динамическому созданию окон.
- Познакомиться с классами TextBox, Label, Button, Control.
- В усложненной версии приложения требуется продемонстрировать восстановления значений по умолчанию в окне TextBox. Имеется ввиду, что после закрытия окна и последующей демонстрации его оператору, тест в поле ввода должен стать исходным. При этом, новый объект окна не создается, а используется первоначально созданный.

5.2 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛоговое ОКНО OkCancel

Тема:

Создание модальных окон.

Цель:

Разработать класс для демонстрации вопроса оператору с двумя кнопками Да - Нет или Ok - Cancel и приложение для демонстрации окна этого класса.

Задание:

- Познакомиться с методами демонстрации окон.
- Познакомиться с классом Form и свойствами окна.
- Познакомиться с классами Button, Panel и возможностями привязки управляющих элементов к окну.

5.3 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛоговое ОКНО ВВОДА ДАННЫХ

Тема:

Создание модальных окон.

Цель:

Разработать класс для демонстрации вопроса оператору с несколькими полями ввода и двумя кнопками Да - Нет или Ok - Cancel и приложение для демонстрации окна этого класса.

Задание:

- Класс для демонстрации окна с несколькими полями ввода наследовать от класса из предыдущей лабораторной работы.
- Использовать шаблон словаря Dictionary<TextBox, type>, где type - некоторый тип для описание поля ввода в диалоговом окне (должен содержать по крайней мере название поля ввода, значение по умолчанию, поле объекта для введенного значения).
- Познакомиться с методами, принимающими переменное число фактических параметров.
- Познакомиться с событием Click.
- Создать метод обработчик для делегата Click.

5.4 ЛАБОРАТОРНАЯ РАБОТА: МЕНЮ В ГЛАВНОМ ОКНЕ ПРИЛОЖЕНИЯ

Тема:

Многодокументный интерфейс.

Цель:

Создать приложение с главным окном приложения, содержащее меню с элементами File, Exit, Work, About и статус - строкой. Приложение должно использовать статус - строку для отображения своего состояния. Нажатие на Exit должно закрывать приложение. Нажатие на About - приводит к появлению диалогового окна со справкой о приложении.

Задание:

- Познакомиться с классами MainMenu, MenuItem.
- Познакомиться с классами StatusStrip, ToolStripStatusLabel.
- В классе окна создать три метода обработчика для событий Click в разных элементах меню. (В этом случае отсутствует необходимость использовать в обработчике параметр object sender).

5.5 ЛАБОРАТОРНАЯ РАБОТА: ПАНЕЛЬ ИНСТРУМЕНТОВ ДОЧЕРНЕГО ОКНА С ПРЕДСТАВЛЕНИЕМ ТАБЛИЧНЫХ ДАННЫХ

Тема:

Представление табличных данных.

Цель:

К приложению из лабораторной 5.4 добавить класс дочернего окна для демонстрации содержимого CSV файла при помощи управляющего элемента DataGridView. Главное окно приложения из предыдущей лабораторной остается, а по нажатию на элемент меню Work главного окна, дочернее окно должно содержать пока не используемую панель инструментов с набором кнопок: Открыть, Перечитать, Редактировать, Добавить, удалить, Сортировать, Фильтровать, Экспортировать, Заккрыть.

Задание:

- Познакомиться с классам ToolBar, ToolBarButton (или ToolStrip).
- Создать метод обработчик для события Click на панели инструментов, требуется обрабатывать нажатия на кнопку Открыть и кнопку Заккрыть.

- Для открытия файла использовать стандартное диалоговое окно OpenFileDialog, поиск файлов начинать с каталога из которого было запущено приложение (а не с диска c:). Еще раз: стандартные диалоговые окна должны начинать показ файлов ТОЛЬКО из каталога, в котором находится исполняемая сборка.
- Создать метод обработчик для события Click на элемент меню из главного окна приложения. Открываемое дочернее окно должно изменить состояние статус строки главного окна приложения.
- Познакомиться с классом DataGridView и научиться заполнять его данными из любого CSV файла.

5.6 ЛАБОРАТОРНАЯ РАБОТА: ОКНО РЕДАКТИРОВАНИЯ ТАБЛИЦ

Тема:

Просмотр и редактирование табличных данных.

Цель:

Создать приложение с главным окном приложения 5.5 и разработанным дочерним окном для просмотра таблицы, редактирования записей из таблицы при помощи диалогового окна приложения лабораторной 5.3. Данные для таблицы должны находиться в CSV файле с одной из кодировок (866, 1251, UTF8, UTF16).

Задание:

- Продолжить освоение работы с классом DataGridView.
- Кодировку задавать двумя способами: по старому: ключом из командной строки, по новому: при помощи диалогового окна из главного меню, в которое добавить элемент Parameters (между Work и About).
- Строчки из таблицы (записи) редактировать при помощи диалогового окна.
- Стандартные окна для открытия и сохранения файлов по умолчанию должны показывать каталог из которого было запущено приложение.
- Использовать классы для кодирования символов национальных алфавитов (кириллицы) (Encoding).
- Пересмотреть особенности использования коллекций в .Net.
- Создать методы обработчики для чтения, сортировки, редактирования, экспортирования и сохранения таблицы в файле.
- При нажатии на элемент меню Work главного окна желательно открывать не более одного дочернего окна для просмотра содержания таблицы.

5.7 ЛАБОРАТОРНАЯ РАБОТА: МНОГОДОКУМЕНТНЫЙ ИНТЕРФЕЙС

Тема:

Многодокументный интерфейс.

Цель:

Создать приложение для просмотра и редактирования базы данных Поставщиков см. [13, с. 257].

Задание:

- В приложении использовать все ранее изученные классы.
- В главном меню добавить элемент меню Windows, в которое должны автоматически добавляться заголовки всех открываемых окон.

5.8 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ

В качестве модульной работы написать:

1. Оконную версию приложения построчного сравнения двух текстовых файлов (см. утилиту windiff, см. <https://support.microsoft.com/ru-ru/help/159214/how-to-use-the-windiff-exe-utility>).
2. Оконную версию утилиты для нахождения разницы в двух каталогах.
3. Оконную версию приложения Касса (выдача сдачи). В отличие от консольной версии приложение должно конвертировать тестовый файл с описанием состояния кассы в двоичный и наоборот. Для хранения состояния кассы использовать двоичный файл. Иметь возможность просматривать текущее состояние кассы. Окно приложения появляется только в случае отсутствия в командной строке ключей -txt2bin, -bin2txt и -с MMM.

Приложение для выдачи сдачи

```
a.exe [-?] [ -txt2bin | -bin2txt ] -f state.bin [-с MMM]
-?          выдача подсказки
-txt2bin    состояние кассы из стандартного ввода сконвертировать в файл state.bin
-bin2txt    состояние кассы из state.bin вывести в стандартный вывод
-с MMM      выдать сдачу в консольном режиме, не показывать окно приложения
-f state.bin файл с описанием состояния кассы.
```

4. Оконную версию приложения транслитерации имен файлов с заменой спец.символов (проход по дереву каталогов). Требуется переименовать файлы, обработка ошибок, журнал работы приложения. По ключу -demo демонстрация - не переименовывать файлы, а только вывести список, какие файлы будут переименованы и новые имена.
5. Оконную версию приложения для включения-отключения автозапуска для сменных накопителей (USB-флеш-накопителей, оптических дисков (CD-ROM, DVD-ROM)).

6. Приложение для просмотра и редактирования базы данных Поставщиков (без журналирования работы приложения), [13].
7. Приложение с двигающейся (и убегаящейся от указателя мышки) кнопки Ok. Приложение должно демонстрировать таблицу координат кнопки.
8. Оконную версию приложения для изменения кодировки текстового файла.
9. Многодокументное приложение, повторяющее интерфейс системы ограничения доступа из [31, с. 83].
10. Утилита для обнаружения на кадре полосок одинакового цвета (подготовка к обнаружению знаков дорожного движения).

Желательно, что бы приложения удовлетворяли следующим условиям:

- использовать управляющие элементы Menu + MenuItem (или MenuStrip + ToolStripMenuItem) - для предоставления команд оператора в приложении.
- использовать управляющие элементы ToolBar + ToolBarButton (или ToolStrip + ToolStripButton) - альтернативный способ предоставлять команды оператора.
- использовать управляющий элемент StatusBar (или StatusStrip) - для отображения информации о состоянии приложения и дочерних окон.
- использовать управляющий элемент DataGridView для отображения табличных данных.
- использовать мультидокументный интерфейс для открытия нескольких дочерних окон.

6 МОДУЛЬ: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММИРОВАНИЯ В .Net

6.1 ЛАБОРАТОРНАЯ РАБОТА: РЕСУРСЫ В ПРИЛОЖЕНИЯХ

Тема:

Ресурсы в приложениях.

Цель:

Создать приложение с изменяющимся рисунком в главном окне приложения.

Задание:

- Познакомиться с понятием ресурсы приложения.
- Познакомиться с классом ResXResourceWriter, создать файл ресурсов resx.
- Познакомиться с компилятором ресурсов resgen.exe.
- Познакомиться с классами ResourceManager.
- Познакомиться с понятием таймера (Timer).

6.2 ЛАБОРАТОРНАЯ РАБОТА: ПРИМИТИВЫ ГРАФИКИ

Тема:

Основы программирования графики.

Цель:

Создать приложение для чтения из файла последовательности координат пикселей экрана (одна строка - два числа) и отображения их в качестве не масштабированной кривой в главном окне приложения и нарисованным вертикальным и горизонтальным текстом в заданном месте, заданного цвета.

Задание:

- Познакомиться с событием Paint, классом PaintEventArgs, методом Invalidate.
- Познакомиться с классом Graphics, с примитивами для рисования ломанных, эллипсов и прямоугольников.
- Познакомиться с типами SolidBrush, Pen, Color.
- При заданном ключе -v - подписать номера точек ломаной.

6.3 ЛАБОРАТОРНАЯ РАБОТА: МАСШТАБИРОВАНИЕ ЛОМАННОЙ

Тема:

Основы программирования графики.

Цель:

Создать приложение для чтения из файла последовательности координат на местности (одна строка - два числа) и отображения их как масштабированную кривую в главном окне приложения, предусмотреть возможность изменения масштаба и перемещения ломаной на экране при помощи кнопок панели инструментов ToolBar.

Задание:

- Повторить использование классов `ToolBar`, `ToolBarButton` (или `ToolStrip`);
- При заданном ключе -v - подписать номера точек ломаной.
- Использовать поле `ToolBarButton.ToolTipText` для демонстрации всплывающей подсказки при попадании указателя мышки в область панели инструментов.
- Иметь два массива, первый (`coors`) массив пар вещественных чисел, должен использоваться для хранения реальных координат кривой (например градусы мировой системы координат 1984 года), второй (`pxls`) - массив пар целых чисел, который будет хранить координаты пикселей в окне для отображения кривой.
- Познакомиться с методом `Invalidate`.
- Познакомиться со способами проецирования плоскости на плоскость.
- Создать класс `mapping` с обязательным методом для отображения точек реальности в точки экрана

```
void map (double X, double Y // точки реальности
        , out int x, out int y // точки экрана
        ) {}
```

- Класс `mapping` так же должен иметь методы для изменения параметров отображения. Например:

```
public void moveNordSouth (
    int shift ///< проценты сдвига области видимости, >0 -- на север, <0 -- на юг
){}
public void moveNordSouth (
    int shift ///< проценты сдвига области видимости, >0 -- на север, <0 -- на юг
){}
public void zoom (
    int shift ///< >0 --увеличение области видимости (мелкий масштаб),
){}
```

- Метод - обработчик нажатия на кнопки панели инструментов должен изменять параметры класса `mapping`, по массиву реальных координат строить массив координат пикселей экрана и, затем, вызывать метод `Invalidate` для области окна в которой выполняется рисование.

6.4 ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ МЫШИ

Тема:

Основы программирования графики.

Цель:

Создать приложение для масштабирования кривой в главном окне приложения, предусмотреть возможность изменения масштаба и перемещения ломаной на экране при помощи кликов мышки. Предусмотреть возможность сообщать оператору попал ли клик мышки по окну в область окна рядом с ломаной.

Задание:

- Для вычисления расстояния и направления перемещения графика использовать события мышки
 - MouseDown - начать перетаскивание и отметить начальную точку для вычисления направления и расстояния перемещения;
 - MouseUp - прекратить перетаскивание;
 - MouseMove - чтобы вызвать эффект перетаскивания кривой, для каждого такого события отмечать текущую точку для вычисления направления и расстояния перемещения, пересчитывать новый массив координат точек в пикселях и вызывать метод Invalidate() для отрисовки текущего положения ломаной.
- Познакомиться с классом ContextMenuStrip.
- Познакомиться с классами Region и GraphicsPath.
- При попадании маркером мыши в область рядом с нарисованной ломаной создать контекстное меню с одним содержанием, при попадании маркером мыши далеко от ломаной, создать контекстное меню с другим содержанием.

Контекстное меню

Иначе, всплывающее меню, - скрытое меню, которое появляется при нажатии на правую клавишу (клик) мыши. Обычно оно разное в зависимости от области, где произошел клик.

6.5 ЛАБОРАТОРНАЯ РАБОТА: СКАЧИВАНИЕ ФАЙЛОВ

Тема:

Основы сетевого программирования.

Цель:

Создать приложение для скачивания файлов по заданному адресу в интернете (wget).

Задание:

- Познакомиться с классом HttpRequest.
- Познакомиться с классом HttpResponse.
- Обрабатывать исключительные состояния с понятным информированием пользователя.

6.6 ЛАБОРАТОРНАЯ РАБОТА: ЖУРНАЛИРОВАНИЕ РАБОТЫ ПРИЛОЖЕНИЯ

Тема:

Потоки команд и процессы.

Цель:

Создать библиотеку для журналирования работы приложения. Продемонстрировать её работу на примере многосредового вычисления простых чисел.

Задание:

- Познакомиться с классом Thread.
- Познакомиться с классом Monitor.
- Познакомиться с классом Interlocked.
- Познакомиться с инструкцией lock.

6.7 ЗАДАНИЯ ДЛЯ МОДУЛЬНОЙ РАБОТЫ

1. Утилита для демонстрации летающего круга, круг должен отталкиваться от границ поля.
2. Утилита для демонстрации двух различных контекстных меню, в зависимости от того, был клик выполнен внутри многоугольника (изображенного в окне) или снаружи.
3. Утилита для закрашивания двух треугольников в зависимости от их взаимного расположения:
 - треугольники не пересекаются - закрашивать одним цветом.
 - треугольники имеют одну общую сторону (или общую точку) - закрашивать разными цветами.
 - треугольники пересекаются - не закрашивать вообще.
4. Оконная вызывающая утилита для машины Тьюринга, [7]. От приложения требуется графически отображать перемещения головки машины Тьюринга по ленте.
5. Оконную вызывающую утилиту для машины Тьюринга, [7]. Приложение может не отображать перемещение головки машины Тьюринга по ленте, но должно использовать домен приложений.
6. Утилита для сбора высот местности на заданном участке.
7. Утилита для сбора температуры на заданном участке.
8. Утилита для сбора курса валют.
9. Приложение для просмотра и редактирования базы данных Поставщиков с журналированием работы приложения, [13].

7 ВОПРОСЫ ДЛЯ ЭКЗАМЕНА

Кроме приведенных ниже вопросов, на экзамене будут задаваться вопросы из раздела Вопросы для Дифференциального Зачета (см. 4).

7.1 Создание Оконных Приложений

1. Какие свойства класса Form используются для создания многодокументного интерфейса?
2. Напишите метод для выбора и активизации уже открытого дочернего окна.
3. Какой класс используется для создания окон?
4. Какие методы используются для демонстрации окон?
5. Какие свойства связаны с кнопками управления окном?
6. Какие свойства и перечисления используются при создании диалоговых окон?
7. Перечислите и охарактеризуйте стандартные диалоговые окна.
8. Как event delegate используется при разработке оконного интерфейса?
9. Какое событие нужно использовать перед закрытием окна? Приведите пример обработчика.
10. Какие классы, свойства и методы используются для загрузки - сохранения XML документа?
11. Какие классы, свойства и методы используются для извлечения узлов из XML документа?
12. Какие классы, свойства и методы используются для добавления узла в другой XML узел?
13. Какие классы, свойства и методы используются для добавления атрибута к XML узлу?
14. Назовите несколько управляющих элементов. Как управляющий элемент добавляется к окну?
15. Какое ключевое слово предназначено для создания метода с переменным числом параметров? Приведите пример.
16. Назовите управляющие элементы, предназначенные для автоматического размещения содержащихся в них управляющих элементов.
17. Приведите пример использования шаблона для создания словаря.
18. Для чего используется свойство Padding?
19. Для чего можно использовать управляющий элемент Panel?

20. Какое свойство и перечисление используется для размещения управляющих элементов?
21. Какие классы используются для просмотра табличных данных?
22. Приведите пример настройки управляющего элемента для просмотра табличных данных.
23. Приведите пример добавления в управляющий элемент для просмотра табличных данных трех колонок.
24. Какая коллекция и метод используется для добавления строчек в таблицу?
25. Как гарантировать видимость заданной ячейки в управляющем элементе для просмотра табличных данных DataGridView ?
26. Какие свойства, классы, методы и события применяются для использования главного меню в окне?
27. Какие классы и свойства применяются для использования панели инструментов в окне?
28. Приведите пример информационного управляющего элемента, отличного от Label.
29. При помощи какого ключевого слова гарантируется освобождение захваченных программных ресурсов (имеются ввиду файлы, нити (потoki команд), сокеты и соединения с базой данных)? Приведите пример.
30. Какой класс используется для создания (или чтения, или вывода) текстов в заданной кодировке? Пример создания и использования кодировки 1251.

7.2 Дополнительные Возможности Программирования В .Net

1. Какие средства существуют в .Net для определения типа во время выполнения?
2. Какая утилита используется для генерации ресурсов?
3. Какой класс используется для генерации ресурсов?
4. При помощи какого класса осуществляется доступ к ресурсам?
5. Какие методы используются для преобразования целых чисел и сточек (string) в цвет (Color)?
6. Назовите и охарактеризуйте основные классы, которые используются для рисования в окнах .Net.
7. Создайте красный карандаш при помощи словесного описания цвета.
8. Создайте полупрозрачную синюю кисточку.

9. Какой метод панели следует вызывать, что бы перерисовать панель?
10. Приведите пример обработчика события Paint для рисования ломаной.
11. Как создаются всплывающие подсказки на элементах управления (например, на ToolBarButton)?
12. Каким методом измеряют размер текста в пикселях? Приведите пример.
13. Каким методом выводят текст в области окна (без объекта класса Label)? Какие дополнительные классы для этого требуются?
14. Какой класс используется для перемещения отображаемых в окне объектов в вертикальном направлении? Приведите пример инициализации.
15. Какой класс используется для перемещения отображаемых в окне объектов в горизонтальном направлении? Приведите пример инициализации.
16. В каком классе содержатся обычные математические функции? Приведите пример.
17. Какой класс используется для низкоуровневого (побайтного) ввода - вывода? Приведите пример.
18. Какой класс используется для выполнения Http - запросов? Приведите пример скачивания файла.
19. Как можно узнать положение координат указателя мышки 1) на экране, 2) в окне?
20. Какие классы, свойства и методы используются для создания контекстного меню?
21. Какие классы, свойства и методы используются для изменения (добавления, удаления элементов) контекстного меню?
22. Приведите пример создания контекстного меню.
23. Какими средствами можно проверить принадлежит ли заданная точка сложной геометрической фигуре?
24. Что такое процесс и какой класс используется для управления выполняемыми на компьютере процессами и запуска новых (дочерних) процессов?
25. Что такое код завершения приложения? Каким способом приложение может задать код завершения (или код возврата)?
26. Что такое поток команд (нить)? Какой класс используется для запуска и управления нитями?
27. Какой класс используется для организации исключительного доступа нитей к, например, целым переменным?
28. Что такое блок синхронизации и какие классы и методы предназначены для работы с ними.
29. Для чего используется оператор lock? Приведите пример его применения.

30. Как организовать исключительный доступ нитей к статическим переменным?

Что бы организовать исключительный доступ нитей к статическим переменным в сочетании с методами `Monitor.Enter(object o) / Monitor.Exit(object o)` или оператором `lock` необходимо использовать операцию `typeof`.
Пример:

8 ТЕМАТИКА КУРСОВЫХ ПРОЕКТОВ

При оценки курсовых проектов поощряются работы, выполненные с учетом ранее созданных проектов таких как

- Библиотеки обработки аргументов командной строки, см. [19].
- Журналирования приложений, см. [25], [20].
- Библиотеки геометрических вычислений на эллипсоидах см. [26].
- либо отдокументированные при помощи системы Doxygen, см. [16].

Замечание

Отчеты к курсовым работам, излагаемым в этом разделе должны удовлетворять требованиям к оформлению отчетов раздела 9 и дополнительно содержать две UML - диаграммы. Различные примеры UML - диаграмм, относящихся к потокам данных, можно посмотреть в работе [27, с. 4] или [27, с. 7]. Диаграммы строились при помощи свободно-распространяемой утилиты `graphviz` (см. [17]). Примеры UML - диаграмм классов, которые описывающая структуру системы, демонстрирующая классы системы, их атрибуты, методы и зависимости между классами, можно посмотреть в пояснительной записке проекта [19] (файл `args.2.10.pdf` на стр.26). Эта диаграмма был сгенерирована из кода приложения при помощи утилиты `doxygen` (см. [16]).

Кроме того, результаты использования системы Doxygen можно посмотреть в методических рекомендациях к лабораторной работе Масштабированная ломаная см. 13.3.0.1 .

Список тем курсовых:

1. Утилита для асинхронного чтения больших файлов - заполнения управляющего элемента для показа таблиц (`DataGridView`).
2. Игра 'Теннис' для двух игроков.
3. Игра 'Пятнашки'.

4. Класс для работы с GPX - файлами, совместимый с Библиотекой вычислений на эллипсоидах elGeo, см. [26].
5. Утилита для слияния - разделения GPX - файлов.
6. Утилита для нахождения остановок на треке транспортного средства.
7. Утилита для нахождения углов оцифрованных карт Системы Координат 1942 года.
8. Приложение для отображения треков транспортного средства на картах Системы Координат 1942 года, [10]).
9. Утилита для отображения треков транспортного средства на картах открытых сервисов, (см. [37], [6]).
10. Усложненная версия последних двух приложений с демонстрацией кадров из записей видеорегистратора, [14].
11. Утилита для нахождения знака 'Стоп' на кадре видеорегистратора.
12. Утилита для обнаружение линии горизонта на кадрах видеорегистратора.
13. Приложение для получения и воспроизведения потока радиотрасляции и демонстрации текущего времени.
14. Оконная утилита для принудительной сортировки (и транслитерацией) музыкальных файлов в дереве каталога.
15. Утилита для сбора высот для заданной точки, линии или прямоугольника из открытых географических сервисов, [38].
16. Утилита для сбора, хранение и отображения курсов валют (см. [21]).
17. Утилита для сбора температуры, аналогично утилите для сбора валют.
18. Утилита для выгрузки описания географических объектов из файлов проекта OSM в CSV - файлы.
19. Утилита для выгрузка координат географических объектов из файлов проекта OSM в CSV - файлы.
20. Приложение для вращения трехмерной фигурки.
21. Документирование модульной работы при помощи кроссплатформенной системы документирования исходных текстов программ - Doxygen (<http://www.doxygen.nl/>), введение можно посмотреть в [16].
22. Выполнение модульной работы при помощи системы контроля версий Git на хостинге <https://github.com/>.
23. Класс для посчета площади сфероидического многоугольника и приложение для тестирования класса.

24. Приложение для обучения детей письму. БД должна содержать таблицы Слов, Картинок (каждому слову - несколько различных картинок), Букв (с картинками изображений буквы, из которых ребенок будет составлять слово) , Сообщений Приложения (как реакция приложения на действия ребенка), Попытки (историю попыток детей).
25. Приложение распределения времени лекций, лабораторных и самостоятельной работы студента для построения рабочей учебной программы.
26. Приложение для демонстрации нарушения Принципа подстановки Лисков. (См. [24, 23]), которое возникает при попытках наследования вещественных чисел от целых или наоборот.
27. Утилита для стрессового тестирования файловой системы.
28. Многодокументное приложение по описанию системы ограничения доступа из [31], вместо БД использовать текстовые CSV файлы.
29. Windows - версия программного комплекса для билатерального тестирования индивидуально - типологических свойств высшей нервной деятельности человека [33].
30. Утилита для выгрузки информации из файлов MS Excel.
31. Прохождение отборочного турнира любого программистского конкурса на <http://codeforces.ru>.

9 ТРЕБОВАНИЯ К СДАВАЕМЫМ РАБОТАМ

Сдаваемая работа представляет собой файл, созданный одним из популярных архиваторов (zip, rar, 7z). Имя архива содержит номер лабораторной (или модуля) и её версию (попытку сдать). Например,

- lab3.5_2.zip - лабораторная 3.5, вторая версия;
- mod1_1.rar - модуль 1, первая версия;
- coursework.7z - курсовая работа, первая версия.

Номер первой версии можно пропускать.

Архив содержит:

- имена всех файлов только из букв латинского алфавита, цифр точек и символов подчеркивания, а значит не содержат символов кириллицы и пробелов.
- Файл readme1st.txt, который содержит историю разработки: дата, версия и что изменено в версии после отправки лабораторной на доработку.
- исходные тексты для построения приложения,

- в случае если приложение использует динамически подгружаемые библиотеки третьей стороны и/или файлы ресурсов, то архив должен содержать сами библиотеки, ресурсы и файл local.cmd приблизительно следующего содержания:

```
SET R=/r:System.Data.SQLite.dll /r:args.dll  
SET RES=/resource:tiles.resources
```

значение макроса R будет 'R=/r:System.Data.SQLite.dll /r:args.dll' содержать ключи компилятора Микрософт для использования двух библиотек, значение макроса RES - ключ для подключения ресурса tiles.resources.

- тестовые данные,
- в случае использования аргументов командной строки файлы командного интерпретатора cmd для запуска приложения с различными ключами,
- отчет, либо в формате pdf, либо в формате doc от MS Word 2003, либо в формате rtf. Объем работы не менее 10 листов.

Отчеты к работам оформлять согласно требований [11]. В основном это значит, что текст надо набирать шрифтом Times New Roman, 12 pt, интервал между строчками - 1, форматирование по ширине, отступы: 2 см слева, 1.5 - справа, сверху и снизу страницы, абзац - 1 см. Нумерация страниц - внизу по центру. Далее, отчеты должны содержать следующие части:

- Титульный лист;
- Содержание;
- Постановка задачи;
- Теоретическая часть (описание предметной области, краткие теоретические сведения, необходимые для выполнения задачи, описание инструментов программирования и т.д.);
- Ответы на вопросы, соответствующего модуля. Ответы на вопросы должны содержать номер вопроса, его формулировку и файл с фотографией текста, написанного от руки;
- Описание алгоритма программы;
- Описание использования приложения;
- Тесты для проверки работоспособности;
- Выводы;
- Приложение;
- Список используемой литературы.

Титульная страница должна содержать название министерства, университета, института, кафедры, тему работы, исполнителя, преподавателя, город, год.

Список литературы оформлять согласно требований [12].

В отчетах к лабораторным некоторые части разрешается опускать. Нельзя опускать разделы Теоретическая часть и Ответы на вопросы. Теоретическая часть для лабораторных пишется в накопительном виде, то есть, в отчете к текущей лабораторной описывается только то, что надо для её разработки и не пишется то, что было описано в отчетах к предыдущим лабораторным. Теоретическая часть для модуля и курсовой должна содержать все вопросы из текущего модуля/курсовой, относящиеся к приложению.

Разделы Ответы на вопросы и Теоретическая часть должны содержать объем информации, достаточный для немедленного применения, в ответы можно добавлять информацию, непосредственно связанную с вопросом. Например, ответ на вопрос: 'Целые типы', должен содержать список всех целых типов char, unsigned char, int и так далее. Примеры описания с инициализацией и без. Диапазоны допустимых значений. Разницу между знаковыми и беззнаковыми типами. Операции, которые могут быть применимы к данным типам. Правила преобразования один в другой. Дополнительно, можно рассказать про спецификаторы места вывода для методов Format и WriteLine.

10 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЯ C#

10.1 ВОПРОСЫ КОМПИЛЯЦИИ ПРИЛОЖЕНИЙ В СРЕДЕ MONO ДЛЯ *NIX

Следующий действия были проверены в операционной системе Ubuntu 18.x

- Установка компилятора:

```
$ sudo apt install mono-mcs
```

- Установка среды выполнения:

```
$ sudo apt install mono-runtime
```

- компиляция:

```
$ mcs /out:a.exe sourcecode.cs
```

- выполнение:

```
$ mono a.exe
```

10.2 ЛАБОРАТОРНАЯ РАБОТА: ВЫЧИСЛЕНИЕ ПРОСТЫХ ЧИСЕЛ

При разработке приложения для лабораторной работы [2.1](#) будет полезна следующая информация:

- тест программы должен храниться в файле с расширением .cs;
- компиляция из командной строки MS Visual Studio выполняется следующей командой:

```
csc /out:a.exe *.cs
```

- желательно подключить системное пространство имен System (именованную область видимости) для просмотра по умолчанию при помощи директивы

```
using System;
```

Эта директива дает возможность использовать различные простые типы, математические функции (Math.Sqrt - извлечение корня) и стандартные файлы ввода - вывода без дополнительного квалифицирования (то есть, для описания целой переменной можно писать 'int a', а не 'System.Int32 a');

- программа состоит из классов, хотя бы один из которых содержит статический метод Main (точка входа в приложение)

```
class primeNumber {
    static int Main(){
        // тело метода тут
    }
}
```

- По строчное чтение текста из стандартного ввода выполняется следующим образом:

```
string a = Console.ReadLine(); // сокращенная форма
string b = Console.In.ReadLine(); // полная форма
```

- Запись в стандартный вывод (и стандартный вывод ошибок) выполняется так:

```
Console.WriteLine("Hello, ");           // сокращенная форма
Console.Out.WriteLine("world ");        // полная форма
Console.Error.WriteLine("world ");      // полная форма
Console.WriteLine("this is {0} {1} years old", "Tom", 123);
                                           // вывод с спецификаторами места выводя
```

- конвертация числа из строки в двоичный вид:

```
int N = Int32.Parse("157");
```

- составной оператор - { }; оператор ветвления - if; операторы циклов - for и while; оператор-выражение (например, a++;) после окончания курсов [29] и [30] не должны представлять особых трудностей и интуитивно понятны.

10.3 ЛАБОРАТОРНАЯ РАБОТА: РЕШЕТО ЭРАТОСФЕНА

При разработке приложения для лабораторной работы 2.2 будет полезна следующая информация:

- Одномерный массив объявляется следующим образом:

```
int [] a = null;
```

- Память под массив захватывается следующим образом:

```
a = new int[20];
```

Освобождать память не требуется, этим занимается сборщик мусора. Свойство Length возвращает длину массива.

- Изменение длины массива с 20 до 4.

```
Array.Resize(ref a, 4);
```

Статический метод `Array.Resize` создает новый массив заданного размера, копирует в него содержимое старого массива и возвращает ссылку на новый массив. Ключевое слово `ref` указывает, что требуется вернуть ссылку на новую память.

- Структура `DateTime` предоставляет дату и время от нуля часов 1 января 0001 года до 24 часа 31 декабря 9999 года. Свойство `DateTime.Now` возвращает время и дату текущего момента.
- Операция разности между двумя структурами `DateTime` возвращает структуру `TimeSpan`. Последняя предназначена для хранения интервалов времени. В структуре `DateTime` есть свойство (`DateTime.DayOfWeek`) предоставляющее названия дня недели (например, понедельник), а в `TimeSpan` таких свойств нет.
- Сравнивать время работы двух приложений имеет смысл если они работали дольше чем по секунде.

10.4 ЛАБОРАТОРНАЯ РАБОТА: ПЕРЕДАЧА ФАКТИЧЕСКИХ ПАРАМЕТРОВ В МЕТОДЫ

Выполнение лабораторной работы 2.3 заключается в том, что в методе `Main` для каждой из трех групп типов (значимые, ссылочные и строки) объявить четыре переменных, присвоить им значение (например, для целых - 1, 2, 3, 4) и передать двумя способами (без помощи ключевого слова `ref` или `out` и с ним) в статический метод `f`, в котором двумя способами (без использования конструктора и с ним) изменить значение. Затем проверить какое значение будет содержать каждая из переменных после возвращения из метода `f`. См.

```
---- File:./cs/labs/mod1lab3/p.cs
```

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 1, b = 2, c = 3, d = 4;
            Console.WriteLine("a/b/c/d before f : {0}/{1}/{2}/{3}", a, b, c, d);
            f(a, ref b, c, ref d);
            Console.WriteLine("a/b/c/d after f : {0}/{1}/{2}/{3}", a, b, c, d);
        }

        static void f ( int p1, ref int p2, int p3, ref int p4){
            p1 = 10;
            p2 = 10;
```

```

        p3 = new int();
        p3 = 10;
        p4 = new int();
        p4 = 10;
        Console.WriteLine("p1/p2/p3/p4 inside f : {0}/{1}/{2}/{3}", p1, p2, p3, p4);
    }
}
}

```

----- End Of File:./cs/labs/mod1lab3/p.cs

Далее, вне области видимости класса Program, создать свою структуру:

```

struct ss {
    public    // в отличие от C++ члены структур по умолчанию закрыты
    int p;
}

```

Перегрузить статический метод f для изменения переменных типа ss

```

static void f ( ss p1, ref ss p2, ss p3, ss int p4){
    p1.p = 10;
    p2.p = 10;
    p3 = new ss();
    p3.p = 10;
    p4 = new ss();
    p4.p = 10;
    Console.WriteLine("p1/p2/p3/p4 inside f : {0}/{1}/{2}/{3}",
        p1.p, p2.p, p3.p, p4.p);
}

```

И повторить выполненные действия для четырех переменных типа ss:

```

ss sa, sb, sc, sd;    // память под структуры уже получена
sa.p = 1;             // и был использован конструктор по умолчанию
sb.p = 2;
sc.p = 3;
sd.p = 4;

```

В обоих случаях приложения должно выдать что то вроде следующего:

```

a/b/c/d before f : 1/2/3/4
p1/p2/p3/p4 inside f : 10/10/10/10
a/b/c/d after f : 1/10/3/10

```

По результату работы видно, что после применения метода f, в вызывающем коде (области видимости метода Main) значения поменяли только переменные, которые передавались при помощи ключевого слова ref (то есть по ссылке) независимо от использования конструктора структуры или целого. Это означает, что для фактических параметров значимого типа в метод передаются значение, а, собственно, сам метод работает с копией фактического параметра, а не непосредственно с ним.

Далее, создать свой класс:

```
class cc {
    public
    int p;
}
```

При объявлении переменных код потребуется поменять:

```
cc a = new cc(), b = new cc(), c = new cc(), d = new cc();
a.p = 1;      // для ссылочных типов требуется явно применить конструктор
b.p = 2;      // для захвата памяти
c.p = 3;
d.p = 4;
```

Результат приложения будет вроде следующего:

```
a/b/c/d before f : 1/2/3/4
p1/p2/p3/p4 inside f : 10/10/10/10
a/b/c/d after f : 10/10/3/10
```

Значение переменной *c* не изменилось после применения метода *f*, так как в метод была передана копия ссылки на объект, после захвата памяти в методе *f* старая ссылка, естественно, была потеряна, и внутри метода менялось значение нового объекта, который был потерян при выходе из метода *f*. В случае переменной *d* в метод *f* передалась ссылка на ссылку на объект ;)).

Аналогичные действия повторить еще для массивов и строчек и объяснить результаты. Надо помнить, что объекты класса *string* нельзя изменять.

Обязательно попробовать расположить статический и не статический методы в, например, классе *cc* и применить их.

10.5 ЛАБОРАТОРНАЯ РАБОТА: ОБРАБОТКА СТРОЧЕК

При выполнении лабораторной работы 2.4 требуется на информационном сервисе для разработчиков программного обеспечения MSDN ([3]) просмотреть описание классов *Char* и *String*. Выбрать шесть методов для обработки строчек. В их число обязательно должен входить метод *Split* - для разделения строки на массив подстрок и методе *Join* - для соединения массива строчек в одну строку. Кроме этого, выяснить как при помощи класса *Encode* можно явно задавать кодировку читаемого или выводимого файла.

Замечание

Считается, что файлы стандартного ввода - вывода имеют кодировку 866 страницы (операционной системы MS DOS). Это означает, что стандартным приложением блокнот (*notepad.exe*) не удастся увидеть слово 'привет', выведенное в стандартный вывод следующим образом:

```
Console.WriteLine("привет");
...
c:/tmp>a.exe >1.txt
...
c:/tmp>notepad 1.txt
```

Далее, поскольку текст в классах Char и String внутри приложения хранится в кодировке UTF-16, то следует помнить, что при выводе при помощи метода Console.WriteLine() происходит изменение кодировки из UTF-16 в 866ю, а при вводе через метод Console.ReadLine() - из 866 в UTF-16.

Аргументы командной строки передаются в приложение в массиве args точки входа в приложение (статический метод Main)

```
static void Main(string[] args)    {
    //код здесь
}
```

причем пробелы используются для деления всей строки на слова, которые попадают в элементы упомянутого массива. Символы двойной или одинарной кавычки можно использовать, для того чтобы включать пробелы в элементы массив args.

Приложение должно получить один аргумент командной строки и вывести в стандартный вывод результат применения выбранных методов. Например,

```
с:/tmp>a.exe 1привет:ВОЛКУЗ
длина:  15 символов
цифры:   13
буквы:  приветВОЛКУ
в верхнем регистре:  1ПРИВЕТ:ВОЛКУ:3
в нижнем регистре:   1привет:волку:3
деление символом ':' :   '1привет', 'ВОЛКУ', '3'
пробелов:  0
с:/tmp>
```

Примеры как явно задать кодировку файла можно найти в разделе [15.5](#) .

10.6 ЛАБОРАТОРНАЯ РАБОТА: ВЫБОР ДВУХ СТОЛБЦОВ (ШИРОТА И ДОЛГОТА) ИЗ ТЕКСТОВОГО ФАЙЛА

Смотри [2.5](#) . Примерная подсказка для этого приложения может иметь следующий вид:

Приложение для выбора широты и долготы из текстового файла.

Использование:

a.exe [-?] [-help] [-v] -l_{tt} N₁ -l_{ng} N₂ [-sep SEP] [-8]

где

```
-help      : выдача текущего экрана
-?         : выдача текущего экрана
-v         : выдача ошибок в файле и времени работы приложения
-sep SEP   : задать разделитель. Умолчание - символ пробела
-ltt N1    : N1 номер колонки с широтой
-ltt N2    : N2 номер колонки с долготой
-8         : выводить 8 знаков после запятой, иначе - 6.
```

После выдачи подсказки рекомендуется прекращать работу приложения с кодом возврата отличным от 0. То есть, работа метода Main должен закончиться оператором:

```
Return 1;
```

либо оператором:

```
Environment.Exit(1);
```

Вторая версия приводит к немедленному прекращению работы приложения, даже если выполняется в методе, отличном от Main.

Для ввода всех перечисленных ключей рекомендуется завести следующий набор переменных:

```
bool  vFlag    = false;
int   lttNo    = -1;
int   lngNo    = -1;
char  sep      = ' ';
bool  oFlag    = false;
int   lineNo   = 0; // номер текущей строки
int   errs     = 0; // кол-во ошибок при анализе файла.
```

Обработку ключей командной строки можно выполнять следующим образом:

```
for ( int i = 0; i < args.Length; i++){
    if ( args[i] == "-?" ) {
        // выдать подсказку
        Environment.Exit(1);
    }
    else if (args[i].ToLower() == "-help") {
        // выдать подсказку
        Environment.Exit(1);
    }
    else if (args[i].ToLower() == "-v")
        vFlag = true;
    else if (args[i].ToLower() == "-8")
        oFlag = true;
    else if (args[i].ToLower() == "-sep")
        sep = setChar( ++i, args, "разделитель");
    else if (args[i].ToLower() == "-ltt")
        lttNo = setInt( ++i, args, "широты");
    else if (args[i].ToLower() == "-lng")
        lngNo = setInt( ++i, args, "долготы");
}
if ( lttNo < 0 ) {
    // не задан столбец для ширины
    Environment.Exit(2);
}
if ( lngNo < 0 ) {
    // не задан столбец для долготы
    Environment.Exit(2);
}
if (lttNo == lngNo) {
    // столбцы должны быть разные
    Environment.Exit(2);
}
```

Пример метода для получения параметра аргумента командной строки. Метод может использоваться для ключей -sep (переделать под char), -ltt и -lng.

```
static int setInt (int i, string [] args, string par) {
    int ret = 0;
    if (i < args.Length){
        if (int.TryParse (args[i], out ret))
            ; // из i-й строчки прочитано целое
        else {
            // значение для параметра par не правильного типа
            Environment.Exit(2);
        }
    }
    else {
        // не задано значение для параметра par
        Environment.Exit(2);
    }
    return ret;
}
```

Пример метода для получения следующей строки данных из файла:

```
static string [] getRec (string line, char sep= ' '){
    string [] flds=null;
    char [] seps;
    if (line != null) {
        int lastP = line.IndexOf('#');
        if (lastP >= 0) // удалили комментарий
            line = line.Substring(0, lastP);
        line = line.Trim(); // отсеки лидирующие и завершающие пробелы
        if (sep == ' ') { // если разделитель - пробел добавить
            // табуляцию
            seps = new char[2];
            seps[0] = ' ';
            seps[1] = '\t';
            flds = line.Split // несколько пробелов трактуются как один
                (seps, StringSplitOptions.RemoveEmptyEntries);
        }
        else { // если запятые или точка с запятой, то
            seps = new char[1]; // каждый символ разделителя создает отдельное поле
            seps[0] = sep;
            flds = line.Split
                (seps);
        }
    }
    return flds;
}
```

Функцию в неизменном виде можно использовать для разбора CSV файлов.

Пример цикла для разбора текстового файла и выбора значений из двух столбцов при помощи приведенного метода getRec:

```

int errors = 0;
int lineno = 0;
double aa = 0.0;
double bb = 0.0;
string s;
string [] r;
string fld="no";
r = getRec (s);          // второй параметр - использует значение по умолчанию - пробел
try {
    if (r.Length > 0) {    // если строка не пустая
        fld = "first";    // то читаем данные
        aa = double.Parse(r[1]);
        fld = "second";
        bb = double.Parse(r[2]);
        // в спецификаторах места вывода задана точность
        Console.WriteLine ("{0:0.000000} {1:0.000000}",aa,bb);
    }
}
catch (Exception e){ // ловим все исключения
    if (vFlag) {
        Console.Error.WriteLine (
"line/field: exception: {0}/{1}: '{2}'"
        , lineno
        , fld
        , e.Message
        );
        errors++;
    }
}
}
Console.Error.WriteLine ("there was {0} errors", errors);

```

Каждый блок try должен завершаться либо блоками catch, либо блоком finally, либо обеими. Блоки catch желательно упорядочить по операции наследования. Дети должны идти выше родителей. Так как класс Exception является родителем двух классов для обработки исключительных ситуаций, которые будут часто встречаться при использовании этого кода, то блоки catch для FormatException и IndexOutOfRangeException надо располагать выше:

```

try {
    if (r.Length > 0) {    // если строка не пустая
        aa = double.Parse(r[1]);
        bb = double.Parse(r[2]);
    }
}
catch (FormatException e){
    // в строчке не содержалось плавающее число
}
catch (IndexOutOfRangeException e){
    // вышли за пределы массива
}
catch (Exception e){
    // ловим все остальные исключения
}

```


Некоторые наследники класса `Exception` содержат более подробную информацию об ошибке, чем родитель, но оба рассмотренных выше и `FormatException`, и `IndexOutOfRangeException` содержат только свойства родителя (`Message`, `Source`, `StackTrace`, `TargetSite`, `HResult`).

11 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ПРОГРАММИРОВАНИЕ НА ПЛАТФОРМЕ .Net FRAMEWORK

11.1 ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ ДИНАМИЧЕСКИ ПОДКЛЮЧАЕМЫХ БИБЛИОТЕК

При разработке простой версии приложения для лабораторной работы 3.1 будет полезна следующая информация:

- для подключения новой сборки, оформленной в виде динамически подключаемой библиотеки при компиляции через командную строку требуется использовать ключ `/r`.
- для подключения к консольному приложению библиотеки .Net содержащую типы для создания оконных приложений используйте командную строку:

```
csc /r:System.Windows.Forms.dll /out:a.exe *.cs
```

При компиляции с этим ключом можно использовать типы из сборки, например, `Form` (класс используется для создания окон на дисплее) с указанием полного названия. Полное название имеет вид: `System.Windows.Forms.Form`, где первые три слова определяют пространство имен (или именованную область видимости) и последнее `Form` - название класса.

- для использования короткой формы имени типов из сборки в код программы необходимо добавить директиву

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

После этого класс `Form` (для создания окон), класс `MessageBox` можно использовать в полной и сокращенной формах.

- класс `MessageBox` содержит несколько перегруженных статических методов `Show` для демонстрации различных видов диалоговых окон (с определением заголовка окна, определением текста в окне, определением нескольких кнопок, определением иконок). Кроме того, методы возвращают кнопку, нажатую оператором. при помощи перечисления `DialogResult`.
- для корректной работы приложения перед методом `Main` требуется добавить `STAThreadAttribute` атрибут:

```
[STAThread]
static void Main()
{
}
}
```

- кроме того, можно использовать класс `Chart` для отображения некоторых кривых в окне типа `Form`. Раздел 15.10 содержит пример приложения с графиками синуса и косинуса.

Для выполнения более сложной версии предлагается использовать библиотеку для разбора аргументов командной строки [19]. Кроме того, в разделах 10.5 и 16.1.4 еще раз подробно обсуждаются сами аргументы командной строки. Компиляция приложения, как и в случае системной библиотеки, выполняется следующим образом:

```
csc /r:args.dll /out:a.exe *.cs
```

Область видимости подключается:

```
using Args;
```

В библиотеке доступны несколько типов: `ArgFlg` - для ввода логических значений, `ArgStr` - для ввода строчек, `ArgChar` - для ввода символов, `ArgInt` - для ввода целых, `ArgFloat` - для ввода вещественных. Эти же типы используются для формирования страницы подсказки приложения в методе `Arg.mkVHelp`. Каждый класс имеет метод для проверки и ввода значения - `check`. Пример использования:

```
---- File:./cs/args/p.cs
```

```
#pragma warning disable 642
// прама отключает предупреждение компилятора о пустом операторе в if
using System;
using System.IO;
using Args;

class af
{
    static
    void Main(string[] args)    {
        ArgFloat    // ключ плавающего типа
        perCent     = new ArgFloatMM(0.05, "p", "percent", "percent for something", "PPP");
        ArgStr       // ключ типа строка
        flNm        = new ArgStr("somefile.dat", "f", "file", "data file", "FLNM");
        ArgFlg       //
        vFlag       = new ArgFlg(false, "v", "verbose", "to show additional information");
        ArgFlg       //
        hFlag       = new ArgFlg(false, "?", "help", "to show usage page");

        for (int i = 0; i<args.Length; i++){

            if (hFlag.check(ref i, args))    {
                Arg.mkVHelp("to test command line arguments" // подсказка по утилите
                    , ""
                    , vFlag

```

```

        , Arg.mkArgs(
            hFlag
            ,vFlag
            ,perCent
            ,flNm
        )
    );
    Environment.Exit(1);

}
else if (vFlag.check(ref i, args))
    ; // без прагмы тут будет предупреждение о возможной ошибке
else if (perCent.check(ref i, args))
    ;
else if (flNm.check(ref i, args))
    ;
}
Console.WriteLine("Вы ввели: {0}/{1}/{2}/{3}"
    ,(bool) hFlag
    ,(bool) vFlag
    ,(double) perCent
    ,(string) flNm
);
}
}

```

---- End Of File:./cs/args/p.cs

С ключом подсказки '-?' приложение выведет следующую строку:

```

to test command line arguments
usage:
  p [-?] [-v] [-p PPP] [-f FLNM]
options:
  -?           : to show usage page: True
  -v           : to show additional information: False
  -p PPP       : percent for something: 0.05
  -f FLNM      : data file: somefile.dat

```

Для компиляции собственных динамически подключаемых библиотек используется ключ /t :

```
csc /t:library /out:l.dll *.cs
```

При этом, желательно, чтобы пространство имен в библиотеке (именованная область видимости) совпадало с именем файла. Хотя бы один класс в библиотеке должен иметь модификатор доступа public:

```

namespace l {
    public
    class c{
    }
}

```

Тогда он (класс c) будет доступен в коде приложения построенного следующим образом:

```
csc /r:l.dll *.cs
```

11.2 ЛАБОРАТОРНАЯ РАБОТА: СОРТИРОВКИ И КОЛЛЕКЦИИ

Приложения для лабораторной работы 3.2 с точки зрения обработки текстового файла выполняет те же самые действия, что и приложение из 2.5, причем очевидно, что чтение двух полей таблицы не является принципиальным. Чтение любого конечного числа полей в записи требует очевидные изменения в коде, который использует метод `getRec`. Собственно, сам метод `getRec`, как и `setInt` можно перенести в текущее приложение. Главное отличие состоит в том, что для сортировки данных требуется все данные прочитать в оперативную память (например, в массив).

Примерная подсказка для этого приложения может иметь следующий вид:

Приложение для сортировки данных (фамилия и возраст) по студентам.

Использование:

`a.exe [-?] [-help] [-v] [-s NNN] [-sep SEP] -f FLNM`

где

`-help` : выдача текущего экрана
`-?` : выдача текущего экрана
`-v` : выдача ошибок в файле и времени работы приложения
`-sep SEP` : задать разделитель. Умолчание - символ пробела
`-s NNN` : NNN номер поля для сортировки (первое или второе)
`-f FLNM` : FLNM имя CSV файла для чтения

Напомним, что квадратные скобки вокруг аргумента командной строки означают не обязательность аргумента, таким образом аргумент `-f FLNM` является обязательным. Приложение не должно работать без этого аргумента.

Далее, требование сортировать файл не предполагает разработку метода для сортировки, а предполагает использование возможностей `.Net`, предназначенных для этой цели. А они (эти возможности для сортировок и чтения файлов) предлагаются в виде интерфейсов (то есть, наборов методов и свойств без реализации).

Во-первых, будет использоваться класс `StreamReader`, предназначенный для чтения текстовых файлов и требующий явное освобождение ресурсов, связанных с чтением файла. В терминах языка Си - файл требуется открыть, прочитать и закрыть. Открытие файла выполняется в конструкторе, чтение строчки (как и в случае стандартного ввода) - методом `ReadLine`, закрытие - привычным методом `Close` или его аналогом, принятым в `.Net` - `Dispose`. Класс `StreamReader` имеет реализованный метод `Dispose`, так как он наследовался от интерфейса `IDisposable`. Таким образом, код для чтения будет выглядеть следующим образом:

```
string flNm; // переменная для имени файла,
              // которое надо получить в результате обработки
...          // аргументов командной строки
string s;
string []flds;
StreamReader sr = new StreamReader(flNm);
for (lineno=1 ; (s = sr.ReadLine())!=null; lineno++){
    flds = getRec(s);
    ...
}
```

```
    }
    sr.Dispose();
```

Во-вторых, для использования системного метода сортировки, необходимо создать свой класс - наследник интерфейса-шаблона `Comparable <тип>`:

```
using System.Collections.Generic;
...
class X : Comparable <X> {
    public string nm;    // имя студента
    public int    age;   // возраст студента
    static public int sortOrder; // статическое поле, чтобы задать
                                // порядок сортировки

    public // обязательный для реализации метод для сравнения
           // текущего и другого объекта
    int CompareTo (X other)  {
        if (sortOrder==1)
            return nm.CompareTo(other.nm);
        else
            return age.CompareTo(other.age);
    }

    // конструктор
    public X(string s, string i){
        this.s = s;
        this.i = int.Parse(i);
    }
}
```

Класс `X` наследуется от интерфейса - шаблона `Comparable <X>` и обязан иметь реализацию метода `int CompareTo (X other)`. В которой сравнение двух объектов класса `X` в одном случае (`sortOrder==1`) выполняется при помощи сравнения имен студентов, в другом (`sortOrder!=1`) при помощи сравнения их возрастов. Таким образом функция сортировки будет расставлять объекты толи в порядке, заданном именами, толи в порядке заданном возрастом.

Далее, накапливать объекты этого класса будем в списке - шаблоне типа `List<X>`:

```
List <X>  recs = new List <X>();
X         rec;
```

а в приведенный выше цикл нужно вставить строчки для создания объектов этого цикла и добавления их в список:

```
rec = new X(flds[0], flds[1]);
recs.Add(rec);
```

После выхода из цикла осталось из списка получить массив и, если это заданно аргументам командной строки, его отсортировать:

```
X []  recs2 = recs.ToArray(); // получить массив
X.sortOrder = 1;              // поле статическое, достучиться к нему
                                // можно через тип.
// X.sortOrder = 2;           задать порядок сортировки
Array.Sort(recs2);
```

И вывести в стандартный вывод данные в новом порядке.

В разделе 15.2 приводится пример создания собственного интерфейса для похожей задачи.

11.3 ЛАБОРАТОРНАЯ РАБОТА: ЗАКРЫТИЕ ОКНА

При разработке приложения для лабораторной работы 3.3 можно поступать как в будет полезна следующая информация:

- требуется подключение сборки System.Windows.Forms.dll как объяснялось в 16.1.1 ;
- Построить класс наследник для класса Form:

```
class w : Form {
public w ( string nm){
    Text = nm;          // задать заголовок окна
}
}

;
```

- Показать объект класса w на экране компьютере одним из трех возможных методов (а лучше поэкспериментировать со всеми тремя (ShowDialog(), Show(), Application.Run()) и запомнить разницу):

```
static
int Main(){
    w a = new w("dialog");
    DialogResult rc = a.ShowDialog();    //
    Console.WriteLine("rc = {0}", rc);
}
```

Методом ShowDialog показывают диалоговое окно, которое блокирует доступ к другим окнам приложения, выполнение останавливается на методе ShowDialog (вызывающий код ждет закрытия окна), можно вернуть результат диалога, объект a не был разрушен и его можно показать еще раз.

- Методом Show показывают дочерние окна, вызывающий код в операторе применении метода Show не ждет закрытия окна, окно после закрытия можно показать еще раз (объект не разрушается).

```
static
int Main(){
    w a = new w("dialog");
    w b = new w("1");
    b.Show();
    a.ShowDialog();
    b.Text = "222222222";
    b.Show();
    a.ShowDialog();
}
```

- Методом `Application.Run()` показывают главное окно приложения. Вызывающий код ждет закрытие окна, объект разрушается. Обычно применение метода `Show` происходит при существующем главном окне приложения.

```
static
int Main(){
    w a = new w("Main");
    Application.Run(a);
    Application.Run(a); // тут будет необработанное исключение
}
```

- Для различных событий, которые могут происходить в связи с работой приложения предлагается большой набор переменных - делегатов, каждое из которых представляет собой список указателей на методы специального типа `event delegate` и при возникновении события методы из переменной (если они были добавлены в список) выполняются. Например, при попытке закрыть окно (нажать крестик в правом верхнем углу окна) будут вызываться методы из переменной

```
event System.Windows.Forms.FormClosingEventHandler FormClosing;
```

где тип `event FormClosingEventHandler` - тип `event delegate` - обработчик события закрытия окна.

- что бы метод (обработчик) можно было положить в переменную - делегат (подписать на событие) он должен иметь правильный тип - делегат. В нашем случае это `event FormClosingEventHandler`. Метод объявляется следующим образом

```
void fc (object sender, FormClosingEventArgs a){
    Console.WriteLine("Form Closing of {0} is here ", Text);
}
void fc2 (object sender, EventArgs a){
    Console.WriteLine("Form Closing2 of {0} is here ", Text);
}
```

- Подписывание метода на событие происходит следующим образом:

```
public w ( string nm){
    Text = nm;
    FormClosing += fc;
    FormClosing += fc2;// в список можно положить несколько указателей на методы
}
```

- Метод `fc` второй формальный параметр может иметь типа `FormClosingEventArgs` (специальный для события закрытия окна) либо родительский `EventArgs` (который подходит ко всем событиям). Первый случай предпочтительнее, так как в переменной специализированного типа `FormClosingEventArgs` существуют дополнительные поля.
- Что бы иметь возможность отменить закрытие окна можно написать следующий обработчик:

```

void fc3 (object sender, FormClosingEventArgs a){
    if (e.CloseReason == CloseReason.UserClosing)
    {
        DialogResult resul =
            MessageBox.Show(
                "Точно закрываем окно?", "", MessageBoxButtons.YesNo);
        if (resul == DialogResult.Yes)
        {
            a.Cancel = false;
        }
        else
            a.Cancel = true;
    }
}

```

В котором выясняется причина закрытия окна (а именно, нужно убедиться что окно закрывает пользователь `CloseReason.UserClosing`, окно может закрываться по другой причине, например, полностью завершает работу операционная система), показывается диалоговое окно с кнопками Yes и No, обрабатывается результат нажатия на кнопки и отменяется закрытие окна.

Методические указания для выполнения этой лабораторной можно посмотреть в разделе [16.2.2](#)

11.4 ЛАБОРАТОРНАЯ РАБОТА: ДВОИЧНЫЕ ФАЙЛЫ

Полная версия приложения, удовлетворяющего условиям лабораторной работы [3.4](#), может иметь примерно такую подсказку:

Приложение для конвертации текстовых записей в двоичные и наоборот.

Использование:

`a.exe [-?] [-v] [-s NNN] [-sep SEP] [-3] [-douple] {-txt2bin | -bin2txt} -f FLNM`

где

<code>-help</code>	: выдача текущего экрана
<code>-?</code>	: выдача текущего экрана
<code>-v</code>	: выдача ошибок в файле и времени работы приложения
<code>-sep SEP</code>	: задать разделитель. Умолчание - символ пробела
<code>-txt2bin</code>	: направление конвертации - со стандартного ввода в двоичный
<code>-bin2txt</code>	: направление конвертации - с двоичного в стандартный вывод
<code>-3</code>	: записи состоят из трех чисел (умолчание из двух)
<code>-douple</code>	: записи состоят из вещественных чисел (умолчание из целых)
<code>-s NNN</code>	: NNN номер поля для сортировки
<code>-f FLNM</code>	: FLNM имя двоичного файла

Вертикальная черта '|' означает выбор одной из альтернатив, а фигурные скобки - означают обязательность одного или второго ключа. Квадратные скобки

`[-txt2bin | -bin2txt]`

означали бы, что можно пропустить оба ключа.

Далее, из лабораторной 10.6 можно использовать метод `getRes` для чтения записей из текстового файла, который написан с учетом обработки комментариев в CSV-файле, а в лабораторной 3.2 обсуждались вопросы сортировки коллекций.

Для чтения - записи простых типов в двоичном виде предназначены классы `BinaryReader` и `BinaryWriter`. Но конструкторы классов `BinaryReader` и `BinaryWriter` в качестве параметров не принимают название файла, как это было у `StreamReader` и `StreamWriter`. Они принимают класс `Stream`, предназначенный для низкоуровневого чтения - записи. А объекты класса `Stream` можно получить при помощи методов класса `File`, обычно это `Open`. Кроме того, напомним, что эти классы используют важный ресурс (а именно файл - из файловой системы), поэтому они являются наследниками интерфейса `IDisposable` и, значит, имеет метод `Dispose` для освобождения своего ресурса. Для удобства использования классов - наследников интерфейса `IDisposable` в .Net существует специальный оператор `using`, который гарантирует вызов метода `Dispose` независимо от возникновения или не возникновения исключительных состояний во время работы с файлом. Таким образом, при использовании инструкции `using` код для ввода данных будет выглядеть следующим образом:

```
string flNm = "1.bin"; // имя файла
using (BinaryReader br = new BinaryReader (
    File.Open(flNm, FileMode.Open) )){
    ....
} // после выхода из цикла и этой закрывающей
// скобки метод br.Dispose() будет вызван
```

А код для вывода - следующим:

```
using (BinaryWriter bw = new BinaryWriter (
    File.Open(flNm, FileMode.Create) )){
    ....
}
```

Перечисление `FileMode` задает различные режимы для работы с файлами. Например,

- `FileMode.Open` - операционная система должна открыть существующий файл. Отсутствие файла приведет к исключительному состоянию `FileNotFoundException`. Применение метода `Write` приведет к выводу данных в начало файла, таким образом, некоторые из старых данных будут потеряны;
- `FileMode.Create` - операционная система создаст новый файл если его не было, если он был, файл будет перезаписан;
- `FileMode.Append` - операционная система должна открыть существующий файл, и поток `Stream` позиционируется в конец файла. Применение метода `Write` приведет к дописыванию данных в конец файла. Все старые данные останутся.

Как упоминалось выше в каждом из классов для ввода - вывода двоичных данных существует множество методов. По крайней мере по одному для каждого простого типа. Рассмотрим некоторые для ввода данных:

- логическая переменная -

```
bool p = br.ReadBoolean();
```

- целая переменная -

```
int p = br.ReadInt32();
```

- короткое целое -

```
short p = br.ReadInt16();
```

- длинное -

```
short p = br.ReadInt64();
```

- отдельный байт -

```
byte p = br.ReadByte();
```

- короткое плавающее -

```
float p = br.ReadSingle();
```

- плавающее двойной точности -

```
double p = br.ReadDouble();
```

и так далее. Для вывода все гораздо проще, так как для каждого простого типа есть свой перегруженный метод Write.

```
bw.Write(1);           // целое
bw.Write(1L);          // длинное целое
bw.Write((short)1);    // короткое
bw.Write(2.71828F);    // плавающее одиночной точности
bw.Write(2.718281828); // плавающее двойной точности
bw.Write(true);        // логическое
bw.Write("farewell companions, good luck, hurrah");
bw.Write("the boiling sea under us");
```

Конструкцию using можно использовать в форме без конструктора, например, со стандартным ввод - выводом следующим образом:

```
using (TextWriter tw = Console.Out){
    using (TextReader tr = Console.In) {
        //
    }
}
```

где TextWriter и TextReader - это абстрактные классы, предназначенные для текстового ввода - вывода. Эти классы наследуются классами StreamWriter и StreamReader, соответственно, которые обычно используются при явном открытии тестовых файлов.

При выполнении лабораторной можно наследовать типы BinaryReader и BinaryWriter, добавив к ним требуемые методы для ввода - вывода двух целых, трех целых, двух плавающих и трех плавающих чисел. Или можно поступить аналогично разделу 15.2, в котором обсуждается более сложный способ, использующий механизм абстрактных классов.

11.5 ЛАБОРАТОРНАЯ РАБОТА: ВЫВОД ТАБЛИЦЫ В XML ФОРМАТЕ

Приложение, удовлетворяющее условиям лабораторной работы 3.5, может иметь примерно такую подсказку:

```
to convert some csv-file with specified encoding to xml-file
usage:
  a.exe [-?] -o PATH [-e ENC] [-s CHAR] [-f FLDNM1 [-f FLDNM2] ...]
options:
  -?           : to see this help: True
  -o PATH      : output xml path: .
  -f FLDNMx    : to set name of specified field. (number:name): field
  -e ENC       : encoding of standart input (866, 1251, UTF-8): 866
  -s CHAR      : csv separator: ,
```

Из нововведений в командной строке нужно отметить, что текст

```
[-f FLDNM1 [-f FLDNM2] ...]
```

сообщает оператору, что значения, которые задаются ключом -f, будут накапливаться. Их надо ввести столько, сколько полей предполагается явно именовать. Для всех остальных ключей, встреченных до этого момента, всегда значение из следующего (более правого) ключа, заменяет значение из предыдущего (левого) ключа.

Сначала зададим кодировку стандартного ввода. Для это используется свойство `InputEncoding` класса `Console`, которое имеет тип `Encoding`. Объекты класса `Encoding` создаются из строчных или целочисленных значений при помощи статических перегруженных методов этого же класса - `GetEncoding`. Список кодировок, поддерживаемых .Net, можно посмотреть на странице <https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.8>. Для текущей лабораторной представляют интерес две узкие (один байт - один символ) кодировки и одна широкая (один или два байта соответствуют одному символу): 866 - использовалась в DOS-е, а сейчас в консоли; 1251 - использовалась в Windows-95, равно как и сейчас; UTF-8 - новая кодировка, позволяющая пользоваться несколькими национальными алфавитами, включая китайский. Согласно приведенного выше списка, задать кодировку стандартного ввода, можно следующим образом:

```
Console.InputEncoding = Encoding.GetEncoding( "cp866" ); // 1
Console.InputEncoding = Encoding.GetEncoding( "windows-1251");
Console.InputEncoding = Encoding.GetEncoding( "utf-8" );
```

После выполнения одной из инструкций (для конкретики - первой) метод `ReadLine`

```
string s = Console.ReadLine();
```

будет выполнять перекодирование текста в умолчательную внутреннюю кодировку UTF-16 переменной `s` в предположении, что входной файл находится в 866 кодировке.

Замечание

Кроме, обсужденных в этом и предыдущем разделе, классов представляет интерес их абстрактный родитель - Stream. Он позволяет вводить - выводить данные без какого либо преобразования. В разделе [13.5](#) можно увидеть пример его использования.

Для задания имен полям может использоваться шаблон словаря.

```
using System.Collections.Generic;
...
Dictionary<int, string> fldDict;
```

Тип словаря означает, что целые числа будут выполнять роль индекса (Key в контексте словаря), а строки - значения, то есть, результат применения операции индексирования к переменной fldDict и индексу (ключу). Это в случае, если пара с таким индексом и значением была ранее добавлена в словарь. В противном случае, применение операции индексирования приводит к исключительной ситуации

KeyNotFoundException: The given key was not present in the dictionary

Добавление пары к словарю производится следующим образом:

```
static
void addNm(string field, Dictionary<int, string> fldDict){
    string[] val = field.Split(':'); // разделить параметр аргумента ком.строки.
    try {
        int k = int.Parse(val[0]);
        if (fldDict.ContainsKey(k) )
            return;
        fldDict.Add(k, val[1]);          // добавить следующую пару ключ - значение
    }
    catch {
        Console.Error.WriteLine("wrong parameter: {0}", field);
    }
}
```

После обработки всех ключей в качестве имени поля с номером i, будем использовать fldDict[i], если индекс i существует. Для этого используется метод ContainsKey:

```
string nm=null;
if( fldDict != null && fldDict.ContainsKey(i))
    nm = fldDict[i];
```

Иначе, в качестве имени поля можно взять чтото вроде такого:

```
nm = string.Format("field_{0}", i);
```

Перейдем к выводу в xml файла. Для этого потребуется область видимости System.Xml, из неё типы данных: XmlDocument - документ и узел XmlNode. Документ состоит из вложенных друг в друга узлов. Узел можно вкладывать в узел в качестве узла или в качестве атрибута. В атрибуты нельзя вкладывать узлы.

Создать и сохранить xml-документ, можно следующим образом:

```

XmlDocument xd = new XmlDocument();
xd.LoadXml("<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n<Table></Table>");
XmlNode tbl = xd.DocumentElement; // первый узел в документе - таблица
....
xd.Save("doc.xml");

```

Далее, к полученной таблице tbl нужно добавлять записи:

```

XmlNode attr;
XmlNode rec;
int recNo = 0;
string str = null;
string[] vals;
while ((str=tr.ReadLine()) != null){
    vals = getRec (str, ',');
    if(vals.Length == 1
        && vals[0].Length <=0 )    continue;
    recNo++;
    rec = xd.CreateElement("Record"); // создали новую запись
    attr = xd.CreateNode(             // создали атрибут - её номер
        XmlNodeType.Attribute, "No", null);
    attr.Value = recNo.ToString();
    rec.Attributes.SetNamedItem(attr); // добавили атрибут в запись
    ... // тут создать содержимое записи, то есть несколько полей.
    tbl.AppendChild(rec);             // добавили запись в таблицу
}

```

Осталось создать содержимое записи, то есть, добавить к узлу rec - запись несколько узлов поле, которые будут создаваться в переменной fld. У узла поле должно быть название (найденное в словаре и запомненное в ранее описанной переменной nm) и значение.

```

XmlNode fld ;
for (int i = 0; i < values.Length; i++){
    ...
    fld = xd.CreateElement(nm); // создали новый узел - очередное поле
    fld.InnerText = values[i];  // присвоили ему значение
    rec.AppendChild(fld);       // добавили его к записи
}

```

На этом вывод в xml - файл закончен. Аргументы командной строки позволяют использовать командные файлы консоли для тестирования приложений. Например, для тестирования текущего приложения использовался следующий файл:

```
---- File:./cs/labs/2xml/test.cmd
```

```

rem На входе досовский файл, переименовать одно поле
a -o .866.xml -f 1:aaa <866.txt
rem на входе - виндосовский файл
a -o .1251.xml -e 1251 <1251.txt
rem на входе - файл с кодировкой utf-8, переименовать два поля
a -o .utf-8.xml -e utf-8 <utf-8.txt -f 2:bbb -f 3:ccc

```

```
---- End Of File: ./cs/labs/2xml/test.cmd
```

Раздел 15.5 содержит простые примеры изменения кодировок текстовых файлов.

12 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: СОЗДАНИЕ ОКОННЫХ ПРИЛОЖЕНИЙ

Со следующей лабораторной работы, собственно, только и начинается разработка оконных приложений, которая существенно отличается от разработки консольных. Согласно классификации в [40] эти приложения принадлежат различным типам программных систем. Консольные приложения обычно являются преобразующими системами, то есть, системами, которые завершают свое выполнение после преобразования входных данных. Это приложения из почти всех лабораторных, которые были выполнены до сих пор. Оконные приложения - относятся к интерактивным (или диалоговым) системам, это системы, взаимодействующие с окружающей средой в режиме диалога. Характерной особенностью таких систем является то, что они могут контролировать скорость взаимодействия с окружающей средой - заставлять окружающую среду (читай оператора) ждать. Оказывается, что практически все реальные консольные приложения, которыми можно пользоваться не являются диалоговыми. К реальным примерам диалоговых консольных приложений можно отнести интерпретаторы командной строки, в семействе операционных систем Windows - это приложение cmd.exe, в семействе Unix - это sh, bash, zsh и так далее (и консольное приложение, добавляющее пользователей ОС). Также интерактивные консольные приложения часто встречаются в базах данных, например, в СУБД SQLite, это это оболочка командной строки - sqlite3.exe (см. <https://sqlite.org/cli.html>).

Методы проектирования и декомпозиции (декомпозиция означает разделение всего приложения на набор классов, или модулей или функций, так чтобы было удобно разрабатывать само приложение) преобразующей системы известны давно. С ними можно познакомиться в замечательной и не устаревшей работе Глена Майерса 'Надежность программного обеспечения' (см. [9]), в которой изложен замечательный формальный метод STS - декомпозиции (так же см. [22]) позволяющий проектировать практически любые преобразующие системы. При этом, для декомпозиции приложения (то есть, выбора модулей или классов с нужными свойствами) используется поток данных, которые приложение преобразует из входных в выходные.

С другой стороны, проектирование оконных приложений далеко не так формализовано. С одним из подходов к декомпозиции оконного приложения можно познакомиться в работе [27], при этом работу диалогового приложения можно рассматривать на некотором уровне абстракции, на котором её (работу диалогового) можно трактовать как работу преобразующего приложения, и, значит, можно применять методы проектирования изложенные в [9]. Последующие лабораторные подбирались с учетом идей, изложенных в [27].

12.1 ЛАБОРАТОРНАЯ РАБОТА: УПРАВЛЯЮЩИЕ ЭЛЕМЕНТЫ

Окна оконных приложений содержат внутри себя специальные области, предназначенные для выполнения некоторых отдельных подзадач из общей задачи организации обмена данными между приложением и оператором. Эти области называются управляющими элементами. Для связывания с ними методов и данных приложения используется класс `Control`, который является родителем всех управляющих элементов .Net. К таким элементам относятся, например, следующие классы: `TextBox` (область для ввода данных оператором), `Button` (кнопка, нажатие на которую при помощи устройства мышь, вызывает некоторые действия приложения), `Label` (метка - область экрана для вывода (обычно) текстовой информации) и так далее. Что бы управляющий элемент (то есть, область на экране) созданного объекта, появилась в окне, объект нужно добавить в коллекцию управляющих элементов окна (`Controls`). Как всегда с коллекциями это выполняется при помощи метода `Add`.

```
Control cnt = new Button();  
Controls.Add(cnt);
```

Поскольку области на экране (прямоугольники) обладают такими характеристиками как размер и положение, их можно и нужно задавать при помощи использования методов и свойств объекта. В частности, положение управляющих элементов (равно как и окна) задается координатами левого верхнего угла, например, при помощи свойства `Location` имеющего тип `Point` из пространства имен `System.Drawing`.

```
int y = 10    // расстояние в пикселях от верхнего края окна  
, x = 20;    // расстояние в пикселях от левого края окна  
cnt.Location = new Point(x, y);
```

Для этой же цели можно использовать свойства `Top` и `Left`. Размеры задаются свойствами (типа `Point`) `Size` (размер с границей элемента) и `ClientSize` (размер без границы элемента), или свойствами (типа `integer`) `Width` и `Height`. Для выполнения лабораторной достаточно упомянуть свойства (типа `string`) `Text` - текст, который виден оператору, `Name` - идентификатор (текст, который не виден оператору), который может использовать программист в своих целях. Например, при помощи этого свойства можно искать в коллекции `Controls` требуемый объект.

Выполнение лабораторной 5.1 предусматривает просто демонстрацию окна, содержащего заданный управляющий элемент. В случае класса `wnd`, приведенного ниже это может быть строка

```
Application.Run(new wnd(5, 10, 3, "some text")); // главное окно приложения
```

12.1.1 Законченный пример конструктора окна

```
---- File:./cs/labs/mod3lab1/wnd.cs
```

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

```

namespace m3l1{
    class wnd : Form    {
        public
        wnd(int x, int y, int f, string w)    {
            this.Width  = 300; // изменение размера окна
            this.Height = 200;
            Control cnt;
            if (f == 1)
                cnt = new Button();
            else if (f == 2)
                cnt = new Label();
            else if (f == 3)
                cnt = new TextBox();
            else
                return;
            // cnt.Location = new Point(x, y);
            cnt.Left = x;    // изменение координат
            cnt.Top  = y;    // левого верхнего угла
            cnt.Text = w;    // управляющего элемента
            Controls.Add(cnt);
        }
    }
}

---- End Of File:./cs/labs/mod3lab1/wnd.cs

```

12.2 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛОГОВОЕ ОКНО OkCancel

Порядок выполнения лабораторной работы [5.2](#) приводится ниже.

Создается родительский класс OkCancel для диалогового окна с кнопками Ок и Отменить. Подразумевается известным, что управляющий элемент появляется в окне после добавления элемента к коллекции Controls. Например, Controls.Add(butOk); Кроме этого требуется настраивать некоторые свойства, обычные для диалоговых окон.

- Что бы иметь возможность использовать тип Size, к приложению добавить библиотеку System.Drawing.dll и в код добавить сторочку

```
using System.Drawing;
```

- Создать класс OkCancel, который наследуется из класса Form. Класс должен содержать две кнопки

```
Button butOk;
Button butCancel;
```

- Запретить оператору менять размер диалогового окна и убрать все лишние управляющие элементы.


```

FormBorderStyle = FormBorderStyle.FixedDialog;
MinimizeBox    = false;
MaximizeBox    = false;
ControlBox     = false;
AutoScroll     = false;

```

- Создать переменную $p = 4$, это число будет использоваться для создания полей между границей окна и его управляющими элементами. Свойство окна называется `Padding`

```
Padding = new Padding (p,p,p,p);
```

- Далее, задать свойство окна для того, чтобы в нем умещались добавляемые управляющие элементы.

```
AutoSize = true;
```

- Создать первую панель с заданной высотой и разместить ее сверху формы.

```

Panel p0 = new Panel ( );    // эта панель задает высоту строки
p0.Size  = new Size (1, 32);
p0.Dock  = DockStyle.Top;
Controls.Add (p0);

```

- Создать кнопку `Ok` и положить её в правую сторону.

```

butOk = new Button();
butOk.Size = new Size( 112,32);
butOk.Dock = DockStyle.Right;    // вправо
butOk.Text = "Ok";

```

- Для пустого пространства между кнопками справа добавить еще одну панель.

```

p0 = new Panel();    // эта панель задает расстояние между кнопками
p0.Size = new Size( 112,32);
p0.Dock = DockStyle.Right;

```

- Справа добавляем кнопку `Cancel`

```

butCancel = new Button();
butCancel.Size = Size( 112,32);
butCancel.Dock = DockStyle.Right;
butCancel.Text = "Cancel";

```

- Присвоить кнопкам возвращаемые значения и связать кнопки с клавишами.

```

butOk.DialogResult = DialogResult.OK;
butCancel.DialogResult = DialogResult.Cancel;
DialogResult = DialogResult.Cancel;
AcceptButton = butOk;    // нажатие клавиши Enter как на ок
CancelButton = butCancel;    // нажатие клавиши Esc КАК НА Cancel

```

- В методе Main создать объект созданного класса и получить нажатую кнопку.

```
OkCancel w = new OkCancel();
DialogResult rc = w.ShowDialog();
Console.WriteLine("result is {0}", rc);
```

12.3 ЛАБОРАТОРНАЯ РАБОТА: ДИАЛоговое ОКНО ВВОДА ДАННЫХ

Лабораторная 5.3 является одной из самых сложных. В ней (как и во всех следующих) остаются в силе все указания для лабораторной 16.2.3, поэтому можно перекопировать все файлы этой лабораторной в новый каталог и менять метод Main. Во всех файлах проекта должны совпадать имена именованных областей видимости, которые задаются строчками

```
namespace laba3 {
    ....
}
```

Далее, поскольку в предыдущей лабораторной для горизонтального размера трех управляющих элементов использовалось число 112 (с расчетом на экраны хорошего разрешения), то для обсуждаемых элементов выберем число $3 \cdot 112 / 2 = 168$.

```
t1.Size = new Size(162, 32);
```

Итак, для выполнения лабораторной требуется:

- Создать класс для описания полей ввода fld. Класс будет содержать название поля ввода, видимого оператору (тип Label); строку, в которую попадет значение, введенное оператором в поле ввода (тип TextBox); строка с умолчательным значением для поля ввода.

```
public class fld {
    public string name;        /// название поля
    public string value;       /// место для получения значения
    public string def;         /// значение по умолчанию
    public fld (string nm, string d = ""){
        name = nm;
        value = "";
        def = d;
    }
}
```

- В методе Main создать два объекта класса fld и окно класса inWnd. Построить окно для ввода этих параметров и продемонстрировать введенные значения после нажатия на кнопку Ok:

```
fld a = new fld("Имя", "Вася");
fld b = new fld("Фамилия", "Пупкин");
inWnd w = new inWnd("Окно для входа в приложение", a, b);
DialogResult rc = w.ShowDialog();
if (rc == DialogResult.OK) {
    Console.WriteLine("в приложение зашел пользователь: {0}/{1}", a.value, b.value);
}
```

- Класс для ввода значений inWnd наследовать из класса OkCancel (см. [16.2.3](#)). Для поддержки нескольких полей ввода в класс требуется добавить список пар TextBox и fld:

```
using System.Collections.Generic;
public class inWnd: OkCancel    {
    Dictionary<TextBox, fld>    flds = null;
}
```

Он будет использоваться в обработчике нажатия на кнопки для переноса значений из полей ввода в параметры окна типа fld и восстановления умолчательных значений.

- Для случая ввода двух полей конструктор inWnd должен иметь три параметра:

```
public inWnd (string tit, fld i, fld f){
    Text = tit;                               // заголовок окна
    flds = new Dictionary<TextBox, fld>();    // создать список пар
}
```

- В класс inWnd добавить метод создан

```
public TextBox addFld                               // метод добавляет в окно новое поле ввода
(int n,  fld par){
    return null;
}
```

- После этого в конструктор добавим два вызова этого метода

```
flds.Add(addFld (0,  i), i);           /// добавили первое поле ввода
flds.Add(addFld (1,  f), f);           /// добавили второе поле ввода
```

- Пишем тело метода addFld, в котором для добавления одного поля требуется два объекта Panel, объект Label - метка и объект TextBox - поле ввода. Метод будет возвращать созданный TextBox:

Первая панель содержит панель (чтобы зафиксировать высоту), метку и поле ввода.

```
Panel p      =new Panel();               /// панель содержит метку и поле ввода
p.Size       = new Size(1, 32);
p.AutoSize   = true;
p.Dock       = DockStyle.Top;
p.Padding    = new Padding (0,0,0,8);    // добавить пустое пространство
                                              // между метками
```

В неё добавляем вторую панель, прижимая вверх:

```
Panel p1     = new Panel();               /// панель для высоты
p1.Size      = new Size(1, 32);
p1.Dock      = DockStyle.Top;             /// <-----
p.Controls.Add(p1);                       /// добавить в первую
```

Добавляем метку с названием поля ввода, прижимая её вправо:

```
l1 = new Label();
l1.Size      = new Size(162, 32);;
l1.Text      = par.name;          /// название из параметра i
l1.Dock      = DockStyle.Right ;
p.Controls.Add(l1);
```

Добавляем поле ввода, прижимая вправо:

```
t1 = new TextBox();
t1.Size      = new Size(162, 32);
t1.TabIndex  = n;                // задать порядок прохода по полям ввода
                                   // при нажатии на клавиши Tab
t1.Text      = par.def;          /// присвоили умолчательное значение
                                   // полю ввода
t1.Dock      = DockStyle.Right;
p.Controls.Add(t1);
```

Первое поле р создано, добавляем его в список управляющих элементов окна:

```
Controls.Add(p);          // добавили в окно готовую панель с двумя управляющими элементами
return t1;                // вернули TextBox
```

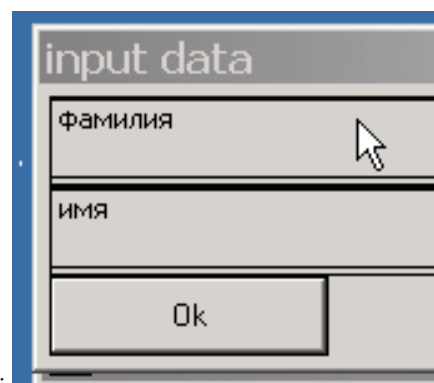
- В конструктор inWnd добавить обработчик нажатия на кнопку Ok:

```
public inWnd (string tit, fld i, fld f){
    ....
    butOk.Click += _KeyDown;
}
```

- Тело обработчика, естественно, нужно добавить в класс inWnd:

```
void _KeyDown(object sender, EventArgs e) {
    TextBox tb;
    fld      f;
    foreach (KeyValuePair<TextBox, fld> Item in flds)
    {
        tb = Item.Key;
        f  = Item.Value;
        f.value = tb.Text;    // в этот момент введенный текст попадает в параметр окна
        tb.Text = f.def;      // а теперь восстанавливается значение по умолчанию.
    }
}
```

- Похожий обработчик для восстановления значений по умолчанию желательно добавить либо к нажатию на кнопку Cancel, либо к событию активизации окна Activate.



Если, после всех этих беспримерных усилий, получилось окно вроде:

Окно ввода данных

то это означает, что можно создавать диалоговые окна для ввода данных оператором приложения.

Полученный класс можно использовать во всех оставшихся лабораторных курса.

Замечание

В тестирующем приложении для библиотеки Обработки аргументов командной строки (см. [19]) демонстрируется создание такого диалогового окна, класса `OkCancelDlg`, причем его конструктор принимает переменное число параметров:

```
OkCancelDlg it = new OkCancelDlg( "window to input some params", Logger
                                , user          // можно добавлять или
                                , pwd           // удалять параметры, это
                                , email         // повлияет на их количество в
                                , perCent       // демонстрируемом окне
                                , test_Str_Arr );
DialogResult rc = it.ShowDialog();
if (rc == DialogResult.OK) {
    Console.WriteLine ("your input : {0}/{1}/{2}/{3}/{4}"
        , (string)user
        , (string)pwd
        , (string)email
        , (string)test_Str_Arr
        , (double) perCent
    );
}
else
    Console.WriteLine ("nothing to show");
```

В классе поддерживается ввод паролей, шаблоны для ввода текстов (проверка на похожесть введенного текста на адрес электронной почты), целые и вещественный типы с ограничением по диапазону, ввод перечислений.

12.4 ЛАБОРАТОРНАЯ РАБОТА: МЕНЮ В ГЛАВНОМ ОКНЕ ПРИЛОЖЕНИЯ

Перед началом выполнения лабораторной работы 5.4 напомним, что для демонстрации главного окна приложения необходимо использовать метод

```
Application.Run(new Form());
```

Главное окно приложения обычно содержит меню из следующих элементов: Exit - Выход (подэлемент элемента меню File); крайний слева элемент меню File - обычно содержит несколько подэлементов для выхода из приложения, открытия файла, закрытия файла и так далее; крайний справа элемент меню - About - О себе (содержит информацию о назначении и использовании приложения). Внизу будет размещена строка - строка для отображения состояния приложения. Обычно оператор сможет менять размер главного окна приложения.

- Сначала в области видимости класса окна объявим объект статус строки, управляющий элемент класса StatusStrip:

```
StatusStrip statStrip;
```

- В конструкторе окна создадим управляющий элемент Menu:

```
Menu = new MainMenu();
```

- Минимальный набор требуемых элементов меню:

```
MenuItem FileIt = new MenuItem("File");
MenuItem workIt = new MenuItem("work");
MenuItem AboutIt = new MenuItem("About");
MenuItem ExitIt = new MenuItem("Exit");
```

- Добавим их (кроме Exit) к главному меню:

```
Menu.MenuItems.Add(FileIt);
Menu.MenuItems.Add(workIt);
Menu.MenuItems.Add(AboutIt);
```

- Добавим Exit к элементу File:

```
FileIt.MenuItems.Add(ExitIt);
```

- В конструкторе окна осталось добавить статус строку из двух меток:

```
statStrip = new StatusStrip();
ToolStripStatusLabel
    statLabel = new ToolStripStatusLabel();
statStrip.Items.Add(statLabel);

statLabel = new ToolStripStatusLabel();
statStrip.Items.Add(statLabel);          ///
Controls.Add(statStrip);
statStrip.Items[0].Text = "окно готово";
statStrip.Items[1].Text = "дочерних окон нет";
```

- Далее, требуется создать два обработчика нажатий на элементы меню (Click). Один - для элемента About:

```
void aboutF(object sender, EventArgs a)
{
    Console.WriteLine("about");
    statStrip.Items[1].Text = "открыли окно About";
    MessageBox.Show("This programm was made by ImiaRek, Kiev 201?.");
    statStrip.Items[1].Text = "дочерних окон нет";
}
```

Второй для закрытия приложения (Exit):

```
void exitF(object sender, EventArgs x)
{
    Console.WriteLine("FormClosing");
    Close();
}
```

- В конструкторе добавим их к событиям - переменным:

```
AboutIt.Click += aboutF;
ExitIt .Click += exitF;
```

12.5 ЛАБОРАТОРНАЯ РАБОТА: ПАНЕЛЬ ИНСТРУМЕНТОВ ДОЧЕРНЕГО ОКНА С ПРЕДСТАВЛЕНИЕМ ТАБЛИЧНЫХ ДАННЫХ

Лабораторная 5.5 объемная и, поэтому, объяснение разделяется на две части. В первой части будет создан и заполнен данными управляющий элемент для представления таблиц, во второй, к этому окну добавим управляющий элемент панель инструментов для будущего редактирования таблицы.

12.5.1 Окно с представлением табличных данных

Для выполнения лабораторной работы необходимо выполнить следующие действия:

- Подключить к проекту динамически подгружаемую библиотеку System.Windows.Forms.dll. При компиляции проекта из командной строки, команду

```
csc /out:a.exe *.cs
```

заменить на

```
csc /out:a.exe /r:System.Windows.Forms.dll *.cs
```

- Именованную область видимости System.Windows.Forms добавить к просмотру по умолчанию. В код добавить сторочку

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

- Перед методом Main добавить атрибут статического однонитевого приложения:

```
[STAThread]
static void Main(string[] args)
```

- Класс OpenFileDialog использовать для выбора нужного файла с данными. Приблизительно в таком виде:

```
OpenFileDialog o1
    = new OpenFileDialog(); // создать переменную
o1.Filter          // отфильтровать файлы директории
    = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
o1.InitialDirectory // начальный каталог для просмотра
    = Path.GetDirectoryName( // должен совпадать с местом
                             // откуда выполняется сборка exe
                             Assembly.GetExecutingAssembly().Location
    );
if (o1.ShowDialog() == DialogResult.OK)
{
    // если файл был задан, получить его имя
    string fnm = openFileDialog1.FileName;
    // продолжать работу здесь
}
```

- Создать класс для демонстрации будущего окна, с управляющим элементом DataGridView:

```
class wnd: Form {
    string flNm;
    protected DataGridView dgv; // это поле будет доступно наследникам класса
    public wnd (string fileName) {
        flNm = fileName;
        dgv = new DataGridView();
        dgv.ReadOnly = true; // запретить редактирование ячеек оператором
        dgv.AllowUserToAddRows = false; // запретить добавление строчек оператором
        dgv.Dock = DockStyle.Fill; // заполнить все окно
        dgv.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        Controls.Add(dgv);
    }
}
```

- Создать обработчик для заполнения DataGridView:

```
void fLoad(object sender, EventArgs a) {
    Console.WriteLine("Load");
    ReadFile(flNm);
}
```

- Создать метод для чтения файла:


```

public void ReadFile(string path){
    string str;
    string[] fields;
    bool firsttime = true;
    using (StreamReader sr = new StreamReader(path)) {
        while ((str = sr.ReadLine()) != null){
            fields = str.Split(';');    // точка с запятой - разделитель записей
            if (fields.Length <= 0)    // пропустим пустые строки
                continue;
            if (firsttime){
                // на первой строке узнать кол-во колонок
                dgv.ColumnCount = fields.Length;
                for (int i = 0; i < fields.Length; i++) {
                    dgv.Columns[i].Name = String.Format("Pole{0}", i+1);
                }
                firsttime = false;
            }
            dgv.Rows.Add(fields);    // добавить новую строку
        }
    }
}

```

- В конструкторе wnd к событию Load добавить созданный обработчик:

```
Load += fLoad;
```

Событие Load возникает при появлении окна на экране компьютера в первый раз при выполнении приложения. Для каждого появления каждого окна при каждом выполнении приложения существует одно событие Load.

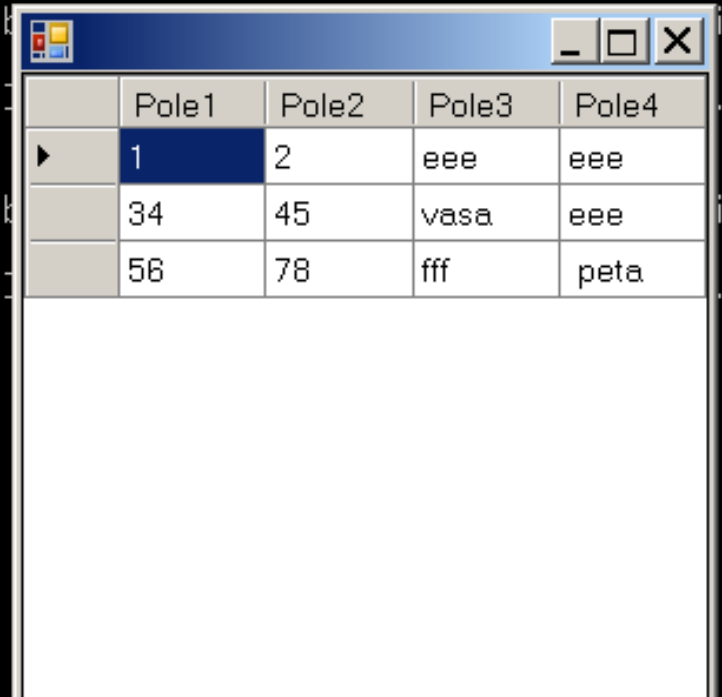
- В методе Main создать окно созданного класса wnd и показать его в качестве главного окна приложения:

```

{ // если файл не был задан, получить его имя
  string fnm = openFileDialog1.FileName;
  // продолжать работу здесь
  Application.Run(new wnd(fnm));
}

```

Эта деятельность должна закончиться приложением с приблизительно таким окном:



	Pole1	Pole2	Pole3	Pole4
▶	1	2	eee	eee
	34	45	vasa	eee
	56	78	fff	peta

Первая демонстрация табличных данных

12.5.2 Панель инструментов дочернего окна

В лабораторной 12.5.1 объяснялись начальные действия по наполнению управляющего элемента DataGridView. Далее, требуется добавить к окну два управляющих элемента: уже использованную в лабораторной 12.4 статус строку - StatusStrip и новый элемент - полоса инструментов ToolBar. Итак, для разработки примера по представлению табличных данных необходимо:

- Из класса wnd наследовать класс wnd2:

```
public class wnd2 : wnd
{
    ToolBar    tb        = new ToolBar();
    StatusStrip statStrip = new StatusStrip();
    public wnd2(string fnm):base (fnm)
    {
    }
}
```

- Заменить создание окна в методе Main:

```
// предыдущее окно класса родителя без управляющих элементов
// Form y = new wnd(fnm);
//разрабатываемое окно класса наследника с управляющими элементами
Form y = new wnd2(fnm);
```

- В конструктор wnd2 добавить создание и инициализацию статус строки:

```
ToolStripStatusLabel
statLabel = new ToolStripStatusLabel();
statStrip.Items.Add(statLabel);
Controls.Add(statStrip);
statStrip.Items[0].Text = "datagrid window is ready";
```

- В конструктор wnd2 добавить размеры для будущих кнопок управляющего элемента ToolBar (полоса инструментов), что бы сделать их одинаковыми:

```
tb.ButtonSize = new System.Drawing.Size((int)(200/ 3), (int)(40) );
```

- В конструктор wnd2 создать несколько кнопок для управляющего элемента ToolBar (это объекты класса ToolBarButton). Задать у них текст подсказки (поле класса ToolTipText):

```
ToolBarButton tlbExit = new ToolBarButton("Exit");
tlbExit.ToolTipText = "закрыть окно таблицы";
ToolBarButton tIns = new ToolBarButton("Insert");
tIns.ToolTipText = "добавить запись";
ToolBarButton tEdit = new ToolBarButton("Edit");
tEdit.ToolTipText = "редактировать запись";
ToolBarButton tDel = new ToolBarButton("Delete");
tDel.ToolTipText = "удалить запись";
ToolBarButton tSave = new ToolBarButton("Save");
tSave.ToolTipText = "сохранить изменения";
ToolBarButton tExp = new ToolBarButton("Export");
tExp.ToolTipText = "экспортировать таблицу";
```

- Добавить их к полосе инструментов

```
tb.Buttons.AddRange(new ToolBarButton[] {
    tIns
    , tEdit
    , tDel
    , tSave
    , tExp
    , tlbExit
});
```

- Прикрепить полосу инструментов к верхнему краю окна:

```
tb.Dock = DockStyle.Top;
Controls.Add(tb);
```

- Создать еще один обработчик события Load (первый будет вызываться из класса wnd), который запишет в статус строку количество строчек управляющего элемента DataGridView:

```

void fLoad(object sender, EventArgs a)
{
    statStrip.Items[0].Text
    = string.Format("{0} records has been red", dgv.Rows.Count);
}

```

- Добавить обработчик к делегату - полю события Load:

```
Load += fLoad;
```

- Создать обработчик события Click на полосе инструментов:

```

void ToolBarButtonClick(object sender, ToolBarButtonClickEventArgs e)
{
    string bNm = e.Button.Text;
    statStrip.Items[0].Text
    = string.Format("You've pressed {0} button", bNm);
    if (bNm == "Exit")
        Close();
    else if (bNm == "Open")
        ;
    else {
        MessageBox.Show("action not ready!", "Warning");
    }
}

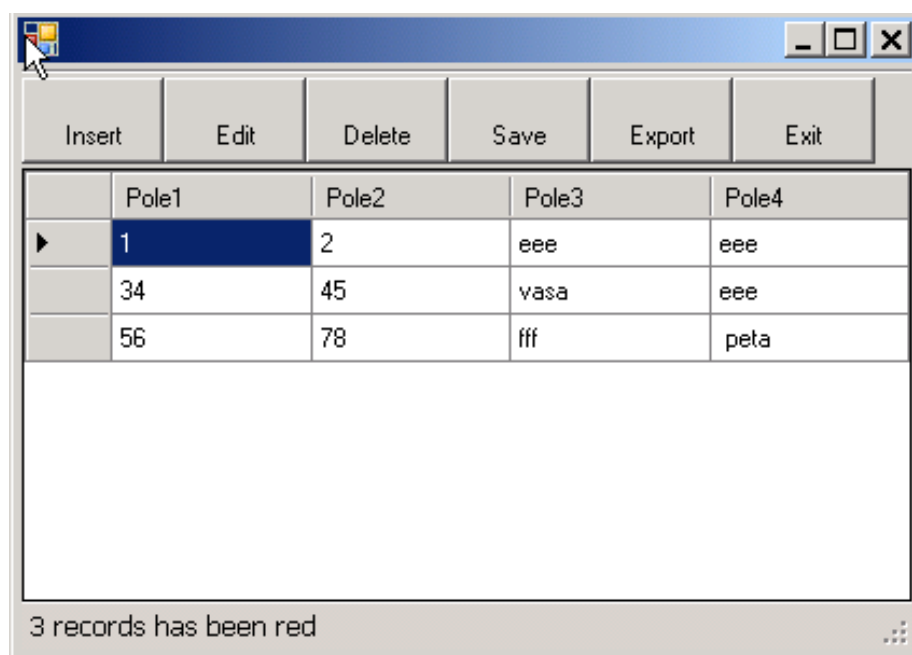
```

По коду примера понятно, что класс параметров данного обработчика ToolBarButtonClickEventArgs позволяет узнать название соответствующей кнопки и вызывать код, предназначенный для неё.

- Добавить обработчик к делегату - полю полосы инструментов tb события ButtonClick:

```
tb.ButtonClick += new ToolBarButtonClickEventHandler(ToolBarButtonClick);
```

Эта деятельность должна закончиться приложением с приблизительно таким окном:



демонстрация табличных данных

Далее, это разработанное окно показывать при нажатии на элемент меню Work главного окна (см. 5.4).

12.6 ЛАБОРАТОРНАЯ РАБОТА: ОКНО РЕДАКТИРОВАНИЯ ТАБЛИЦ

В этой лабораторной класс `inWnd`, разработанный в лабораторной Создание окна ввода данных (5.3), будет использован для редактирования CSV - файлов, при помощи приложения лабораторной Представление табличных данных (5.5).

12.6.1 Редактирование таблицы, часть 1

Разработку приложения для редактирования таблицы начнем с добавления функциональности для кнопки `Insert`.

- Перекопировать в отдельный каталог файлы из 12.5.2 ;
- В этот же каталог из 12.3 скопировать файлы `inWnd.cs` и `OkCancel.cs`;
- В классе `wnd2` К полям `tb` и `statStrip`, и к методу `ToolBarButtonClick` дописать ключевое слово `protected`;
- создать файл `wnd3.cs`:

```
namespace dtGrd
{
```

```

public class wnd3 : wnd2
{
    public wnd3(string fnm):base (fnm)
    {    // удалить обработчик родительского класса
        tb.ButtonClick -= base.ToolBarButtonClick;
    }
}

```

К нему подключить область видимости лабораторной Окно Ввода данных:

```
using laba3;
```

- В методе Main выполнить окно нового класса:

```
Application.Run(new wnd3(fnm));
```

Нажатие на кнопки управляющего элемента панель инструментов перестала реагировать на нажатия кнопок.

- К классу wnd3 добавить его обработчик

```

void ToolBarButtonClick3(object sender, ToolBarButtonEventArgs e)
{
    string bNm = e.Button.Text;
    statStrip.Items[0].Text
        = string.Format("You've pressed {0} button", bNm);
    if (bNm == "Exit")
        Close();
    else if (bNm == "XXX")
        ;
    else { // вызвать родительский обработчик
        base.ToolBarButtonClick
            ( sender, e);
    }
}

```

В котором вызывается обработчик родительского класса wnd2 и добавить его к событию

```
tb.ButtonClick += ToolBarButtonClick3;
```

Это восстановит реакцию приложения при нажатии кнопок панели инструментов.

- Что бы была возможность редактировать таблицы с любым числом колонок, в класс inWnd добавить конструктор для передачи в окно массив полей типа fld:

```

public inWnd(string Title, fld[] ps) : base(Title)
{
    flds = new Dictionary<TextBox, fld>(); // создать список пар поле ввода + параметр
    for (int i = 0; i < ps.Length; i++)    // в список пар добавить поле ввода, сделанное
        flds.Add(addFld (i, ps[i]), ps[i]); // очередного параметра + сам параметр
    butOk.Click += _KeyDown;
}

```

- В класс wnd3 добавить метод для подготовки:

```

public void doIns(){
    fld f ;
    List<fld> flds = new List<fld>();
    for (int i = 0; i < dgv.ColumnCount; i++){
        f = new fld (dgv.Columns[i].Name, "");
        flds.Add(f);
    }
    if(flds.Count > 0){
        Form w = new inWnd("Input new record", flds.ToArray());
        DialogResult rc = w.ShowDialog();
    }
    else {
        statStrip.Items[1].Text
        = string.Format("nothing to do!");
    }
}

```

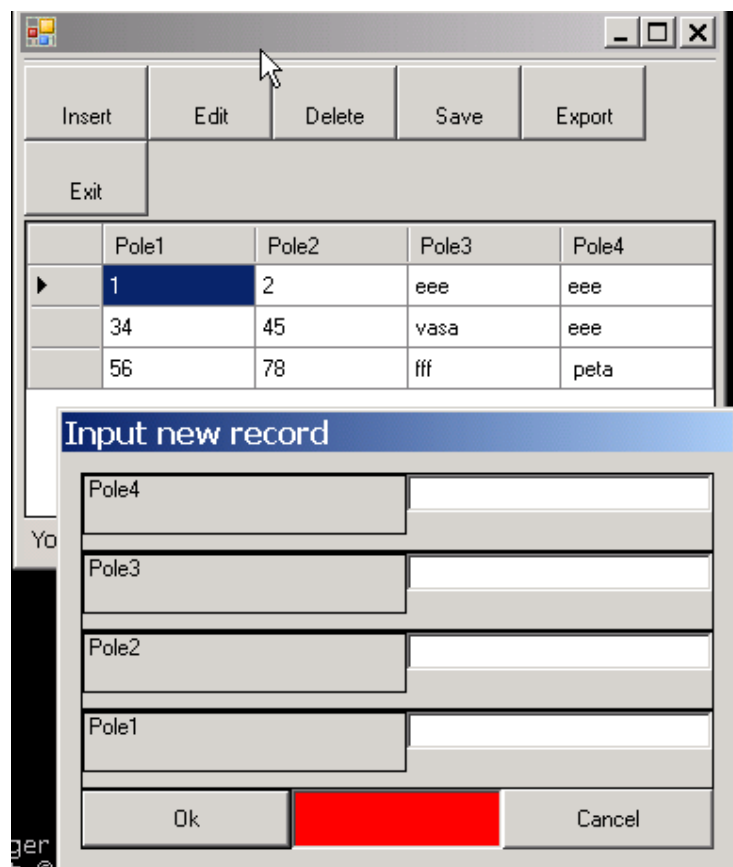
- Для вызова окна ввода данных при нажатии на кнопку "Insert" требуется изменить обработчик ToolBarButtonClick3:

```

string bNm = e.Button.Text;
if (bNm == "Exit")
    Close();
else if (bNm == "Insert")
    doIns();
else { // вызвать родительский обработчик
    base.ToolBarButtonClick
    ( sender, e);
}

```

После нажатии на кнопку Insert на панели инструментов окна просмотра таблицы должно появиться окно ввода данных. Причем количество полей в окне ввода данных будет совпадать с количеством колонок таблицы.



добавление записи

Первая часть окончена.

12.6.2 Редактирование таблицы, часть 2

Во второй части лабораторной добьемся того, что бы данные из окна ввода данных при нажатии на кнопку Ok попадали в таблицу и, после нажатии на кнопку Save, полосы инструментов, попадали в исходный файл. Кроме того, немного изменим вид таблицы (для изучения новых свойств класса DataGridView) и добавим реакцию приложения при нажатии на клавиши клавиатуры.

Продолжаем разработку приложения редактирования CSV-файла.

- Убрать из таблицы самую левую серую колонку без названия с черным треугольником (см. рис. 12.5.2 на стр. 77) и синее пятно для выделения ячейки заменим на широкое синее пятно для выделения записи:

```
dgv.SelectionMode
= DataGridViewSelectionMode.FullRowSelect; // выделяем всю запись
dgv.RowHeadersVisible = false;             // слева удаляем лишнюю колонку
```


- Доработать метод `dolns` из класса `wnd3`, что бы при нажатии на кнопку Ok окна ввода данных, введенная строка попала в таблицу:

```

DialogResult rc = w.ShowDialog();
if (rc == DialogResult.OK) { // если нажали Ok
    statStrip.Items[0].Text
        = string.Format("insert after {0} rec"
            , dgv.CurrentRow.Index);
    string[] fs = new string[flds.Count]; // создали массив для вставки
                                           // новой записи в таблицу
    for (int j = 0; j < flds.Count; j++) { // заполнили массив новыми полями
        fs[j] = flds[j].value;
    }
    dgv.Rows.Insert(dgv.CurrentRow.Index, fs); // вставили новые поля
} // в таблицу

```

Строчка добавлена в таблицу.

- В самом первом родительском классе `wnd` для демонстрации таблицы из лабораторной 5.5 открыть доступ методам из классов наследников (то есть, `wnd3`) к полю с именем файла демонстрируемой таблицы:

```

protected // это поле должно быть доступно
string fileName; // для обработчика кнопки сохранения
                // в окне редактирования таблицы

```

- Создать метод `doSave` для сохранения таблицы в файл из которого она была прочитана:

```

public void doSave(){
    string ss = csvExport (fileName // поле класса родительского класса wnd
        , ',', dgv);
    Console.WriteLine("Save: '{0}'" , ss);
    statStrip.Items[0].Text = ss;
}

```

Предполагается, что текст метода `csvExport`, который из управляющего элемента `dgv` данные переносит в csv - файл с именем `fileName` и разделителем `','` тоже надо написать. Более подробно см. в 16.2.8 .

- Добавить в обработчик `ToolBarButtonClick3` вызов добавленного метода `doSave` (метод `dolns` уже вызывался в первой части):

```

else if (bNm == "Save")
    doSave();

```

Можно проверить работу кнопок панели инструментов `Insert` и `Save`.

- Добавить в обработчик `ToolBarButtonClick3` вызов добавленного метода `doSave` (метод `dolns` уже вызывался в первой части):

```

else if (bNm == "Edit")
    doEdit();

```

, а в класс заглушку метода:

```
public void doEdit(){ }
```

- Сначала в doEdit проверить есть ли записи в таблице:

```
public void doEdit(){
    string ss = "";
    if(dgv.RowCount > 0) {
        // редактирование тут
    }
    else
        ss = "nothing to dp!";
    statStrip.Items[0].Text = ss;
}
```

- Подготовить список полей для создания окна редактирования записи:

```
if(dgv.RowCount > 0) {
    DataGridViewRow dgvR = dgv.Rows[dgv.CurrentRow.Index]; // из текущей строки берем
    DataGridViewCellCollection row = dgvR.Cells;             // значения по умолчанию
    List<fld> flds = new List<fld>();
    fld f;
    for (int i = 0; i < dgv.ColumnCount; i++){
        f = new fld (dgv.Columns[i].Name
                    , (row[i]).Value.ToString());             // значения по умолчанию
        flds.Add(f);                                           // используются для создания поля
    }
    if(flds.Count > 0){
        // редактирование тут
    }
    else
        ss = "Error: there are not any column!";
}
```

- Показать окно и получить из него данные введенные оператором:

```
if(flds.Count > 0){
    Form w = new inWnd("Edit selected record", flds.ToArray());
    DialogResult rc = w.ShowDialog();
    if (rc == DialogResult.OK) {
        // если ок
        for (int kk = 0; kk < flds.Count; kk++){ // перенести значения в таблицу
            dgvR.Cells[kk].Value = flds[kk].value;
        }
    }
}
```

Строчки уже будут редактироваться и, значит, их можно сохранить в файле.

- Доработать нажатие на DoubleClick на записи. Это аналог нажатия на кнопку Edit. Делегат - обработчик:

```
protected
void dc2(object sender, DataGridViewCellEventArgs e)
{
    Console.WriteLine("CellDoubleClick: selrow / row: {0}/{1} "
        , dgv.CurrentRow.Index, e.RowIndex);
    if (e.RowIndex >= 0){
        doEdit();
        dgv.CurrentCell = dgv.Rows[e.RowIndex].Cells[0];
    }
}
```

И в конструкторе добавить обработчик к событию DoubleClick в ячейке таблицы:

```
dgv.CellDoubleClick += dc2; // редактирование записи по enter
```

- Обработчик для нажатия на клавишу Enter:

```
void _PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        Console.WriteLine("CellDoubleClick: selrow / row: {0} "
            , dgv.CurrentRow.Index);
        doEdit();
    }
}
```

Добавить обработчик к событию KeyDown:

```
dgv.PreviewKeyDown += _PreviewKeyDown;
```

- Нажатие на клавишу Enter стандартно приводит к переходу на следующую строку. Лично мне это поведение не нравится, избавиться от него:

```
void _KeyDown(object sender, KeyEventArgs e) {
    if (e.KeyCode == Keys.Enter) {
        e.SuppressKeyPress = true;
    }
}
```

и

```
dgv.KeyDown += _KeyDown;
```

Таблица должна редактироваться и сохраняться. Обработку нажатия на клавишу Insert (вставка записи в таблицу) сделать самостоятельно.

12.7 ЛАБОРАТОРНАЯ РАБОТА: МНОГОДОКУМЕНТНЫЙ ИНТЕРФЕЙС

При разработки приложения для лабораторной 5.7 будет использован код приложений для предыдущих лабораторных: 12.4 (Меню в главном окне приложения) и 12.6 (Редактирование табличных данных) для построения приложения с мультидокументным интерфейсом. Приложение должно показывать таблицы базы данных поставщиков из [13, с. 257], следующего содержания:

сами поставщики:

```
---- File:./date/date/s.csv

S1,Smith,20,London
S2,Jones,10,Paris
S3,Black,30,Paris
S4,Clark,20,London
S5,Adams,30,Athens

---- End Of File:./date/date/s.csv
```

детали, которые они поставляли:

```
---- File:./date/date/p.csv

P1,Nut,Red,12.0,London
P2,Bolt,Green,17.0,Paris
P3,Screw,Blue,17.0,Rome
P4,Screw,Red,14.0,London
P5,Cam,Blue,12.0,Paris
P6,Cog,Red,19.0,London

---- End Of File:./date/date/p.csv
```

проекты, для которых выполнялись поставки деталей:

```
---- File:./date/date/j.csv

J1,Sorter,Paris
J2,Display,Rome
J3,OCR,Athens
J4,Console,Athens
J5,RAID,London
J6,EDS,Oslo
J7,Tape,London

---- End Of File:./date/date/j.csv
```

сами поставки:

```
---- File:./date/date/spj.csv
```

```
S1,P1,J1,200
S1,P1,J4,700
S2,P3,J1,400
S2,P3,J2,200
S2,P3,J3,200
S2,P3,J4,500
S2,P3,J5,600
S2,P3,J6,400
S2,P3,J7,800
S2,P5,J2,100
S3,P3,J1,200
S3,P4,J2,500
S4,P6,J3,300
S4,P6,J7,300
S5,P2,J2,200
S5,P2,J4,100
S5,P5,J5,500
S5,P5,J7,100
S5,P6,J2,200
S5,P1,J4,100
S5,P3,J4,200
S5,P4,J4,800
S5,P5,J4,400
S5,P6,J4,500
```

```
---- End Of File:./date/date/spj.csv
```

Требуется убедиться, что в качестве разделителя и в программе и в файлах используется один и тот же символ. Итак, начинаем выполнять лабораторную:

- В новый каталог перекопируйте файлы из [16.2.5](#) - Program.cs - он будет подвергаться переделке и из [16.2.6](#) - файлы wnd.cs и wnd2.cs.
- в файл главного окна приложения Program.cs добавить подключение именованной области видимости двух последних файлов:

```
using dtGrd;
```

- В классе Program завести статическую переменную типа Form и в неё заполнить главное окно приложения:

```
static public Form mWin;
[STAThread]
static void Main(string[] args)
{
    mWin = new x();
    mWin.IsMdiContainer = true;
    Application.Run(mWin);
}
```

Тут был установлен признак контейнера мультидокументного интерфейса.

- Изменить меню главного окна приложения на следующее:

```
File      Date      Windows  About
Exit      Supplier
          Part
          Project
          Delivery
```

при этом:

```
MenuItem SupplierIt = new MenuItem("Supplier");
MenuItem PartIt     = new MenuItem("Part");
MenuItem ProjectIt  = new MenuItem("Project");
MenuItem DeliveryIt = new MenuItem("Delivery");
MenuItem WinIt      = new MenuItem("Windows");
```

Подробности создания главного меню приложения см. [16.2.5](#) .

- Добавим обработчики в событие Click новых элементов меню:

```
SupplierIt.Click += SupplierF;
PartIt    .Click += PartF    ;
ProjectIt .Click += ProjectF ;
DeliveryIt.Click += DeliveryF;
```

- Добавим четыре практически одинаковых обработчика:

```
void SupplierF(object sender, EventArgs a)
{
    Console.WriteLine("Поставщики");
    statStrip.Items[1].Text = "окно поставщиков не готово";
}
```

- Далее, в каждом из четырех обработчиков зададим свойство MdiParent:

```
z.MdiParent = Program.mWin;
```

и добавим в, приблизительно, таком виде:

```
Form z = new wnd2(      // выбрать папку из которой запущено приложение
    Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
    +"/p.csv");          // добавить файл поставок
z.Text  = "Детали";
z.Name  = "part";
statStrip.Items[1].Text = "Детали загружены";
z.Show();
```

- Элементу меню WinIt задать признак списка дочерних окон мультидокументного приложения:

```
WinIt.MdiList =true;
```

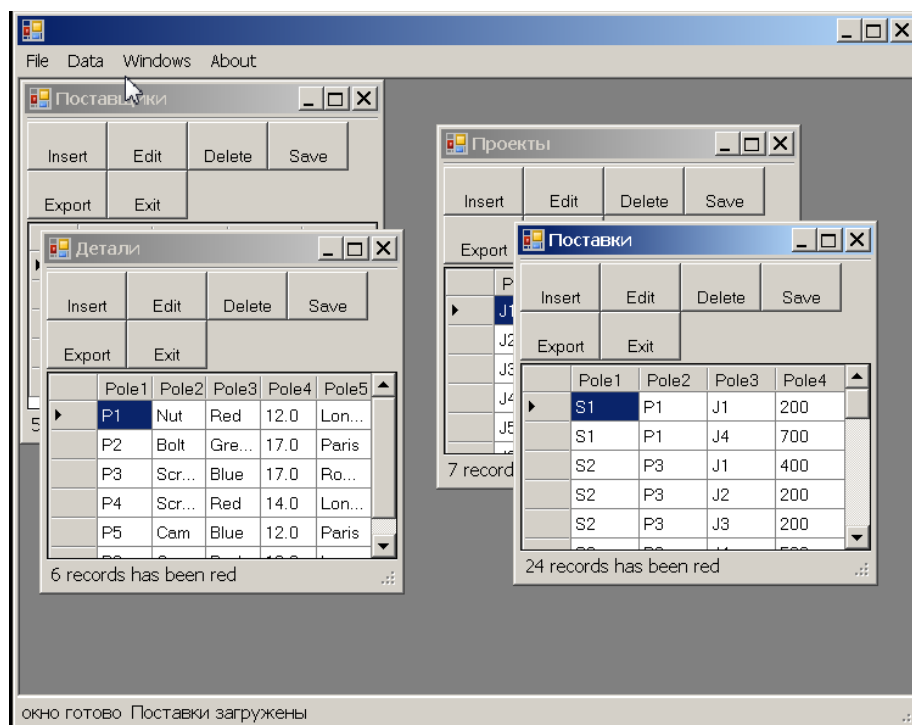
- Избавиться от возможности многократного открытия списка. В результате приложение будет иметь не более четырех открытых окон: одно - для поставщиков, одно - деталей и так далее. Для этого в класс главного окна приложения добавить метод

```
bool shouldIOpen (string text){
    for (int i = 0; i < MdiChildren.Count(); i++)
    {
        if (this.MdiChildren[i].Text == text)
        {
            MdiChildren[i].Activate();
            return false;
        }
    }
    return true;
}
```

- При помощи этого метода изменить обработчики элементов главного меню приложения так, чтобы они не открывали новые окна, а активизировали уже уже открытые. Продемонстрировано на обработчике окна проектов:

```
void ProjectF(object sender, EventArgs a)
{
    Console.WriteLine("Проекты");
    string id = "project";
    if (shouldIOpen(id)) {
        statStrip.Items[1].Text = "окно проектов не готово";
        Form z = new wnd2(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
            + "/j.csv");
        z.MdiParent = Program.mWin;
        z.Text = "Проекты";
        z.Name = id;
        z.Show();
    }
    statStrip.Items[1].Text = "Проекты загружены";
}
```

Усилия по созданию приложения должны закончиться чем то вроде:



Многодокументный интерфейс

Причем содержимое элемента Windows из главного меню будет меняться в зависимости от количества открытых окон. Построенная программа начинает напоминать приложение для ведения базы данных.

13 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ: ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММИРОВАНИЯ В .Net

13.1 ЛАБОРАТОРНАЯ РАБОТА: РЕСУРСЫ В ПРИЛОЖЕНИЯХ

Для выполнения работы 6.1 понадобится создать файл ресурсов, скомпилировать его, добавить к процессу построения приложения и продемонстрировать работу последнего. Под ресурсом в данном контексте понимается иконка, указатель мыши, картинка или тестовая строка. Их можно не включать в сборку и, если они нужны для работы приложения, тогда придется копировать их вместе с приложением. Из приложения доступ к таким ресурсам будет выливаться образом, обычным для чтения файлов. Иначе, их можно включить в сборку, тогда копировать нужно только, собственно, саму сборку, но доступ придется организовывать по другому, не через файлы, а через менеджер ресурсов.

13.1.1 Построение приложения с ресурсами

Для выполнения работы требуется построить файл ресурсов (picslst.resx). Это можно сделать несколькими способами. Например, в приложении на C# файлы ресурсов любых типов создаются объектами класса ResXResourceWriter. Объект создается, в него добавляются ресурсы, например, переменные типа Image (используемые для хранения картинок), объект сохраняется на жесткий диск в момент закрытия. файл ресурсов (1pic.resx) с одной картинкой (some.jpg) можно создать так:

```
ResXResourceWriter rw = new ResXResourceWriter("picslst.resx");
rw.AddResource("pic1" // имя ресурса !!!!
, Image.FromFile("some.jpg", true));
rw.Close();
```

Для создания файла с текстовыми ресурсами можно воспользоваться компилятором ресурсов resgen.exe, (см. <https://docs.microsoft.com/en-us/dotnet/framework/tools/resgen-exe-resource-file-generator>). Пусть файл 1.txt с описанием текстового ресурса ares имеет вид:

```
ares=блаблабла
```

Тексты кириллицы по умолчанию должны быть записаны в кодировке UTF-8 Тогда создание файла 1.resx выполняется следующим образом:

```
resgen 1.txt 1.resx
```

В выходном файле должна быть секция следующего вида:

```
<data name="ares" xml:space="preserve">
  <value>блаблабла</value>
</data>
```

Кроме того, для создания ресурсов из списка картинок можно использовать утилиту из раздела 15.17 .

Окончательная компиляция ресурсов для включения их в сборку (то есть, создание файла с расширением .resources) выполняется следующей командой:

```
---- File:./cs/resxgen/eyelst/_resgen.cmd
```

```
resgen.exe /usesourcepath /compile picslst.resx
```

```
---- End Of File:./cs/resxgen/eyelst/_resgen.cmd
```

Если компиляция ресурсов прошла без ошибок в каталоге должен появиться файл picslst.resources. Его надо подключить к сборке следующей командой:

```
csc /resource:picslst.resources /out:a.exe *.cs
```

Построенное приложение a.exe будет содержать исходные картинки внутри себя, в разделе ресурсы.

13.1.2 Рисование и доступ к ресурсам приложение

Доступ к ресурсам, которые обсуждались в предыдущем разделе, можно получить следующим образом

```
using System.Reflection;
class frmMain : Form {
    ...
    System.Resources.ResourceManager rm
        = new global::System.Resources.ResourceManager(
            "picslst"           // откомпилированный файл ресурсов
                               // подключенный к сборке
            , typeof(frmMain).Assembly); // класс с методом Main
    Image im = (Image)rm.GetObject("pic1"); // взять байты по имени ресурса !!!!
}
```

Для рисования картинки в окне, создать метод обработчик для перерисовки окна. Его имя должно быть OnPaint и параметр иметь тип PaintEventArgs:

```
protected override void OnPaint(PaintEventArgs ea) {
    Graphics grfx = ea.Graphics;
    grfx.DrawImage(im,
        10, 10, im.Width, im.Height);
}
```

Объект grfx (графика) это виртуальная поверхность, на которой методом DrawImage рисуется ранее прочитанная картинка im.

13.1.3 Таймер для поочередного отображение картинок

Пусть требуется поочередно отображать 4 картинки. Тогда требуется в области видимости класса создать и заполнить массив

```
Image[] arrImages = new Image[4];
```

В области видимости завести переменную - текущая картинка:

```
int intCurrentImage = 0;
```

Создать обработчик события таймера Tick:

```
void TimerOnTick(object obj , EventArgs ea ) {
    Invalidate();
    intCurrentImage ++;
    if (intCurrentImage >3 )
        intCurrentImage=0;
}
```

В котором метод Invalidate будет требовать перерисовку окна, то есть, неявно вызывать метод OnPaint. В методе OnPaint требуется рисовать одну из четырех картинок массива arrImages.

В конструкторе окна создать таймер и добавить к его событию обработчик:

```

tmrAnimation      = new System.Windows.Forms.Timer();
tmrAnimation.Enabled = true; // разрешить таймеру тикать
tmrAnimation.Interval = 500; // каждые полсекунды
tmrAnimation.Tick += TimerOnTick;
Load += new EventHandler(frmMain_Load);

```

Лабораторная окончена.

13.2 ЛАБОРАТОРНАЯ РАБОТА: ПРИМИТИВЫ ГРАФИКИ

Для выполнения лабораторной 6.2 координаты пикселей в окне будут задаваться целыми числами. При этом, предполагается, что чтение последовательности пар целых чисел не является проблемой, поэтому для создания метода

```

public Point []                                // чтение массива точек
getPoints(TextReader tr) {}

```

, который читает со стандартного ввода массив точек, не будет объяснений.

- Создать четыре файла: Program.cs, sLine.cs, p4sLine.cs и w4line.cs. Область видимости по умолчанию - line.
- В каждом файле именованную область видимости System.Drawing добавить к просмотру по умолчанию. Она содержит структуру Point - представляет упорядоченную пару целых чисел — координат X и Y, определяющую точку на двумерной плоскости.
- Класс панели p4sLine будет содержать линию экрана для рисования, переменную error с тестом ошибки и делегат - обработчик _paint для вывода сообщения об ошибке:

```

class p4sLine: Form
{
    sLine    sl ;           // линия экрана для рисования
    public p4sLine(sLine l) {
        mkPan( l);
    }
    protected
    void    mkPan(sLine l) {
        sl = l;
        if (l==null) {
            error = "no any line!";
            Paint += _paint;
        }
        else if (l.ps == null || l.ps.Length < 1) {
            error = String.Format("Line '{0}' is empty!", l.nm);
            Paint += _paint;
        }
        else { // тут добавить делегат обработчик для рисования линии
        }
        BackColor = Color.Ivory;
        Dock = DockStyle.Fill;
    }
}

```

```

    }
    // сообщение об ошибке
    public void
    _paint(object sender, PaintEventArgs e){
        // виртуальная плоскость на которой рисуют
        Graphics g = e.Graphics;
        // шрифт
        using (Font f = new Font("Times New Roman", 14)){
            g.DrawString      // нарисовать
            ( error           // этот текст
            ,f                 // этим шрифтом
            ,Brushes.Red      // красной кисточкой
            ,10 ,3* 14        // в таких координатах x и y
            );
        }
    }
}

```

Ордината Y на виртуальной плоскости увеличивается от верхнего края окна к нижнему. Конструктор вызывает специальный метод инициирования панели `mkPan`, так как в будущем его придется вызывать из конструктора наследника после (а не до) действий по инициализации объекта наследника.

- Класс линии экрана (`sLine`), пока пустой, точнее, содержит обсужденный выше метод `getPoints` и название линии:

```

class sLine {
    public string nm = "line";
}

```

- Класс окна `w4line` содержит единственный управляющий элемент - панель `p4sLine` - для рисования линии экрана.

```

class w4line: Form
{
    p4sLine pan;
    public w4line (sLine ln) {
        Padding = new Padding(10);
        pan = new p4sLine (ln);
        Controls.Add (pan);
    }
}

```

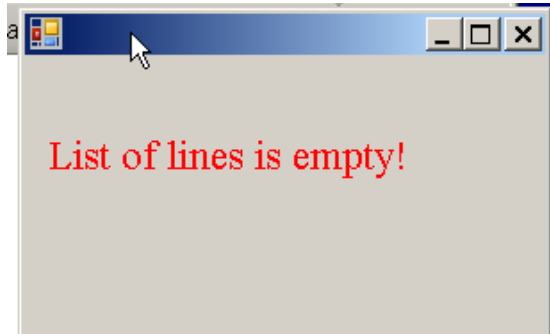
- Статический метод `Main` класса `Program`:

```

static void Main()
{
    sLine ln = new sLine();
    ln.nm = "test line";
    Application.Run(new w4line(ln)); // показали её
}

```

Построит и покажет окно w4line с сообщением об ошибке:



Вывод текста

- Далее, добавить в класс sLine - линии экрана массив точек для рисования - ps, умолчательный карандаш - pen, два признака рисовать незамкнутый набор линий (трек) - mkLine или рисовать замкнутый полигон - mkPolygon.

```
class sLine {
    public string nm      = "line";
    public Point[] ps;    ///<Координаты точек линии в пикселях
    public Pen pen       = Pens.Black;    ///<Карандаш для основного рисования
    public bool mkPolygon = false;
    public bool mkLine   = true;
}
```

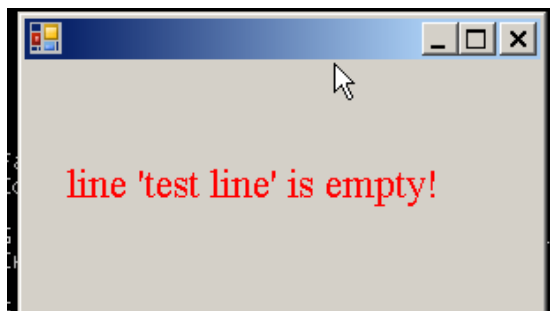
- Далее, добавить в класс sLine делегат - обработчик события Paint, который сообщает об ошибке, при отсутствии точек в линии:

```
// рисование линии
public void _paint (object sender, PaintEventArgs e){
    Graphics g = e.Graphics;
    if (ps != null && ps.Length > 1) {
        if (mkPolygon)
            g.DrawPolygon(pen, ps);    // замкнутая линия
        else
            g.DrawLines(pen, ps);
    }
    else {
        string text = String.Format("line '{0}' is empty!", nm);
        using (Font f = new Font("Times New Roman", 14)){
            g.DrawString( text,f
                , Brushes.Red
                , 20 , 3* 14
            );
        }
    }
}
```

- Далее, изменить в классе `p4sLine` конструктор, добавив в последнюю ветку делегат обработчик (подписать метод на событие) `sl._paint` в событие `Paint`:

```
else { // тут добавить делегат обработчик для рисования линии
    sl = 1;
    Paint += sl._paint; // делегат - обработчик линии
}
```

Сообщение об ошибке должно измениться на следующее:



Еще один вывод текста

- Разработать и выполнить в `Main` метод `getPoints`

```
ln.getPoints(Console.In); // прочитали линию
```

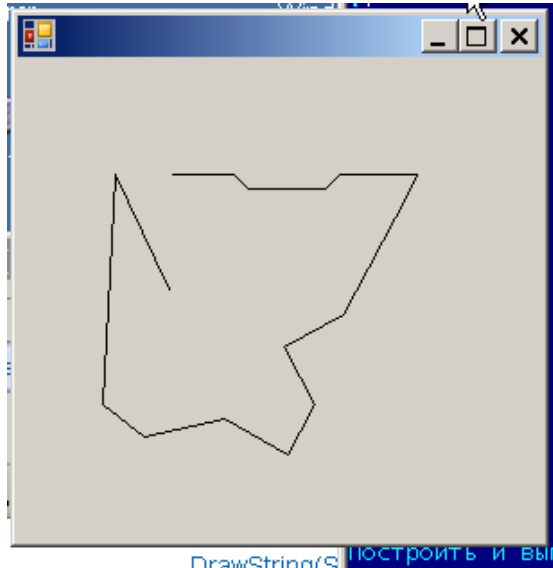
- Набрать файл `1.txt` типа такого (отдаленно напоминает перевернутого слоника):

```
86 64
120 64
128 72
171 72
179 64
222 64
210 88
181 142
148 160
165 192
150 220
115 200
70 210
47 192
54 64
84 128
```

- Построить и выполнить приложение:

```
Program.exe <1.txt
```

Приложение покажет нечто вроде:



Вывод ломаной

- Добавить возможность подписывать точки ломаной самостоятельно, используя метод `DrawString`, попадавший выше.

13.3 ЛАБОРАТОРНАЯ РАБОТА: МАСШТАБИРОВАНИЕ ЛОМАННОЙ

Приложение из прошлой лабораторной работы [13.2](#) выполняет рисование немасштабированной ломаной. Для масштабирования этой ломаной приложение придется существенно доработать. Основная идея этих доработок заключается в том, что, собственно, делегаты обработчики события `Paint` должны работать быстро и, значит, не заниматься никакой другой работой кроме рисования. Это приводит к необходимости выполнять отображение реальных данных (линия типа `rLine`, хранящая данные, например, в градусах МГС84, должна иметь 6 и даже 8 разрядов после запятой в каждом числе (`double`)) в данные, готовые для отображения методами `.Net`, например, `DrawPolygon`. Таким образом, каждое нажатие на кнопки будет приводить к изменению параметров отображения `mapping`, что в свою очередь влечет необходимость пересчета линии экрана и, затем, к вызову метода `Invalidate`, который, в свою очередь, приводит к событию `Paint`. Кроме того, должно быть понятно, что само отображение `mapping` сильно связано с размерами панели, на которой производится рисование линии и этот факт приводит к необходимости создавать класс наследник управляющего элемента `Panel`, который будет содержать класс для отображения и будет заботиться о данных в линии экрана `sLine`.

Все выше сказанное объясняет зачем приходится создавать класс для пары вещественных чисел - `tuple2d`, класс для хранения линии с данными из реальности - `rLine`, класс для отображения точек линии реальности в пиксели линии

экрана - mapping (он и будет выполнять основную функциональность), класс `p4sLine2` (наследник `p4sLine`), который будет заниматься отображением всей линии реальности в линию экрана после изменения масштаба или положения центра окна.

Итак, для разработки приложения надо:

- Скопировать в новый каталог из каталога лабораторной рисования немасштабированной прямой все её файлы;
- Добавить файл `tuple.cs`:

```
class tuple2d {          ///  
    public double X;      ///  
    public double Y;      ///  
    public tuple2d (double x = 0.0, double y = 0.0){  
        X = x;  
        Y = y;  
    }  
    public tuple2d (string x, string y ){  
        X = double.Parse(x);  
        Y = double.Parse(y);  
    }  
}
```

который будет содержать пары координат из реальности.

- Добавить файл `rLine.cs` с областью видимости `System.Threading`:

```
class rLine {          ///  
    public tuple2d[] pnts;          ///  
    public string nm;              ///  
    static  
    readonly char NumberDecimalSeparator  
        = Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator[0];  
    public rLine (string nm, string nordNm="", string eastNm=""){  
        this.nm = nm;  
    }  
    public static tuple2d []          ///  
    getPoints(TextReader tr, bool vFlag=false, char decPnt = '.')  
    {  
        return null;  
    }  
}
```

Свойство `NumberDecimalSeparator` требуется для последующей правильной обработки данных в которых вещественные числа могут содержать десятичную точку или десятичную запятую, в зависимости от настроек умолчательной культуры, установленной на компьютере.

- Содержимое метода `Main` поменять на следующее:

```
rLine ln = new rLine("test real line");  
ln.pnts = rLine.getPoints(Console.In, true);    ///  
Application.Run(new w4line(ln)); // показали её
```


- Полностью изменить класс w4line:

```
class w4line: Form
{
    rLine    rl;
    public w4line (rLine l) {
        rl = l;
        Padding = new Padding(10);
    }
}
```

Теперь приложение компилируется и показывает пустое окно.

- Создать файл p4sLine2.cs с классом наследником класса p4sLine:

```
class p4sLine2: p4sLine
{
    public p4sLine2(rLine l)
    {
    }
}
```

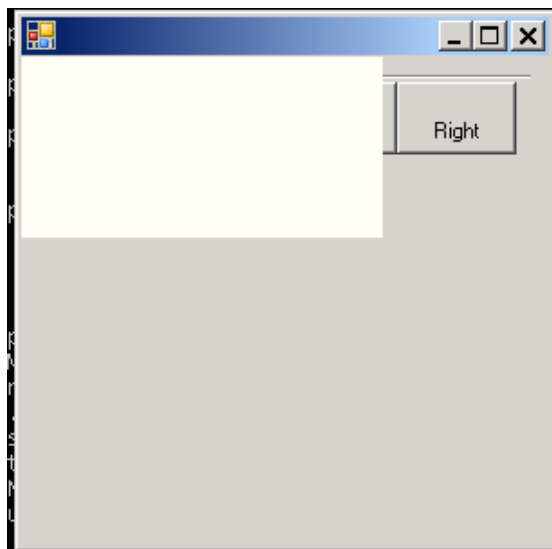
- Добавить в класс окна управляющий элемент полосу инструментов и панель для рисования:

```
ToolBar tb = new ToolBar();
p4sLine2 pan;
```

- В конструкторе окна к полосе инструментов добавить кнопки, создать панель и пока не вызывать делегат обработчик нажатия на полосу инструментов:

```
public w4rLine(rLine l)
{
    rl = l;
    Padding = new Padding(10);
    AutoSize = true;
    StartPosition = FormStartPosition.CenterScreen;
//
    pan = new p4sLine2(rl);
    Controls.Add(pan);
//
    tb.ButtonSize = new System.Drawing.Size((int)(200/ 3), (int)(40) );
    ToolBarButton Zoom_p = new ToolBarButton("Zoom +");
    ToolBarButton Zoom_m = new ToolBarButton("Zoom -");
    ToolBarButton Left = new ToolBarButton("Left");
    ToolBarButton Right = new ToolBarButton("Right");
    ToolBarButton Up = new ToolBarButton("Up");
    ToolBarButton Down = new ToolBarButton("Down");
    tb.Buttons.AddRange(new ToolBarButton[]
    {Zoom_p,Zoom_m,Left,Right,Up, Down });
    tb.Dock = DockStyle.Top;
//
    tb.ButtonClick += ToolBarButtonClick; !!!!! делегат закоментарен
    Controls.Add(tb);
```

Пока должно получиться не слишком красиво:



Плохое окно с полосой инструментов

- Далее, создать файл mapping:

```
public class mapping
{
    int l, r,    // границы окна
        t, b;   // b верх, t низ

    bool isWellformed = false;
    public double  xMin, xMax, yMin, yMax;
        // границы ломаной в реальности
        // изменяя эти значения можно добиться эффекта смещения
        // или изменения масштаба
    public mapping( ):this(0.0, 1.0, 0.0, 1.0){
        this.l = 0;
        this.b = 0;
        this.r = 255;
        this.t = 255;
    }
    public mapping( double x, double X, double y, double Y)
    {
        xMin = x;
        xMax = X;
        yMin = y;
        yMax = Y;
        if (xMin == xMax) // числа должны быть разными чтобы избежать деления на ноль
            xMax = xMin + 0.01;
        if (yMin == yMax)
            yMax = yMin + 0.01;
        this.l = 1;
        this.b = 1;
    }
}
```

```

        this.r = 400;
        this.t = 300;
        isWellformed = true;
    }
    public int w {
        get {return r+l;}    // поле размера l слева и справа
    }
    public int h {
        get {return t+b;}    // поле размера b сверху и снизу
    }
}

```

Где поля xMin, xMax, yMin, yMax будут играть важную роль в отображении ломаной из реальности в пиксели. Это координаты углов минимального прямоугольника, содержащий исходную линию.

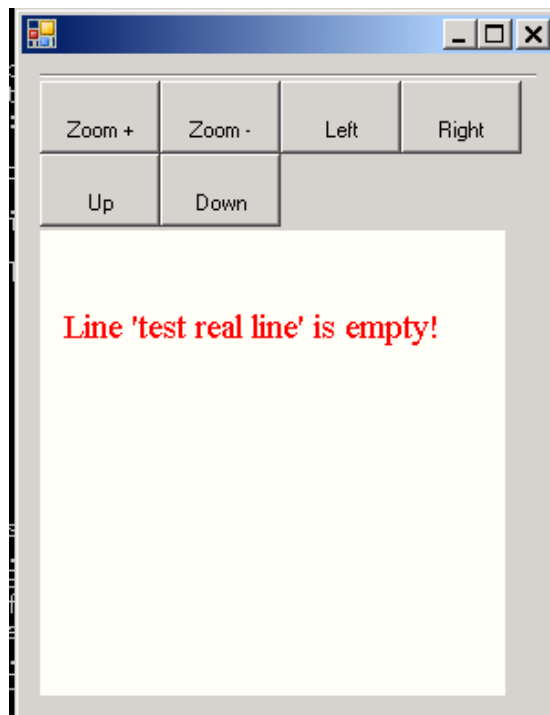
- Теперь можно доработать окно панель для ломаной p4sLine2:

```

class p4sLine2: p4sLine
{
    public
    mapping mp;    // панель будет содержать отображение для масштабирования и сдвига
    public p4sLine2(rLine l)
    {
        mp = new mapping();
        // оператор не сможет изменять размер окна
        MinimumSize = new Size(Convert.ToInt32(mp.w),Convert.ToInt32( mp.h));
        MaximumSize = new Size(Convert.ToInt32(mp.w),Convert.ToInt32( mp.h));
        AutoSize = true;
        if (l==null)
        ;
        else if (l.pnts == null || l.pnts.Length < 1)  {
            sl = new sLine();    // в этом случае
            sl.nm = l.nm;        // еще нечего рисовать
        }
        else {
            sl = new sLine();
            sl.nm = l.nm;        // уже можно высчитать линию экрана
            // <--- например, в этом месте тут
            Paint += sl._paint;  // делегат - обработчик линии
        }
        mkPan(sl);
    }
}

```

Теперь окно приняло приличный вид, вроде такого:



Хорошее маленькое окно с полосой инструментов

- Можно начинать чтение данных. Метод `getPoints` как и в 16.1.8 при чтении целых не должен представлять трудности, но требуется не забыть заменять заданный символ десятичной точки на символ принятый в текущей настройке:

```
public static tuple2d []           // чтение массива точек
getPoints(TextReader tr, bool vFlag=false, char decPnt = '.')
{
    List <tuple2d> ps = new List <tuple2d> ();
    ....
    for ( ; (r = tr.ReadLine())!=null; lineNo++){
        if (NumberDecimalSeparator != decPnt)
            r = r.Replace(decPnt, NumberDecimalSeparator);
        ....
    }
    ...
    return ps.ToArray();
}
```

Данные прочитаны, их надо использовать при создании панели `p4sLine2`.

- Из прочитанной линии получить крайние (левые, правые, верхние, нижние) значения обеих координат:

```
class rLine {    ///< линия реальности
    public bool getBox (out double xMin, out double xMax, out double yMin, out double yMax)
```

```

        xMin = double.MaxValue ;
        xMax = double.MinValue ;
        yMin = double.MaxValue ;
        yMax = double.MinValue ;
        if (pnts !=null && pnts.Length > 0){
            for (int i=0; i< pnts.Length; i++){
                if (pnts[i].X < xMin)
                    xMin = pnts[i].X;
                if (pnts[i].X > xMax)
                    xMax = pnts[i].X;
                if (pnts[i].Y < yMin)
                    yMin = pnts[i].Y;
                if (pnts[i].Y > yMax)
                    yMax = pnts[i].Y;
            }
            return true;
        }
        return false;
    }
}

```

- Их нужно использовать при создании панели p4sLine2 для рисования ломаной:

```

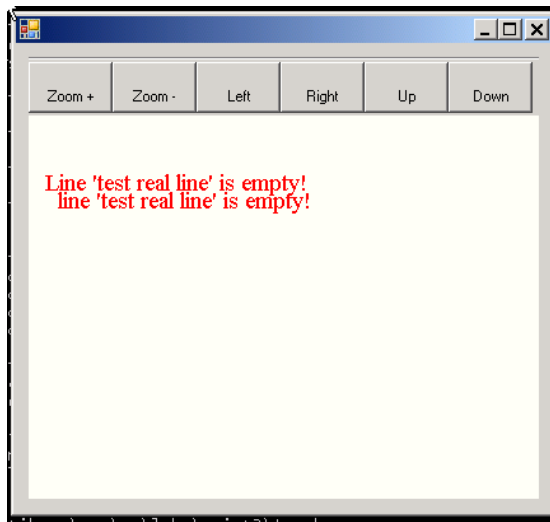
public p4sLine2(rLine l)
{
    double a,b,c,d;
    if (l.getBox(out a, out b, out c, out d)){
        mp = new mapping(a, b, c, d);
        // mp.mkZmEqual();// это метод для того, чтобы масштаб по обеим осям
        // был одинаковый
    }
    else
        mp = new mapping(); // эта строчка уже была
    ...
}

```

Теперь построенное приложение, например a.exe, по команде:

```
a.exe <1.txt
```

Где 1.txt - файл из лабораторной [16.1.8](#) , должно показать большее окно размером 400 на 300 пикселей:



Хорошее большое окно с полосой инструментов

Это означает, что можно приступать к рисованию ломаной.

- В класс `p4sLine2` добавить метод `mkSLine` для создания линии экрана из линии реальности:

```
public void
mkSLine(rLine rl){
    int x, y;
    sl.ps = new Point[rl.pnts.Length];
    for (int i = 0; i < rl.pnts.Length; i++){
        mp.map (rl.pnts[i].X, rl.pnts[i].Y // входные координаты
              , out x, out y);           // выходные координаты на экране
        sl.ps[i] = new Point (x, mp.h - y); // добавить их в линию и перевернуть
    }
}
```

Замечание

`mp.h - y` : создает эффект перевертывания ломаной, надо выполнять вследствие того что левый верхний угол панели для рисования имеет координату (0,0) и значение ординаты увеличивается вниз.

- Применить его в конструкторе панели `p4sLine2`:

```
sl = new sLine();
sl.nm = l.nm;
mkSLine(l);           /// <--- применение
Paint += sl._paint;   // делегат - обработчик линии
```

- В класс отображение `mapping` добавить, собственно, само отображение:

```
public
void map (double X, double Y,           out int x, out int y){
```

```

    if (isWellformed) {
        x = Convert.ToInt32(Math.Round((X - xMin) / (xMax - xMin) * (r-l)+l));
        y = Convert.ToInt32(Math.Round((Y - yMin) / (yMax - yMin) * (t-b)+b));
    }
    else {
        x=0; y=0;
    }
}

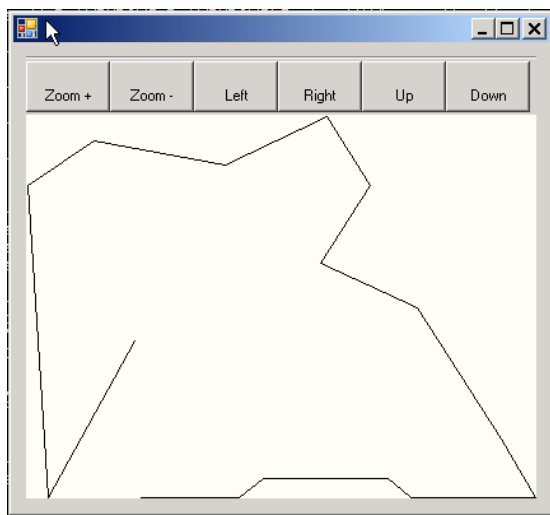
```

Где данное отображение выводится из пропорций:

$$\begin{aligned} (X - xMin) / (xMax - xMin) &= (x - l) / (r-l) \\ (Y - yMin) / (yMax - yMin) &= (y - b) / (t-b) \end{aligned}$$

Меняя xMax и xMin можно добиться изменения масштаба или сдвига по оси слева - направо.

Теперь линию экрана должно быть видно:



Ломаная линия

- В класс отображение mapping добавить два метода позволяющие выполнять смещение по осям слева - направо и сверху - вниз:

```

public void moveNordSouth (
    int shift          ///< проценты сдвига области видимости, >0 -- на север, <0 -- на юг
){
    if (isWellformed) {
        double percent = (yMax - yMin) / 100;
        yMax+= percent*shift;
        yMin+= percent*shift;
    }
}

public void moveEastWest (

```

```

        int shift          ///< проценты сдвига области видимости, >0 -- на север, <0 -- на юг
    ){
    if (isWellformed) {
        double percent = (xMax - xMin) / 100;
        xMax+= percent*shift;
        xMin+= percent*shift;
    }
}

```

- В окно w4line добавить делегат - обработчик щелчка по панели инструментов ToolBarButtonClick:

```

void ToolBarButtonClick(object sender, ToolBarButtonClickEventArgs e)
{
    if (e.Button.Text == "Zoom +")          // поменять параметры отображения
        ;//pan.mp.zoom ( -10 );
    else if (e.Button.Text == "Zoom -")
        ;//pan.mp.zoom ( 10 );
    else if (e.Button.Text == "Right")
        pan.mp.moveEastWest ( 20 );
    else if (e.Button.Text == "Left")
        pan.mp.moveEastWest ( -20 );
    else if (e.Button.Text == "Down")
        pan.mp.moveNordSouth ( -20);
    else if (e.Button.Text == "Up")
        pan.mp.moveNordSouth ( 20);
    else
        return;
    pan.mkSLine(rl);                        // пересчитать линию экрана
    pan.Invalidate();                       // вызвать событие Paint
}

```

И раскомментировать его применение в конструкторе окна:

```
tb.ButtonClick += ToolBarButtonClick; !!!!! делегат закоментарен
```

Приложение должно реагировать при нажатии на кнопки влево, вправо, вверх, вниз. Понятно, что для движения оба крайних значения одной из осей координат менялись в одну сторону. Например в методе moveNordSouth:

```

yMax+= percent*shift;
yMin+= percent*shift;

```

Значит для изменения масштаба в еще не созданном методе zoom надо меньшие границы уменьшать, а большие границы - увеличивать (и наоборот), вроде этого:

```

yMax+= percentY*shift;
yMin-= percentY*shift;
xMax+= percentX*shift;
xMin-= percentX*shift;

```

Осталось добавить метод mkZmEqual, чтобы масштабы по осям вниз - вверх и слева - направо были одинаковыми.

13.3.0.1 Использование Doxygen для документирования кода.
В качестве иллюстрации к использованию Doxygen построим схему зависимости классов полученных при выполнении лабораторной.

- Подготовить файл конфигурации Doxygen-а согласно [16] в каталоге лабораторной.

- В него добавить строчку

```
DOXYFILE_ENCODING      = CP1251
```

- строчки

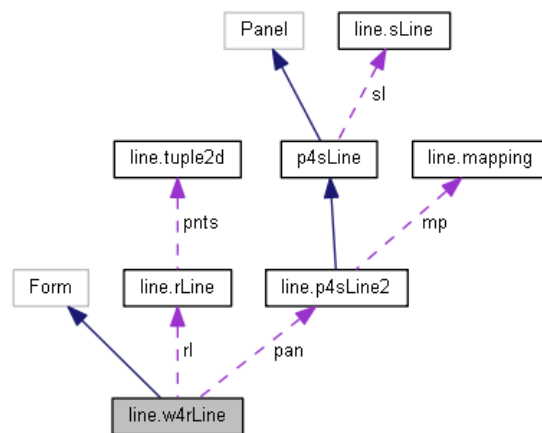
```
INPUT                  = ./csharp/  
GENERATE_LATEX         = YES
```

поменять на

```
INPUT                  = .  
GENERATE_LATEX         = NO
```

чтобы не генерировать документацию на Латехе и файлы для отчета брать из текущего каталога.

После построения документации файл classline_1_1w4r_line__coll__graph.png из каталога ./_bld/html будет представлять нечто вроде этого:



Граф зависимостей классов

На графе сплошной линией отмечается наследование между классами, а прерывистой - включение поля одного класса в другой. Имя поля написано возле линии.

13.4 ЛАБОРАТОРНАЯ РАБОТА: ИСПОЛЬЗОВАНИЕ МЫШИ

Для выполнения лабораторной 6.4 надо скопировать исходный код разработанного в 13.3 приложения в новый каталог. Далее излагается использование мыши для создания контекстного меню.

13.4.1 Введение в использование мыши

События (тип - делегат), которые будут использоваться в приложении для работы с мышью следующие: нажатие клавиши, перемещение указателя, отпускание клавиши. Им соответствуют следующие делегаты класса Form: `MouseDown`, `MouseMove`, `MouseUp`. В аргументе делегата - обработчика типа `MouseEventArgs` понадобится поле, содержащее координаты указателя мыши - `Location`. Функциональность для работы с мышью будет добавляться в наследника класса `w4rLine`.

- В нем потребуется открыть доступ к полям - реальная линия `rl` типа `rLine` и панель рисования линии `pan` типа `p4sLine2`.
- Допisać к ним ключевое слово `protected`:

```
protected rLine rl;
protected p4sLine2 pan;
```

- Создать класс `msWnd`, наследник `w4rLine`:

```
class msWnd: w4rLine
{
    public msWnd(rLine l): base(l) {
    }
}
```

- В класс `msWnd` добавить две переменные :

```
Boolean mouseDown ; // запомнили, что клавиша мышки зажата
PointF MousePoint1; // координаты мышки в момент зажатия клавиши
```

- Допisać два делегата - обработчика нажатий на клавиши мышки:

```
void Form1_MouseDown(object sender, MouseEventArgs e) {
    MousePoint1 = e.Location;
    mouseDown = true;    // клавиша нажата
}
Form1_MouseUp(object sender, MouseEventArgs e) {
    mouseDown = false;    // клавиша отпущена
}
```

- В конструкторе окна добавить их к соответствующим событиям:

```
public msWnd(rLine l): base(l) {
    mouseDown = false;
    pan.MouseDown += Form1_MouseDown;
    pan.MouseUp += Form1_MouseUp;
}
```

- Добавить обработчик для пересчета линии при движении:

```

void
Form1_MouseMove(object sender, MouseEventArgs e){
    if (mouseDown){
        PointF MousePoint2 = e.Location;    // текущее положение указателя
        pan.mp.moveEastWest
            ( (int)(        // высчитать процент от смещения по экрану
                (MousePoint1.X - MousePoint2.X )*100)/ClientSize.Width
            );
        pan.mp.moveNordSouth    // тут перевернута система координат
            (
                - (((int) ( MousePoint1.Y - MousePoint2.Y ))*100)/ClientSize.Height
            );
        pan.mkSLine(rl); // при больших вычислительных затратах
        pan.Invalidate(); // эти строчки можно переместить в Form1_MouseUp
        MousePoint1 = e.Location;
        // линию пересчитали, в след. раз пересчитать относительно этой точки
    }
}

```

13.4.2 Контекстное меню

Для создания управляющих элементов контекстное меню использует два набора классов: ContextMenu и MenuItem; второй - ContextMenuStrip и ToolStripMenuItem. Меню создавать при нажатии на левую клавишу мыши (клик):

```

// MouseClick += Control1_MouseClick; // где-то в конструкторе окна
void
Control1_MouseClick(Object sender, MouseEventArgs e)    {
    Region r = mkRgn();    // построить регион,
                           // то есть, множество пикселей, рядом с ломаной
    if (r.IsVisible(Cursor.Position)) // попала точка в регион?
    {
        ContextMenuStrip.Items.Clear(); //удалить элементы меню
        ToolStripMenuItem mi    // создать новый элемент меню
            = new ToolStripMenuItem("in Region");
        ContextMenuStrip.Items.Add(mi); // добавить его в контекстное меню
        mi.Click +=
            ContextMenuStrip_Click;    // добавить его обработчик
    }
    else
    {
        ContextMenuStrip.Items.Clear();
        ToolStripMenuItem mi = new ToolStripMenuItem("Not in Region");
        ContextMenuStrip.Items.Add(mi);
        mi.Click += ContextMenuStrip_Click;
    }
}
void
ContextMenuStrip_Click(object sender, EventArgs e){
    Console.WriteLine("MenuItem have been clicked");
}

```

Объект типа Region (внутренняя часть геометрической фигуры, которая состоит из прямоугольников и контуров) имеет различные методы для проверки попадания некоторого прямоугольника (или точки) во внутрь региона - IsVisible. Метод был применен выше. Собственно, сам регион создается из составного контура (объекта типа GraphicsPath). Последний строится последовательным добавлением некоторых многоугольников, которые описываются возле каждого ребра ломаной. Например, так:

```
Point []
mkPolygon (Point a, Point b){
    Point[] m = new Point[5];
    m[0] = a;
    m[1] = b;
    m[2]      // чуть-чуть отошли от точки b
    = new Point(b.X + 4, b.Y);
    m[3]      // чуть-чуть не дошли до точки a
    = new Point(a.X + 4, a.Y);
    m[4] = a; // замкнули многоугольник
    return m;
}
```

Тогда построение искомого региона mkRgn может выглядеть так:

```
Region
mkRgn(){
    GraphicsPath gp = new GraphicsPath();
    for (int i = 0; i < points.Length - 1; i++) {
        gp.AddPolygon(mkPolygon(points[i]
                                , points[i+1]
                                )
        );
    }
    return new Region(gp);
}
```

Законченный пример приложения в разделе [15.6](#) меняет контекстное меню, после каждого клика левой клавишей, в зависимости от попадания указателя мыши в область рядом с ломаной.

13.5 ЛАБОРАТОРНАЯ РАБОТА: СКАЧИВАНИЕ ФАЙЛОВ

Пространство имен System.Net предоставляет простой программный интерфейс для многих протоколов, используемых в компьютерных сетях. Знакомство можно начинать со следующих классов: HttpRequest (и его родитель WebRequest)- выполняет запрос к унифицированному (единообразному) идентификатору ресурса (англ. URI Uniform Resource Identifier); HttpResponse - предоставляет ответ от унифицированного идентификатора ресурса; WebException - используется для описания исключительных состояний, возникающих при появлении ошибок во время доступа приложения к компьютерной сети; HttpStatusCode - перечисление содержит значения кодов состояний, определенных для протокола HTTP. При этом, для создания новых объектов типа HttpRequest, надо использовать статический метод Create родительского класса WebRequest.

Далее, при помощи метода `GetResponse` требуется получать объект класса `HttpWebResponse` и, затем, при помощи метода `GetResponseStream` получать объект класса `Stream`. Последний объект и предоставляет низкоуровневый доступ к байтам, которые возвращает запрошенный унифицированный идентификатор ресурсов.

В разделе 15.7 можно увидеть пример скачивания начальной страницы поисковика Google.

13.6 ЛАБОРАТОРНАЯ РАБОТА: ЖУРНАЛИРОВАНИЕ РАБОТЫ ПРИЛОЖЕНИЯ

Для выполнения лабораторной требуется гарантировать:

- попадание строчек в файл журнала в том порядке, в котором потоки обращались к методу вывода сообщений в журнал (аналог метода `WriteLine`);
- что обращение к методу вывода сообщений выполнялось бы быстрее, чем обращение к методу `WriteLine` из класса `StreamWriter`.

Решением является использование очереди сообщений. Очередь `Queue` - это еще один класс из пространства коллекций `System.Collections`. Её особенность заключается в том, что методы добавления в очередь - `Enqueue` и удаления из очереди - `Dequeue` работают по правилу - "Первый пришел, первый ушел". Это правило означает, что элементы очереди извлекаются из неё в той же последовательности, в которой они в неё заносились. Метод для постановки сообщений в очередь будет вызываться из нити (пользователя), а для извлечения сообщений из очереди и вывода их в файл будет использоваться специальная нить, работающая с низким приоритетом. Она будет извлекать сообщения и выводить их по мере возможности, то есть, тогда когда не будут работать пользовательские нити. Кроме того, сообщениям принято сортировать по признаку важности, что позволяет гибко управлять количеством выводимых сообщений. Итак, нужно:

- создать перечисление вроде следующего:

```
public enum IMPORTANCELEVEL {
    Debug        // отладочные сообщения
    , Warning     // предупреждения - не важные сообщения
    , Error       // ошибки приложения - важные сообщения,
    , FatalError  // катастрофические ошибки,
                  // делающие невозможным функционирование приложения
};
```

- создать класс журнала:

```
public class Logger : IDisposable
{
    bool dbg        = false;           //закрывать-открывать файл после каждого вывода
    StreamWriter sw  = null;           // сюда выводятся сообщения
    Queue<String> queue = null;        // Очередь сообщений
}
```

```

// Установленный уровень важности (сообщения ниже установленного уровня - игнорируются)
// уровень важности надо задавать в конструкторе, при создании журнала
IMPORTANCELEVEL ImportanceLevel = IMPORTANCELEVEL.Error;
// Поток вывода сообщений в файл
Thread Log;
// Bool для выхода из бесконечного цикла в потоке логера
bool Working = false;
}

```

Понятно, что наследование от `IDisposable`, связано с файлом вывода `StreamWriter sw` и инструкцией `using`. Эти особенности обсуждались ранее и больше обсуждаться не будут.

- создать метод, который занимается выводом сообщений в файл:

```

void LogMessage()
{
    string p=null;
    // Бесконечный цикл вывода сообщений раз в секунду
    while (Working)
    {
        // Вывод всех сообщений, если есть
        lock (this)
        {
            if (StringQueue.Count > 0)
            {
                p = (string)StringQueue.Dequeue();
                sw.WriteLine(p);
                // в очень проблемных случаях приходится закрывать файл
                // sw после вывода каждого сообщения,
                // чтобы увидеть сообщения, которые приложение не успело
                // сбросить в файл перед крахом
                if (dbg) {
                    sw.Close();
                    sw = new StreamWriter(filename, true);
                }
                p=null;
            }
            else {
                // если сообщений нет, то можно приостановить цикл.
                Thread.Sleep(10);
            }
        }
    }
}

```

- Конструктор `Logger`-а может содержать код вроде такого:

```

this.StringQueue = new Queue();
this.sw = new StreamWriter("logger.txt", true);
this.working = true;
this.Log = new Thread(new System.Threading.ThreadStart(LogMessage));
this.Log.Priority = ThreadPriority.Lowest;
this.Log.Start();

```

Нити могут быть основные и фоновые (свойство `IsBackground`). Метод `Main` будет ждать завершения основных нитей и оборвет работу фоновых (при этом не гарантируется выполнение операторов из секций `finally` фоновых нитей)

- создать метод для завершения работы и освобождения файла:

```
public void Dispose() {
    string p=null;
    // Пытаемся остановить логер
    this.Working = false;
    if (Log != null)
        Log.Join();
    // затем выводим остаток очереди в этом методе
    while (StringQueue.Count > 0) {
        // Если есть еще не выведенные сообщения,
        p = (string)StringQueue.Dequeue(); //!!!!
        sw.WriteLine(p);
        p=null;
    }
    sw.Close();
}
```

Работать будет только текущая нить, поэтому очередь не блокируется инструкцией lock;

- и осталось написать метод для журналирования сообщений:

```
void WriteLine(
    IMPORTANCELEVEL Importance // важность данного сообщения
    , String Format             // строка с форматом сообщения
    , params object[] Segments // параметры сообщения
){
    // Проверка уровня важности сообщения
    if (Working && Importance >= this.ImportanceLevel) {
        // Форматирование сообщения
        String Message =
            String.Format("[{0}]\t", Importance) + "\t" + string.Format(Format, Segments);
        // Добавление сообщения в очередь
        lock (this) {
            StringQueue.Enqueue(Message);
        }
    }
}
```

Вообще говоря, на этом работа закончена. Желательно, чтобы имя журнала было не logger.txt, а строилось из имени приложения. Например, если приложение называется app.exe, то журнал будет называться app.txt. Кроме того, можно использовать не класс Queue, а шаблон очереди Queue<T>.

```
Queue<string> StringQueue = new Queue<string>();
```

что позволит не заботиться о преобразовании типа и быть уверенным в отсутствие ошибок при извлечении объекта из очереди смотри, например метод Dispose:

```
p = StringQueue.Dequeue(); //!!!!
```

Осталось написать приложение хотя бы из двух потоков, которое протестирует использование разработанного класса журналирования. Пример выполнения операции инкремента в двух различных нитях с комментариями, равно как и нескольких способов синхронизации различных нитей (естественно, без использования журналирования) приведен в разделе [15.13](#).

С законченным примером системы журналирования и приложением из двух нитей, которое его использует можно ознакомиться по адресу [\[20\]](#).

14 ОТВЕТЫ

Раздел предназначен для накопления ответов на вопросы и будет использоваться для написания лекций.

14.1 Простые Вопросы

Ниже приводится текст приложения для выполнения заданий из простых вопросов.

```
---- File:./cs/app.cs

#pragma warning disable 219

using System;

class application1{

    static void Main() {
        int    b    = 2;
        int    i    = 0;
        double f    = 0.0;
        string s    = "none";
        string q    = "none";

    #if Q1
        q= "длина массива";

        int [,] a = new int [3,4];
        i = a.Length;

    #elif Q2
        q= "размерность массива";

        int [,] a = new int [3,4];
        i = a.Rank;

    #elif Q3
        q= "длина первой размерности";

        int [,] a = new int [3,4];
        i = a.GetLength(0);
```



```
#elif Q4
    q= "длина второй размерности";

    int [,] a = new int [3,4];
    i = a.GetLength(1);

#elif Q5
    q= "инкремент после";

    i = b++;

#elif Q6
    q= "инкремент до";
    i = ++b;

#elif Q7
    q= "сложение + инкремент после";
    i = b + b++;

#elif Q8
    q= " инкремент после и сложение ";
    i = b++ + b;

#elif Q9
    q= " увеличить на ";
    i += b;

#elif Q10
    q= " уменьшить на ";
    i -= b;

#elif Q11
    q= " целое деление";
    i = 1 / 2;

#elif Q12
    q= " остаток от деления";
    i = 1 % 2;

#elif Q13
    q= " модуль плавающего";
    f = -1.0;
    f = Math.Abs(f);

#elif Q14
    q= " округление плавающего";
    f = -3.14159;
    f = Math.Round(f);
    i = (int)f;

#elif Q15
    q= " возведение -2 в степень 2";
    f = -2.0;
    f = Math.Pow(f, 2);
```

```
#elif Q16
    q= " число Пи";
    f = Math.PI;

#elif Q17
    q= " извлечь корень из 4";
    f  = 4.0;
    f = Math.Sqrt(f);

#elif Q18
    q= " синус 0";
    f = Math.Sin(0.0);

#elif Q19
    q= " косинус 0";
    f = Math.Cos(0.0);
#elif Q20
    q= " тернарная да";
    f = true?2.0:0.0;
#elif Q21
    q= " тернарная нет";
    f = false?2.0:0.0;
#elif Q22
    q= " массива нет";
    int[] z=null;
    f = z==null?2.0:0.0;
#elif Q23
    q= " массив есть";
    int [] z = new int[1];
    f = z==null?2.0:0.0;

#elif Q24
    q= " прочитать 3";
    f = double.Parse("3");

#elif Q25
    q= " ошибка ";
    f = double.Parse(" 2 + 3");

#elif Q26
    q= " приоритет 7.0";
    f = 1.0 + 2.0 * 3;

#elif Q27
    q= " приоритет 9.0";
    f = (1.0 + 2.0) * 3;

#elif Q28
    q= " приоритет / ++ после 9";
    i = b++ + b++ + b++;

#elif Q29
    q= " приоритет / три ++ до 12";
    i = ++b + ++b + ++b;
```

```

#elif Q30
    int    []    a = {-1, 1, 2, 3, -4};    // разная инициализация
                a = new int[3] {11,-22,-33}; // не надо освобождать
    char    [,]    d = new    char    [3,2]    // массив из 6 char
                {{'c','d'},{'s','a'},{'q','w'}};
    double  [] [] c = new    double  [2] []; // массив из 2х массивов.
    c[0]    = new double []    {1.0, 2.0};
    c[1]    = new double [] {1.1, 2.2, 3.3};

    Console.WriteLine
        ("rank    of a/d/c: {0}/{1}/{2}", a.Rank, d.Rank, c.Rank);
    Console.WriteLine
        ("length of a/d/c: {0}/{1}/{2}", a.Length, d.Length, c.Length);

    //  a = Array.Empty<int>();
    Console.WriteLine
        ("length of a: {0}; ", a.Length);
    Array.Sort (a);
    Console.WriteLine
        ("first more then 0 and its index: {0}/{1}; "
         , Array.Find(a, moreThen0)
         , Array.IndexOf(a, Array.Find(a, moreThen0)));
    foreach (char e in d) {
        Console.Write("{0} ", e );
    }

    Environment.Exit(30);

#else

#error  You have to set at least one macro

#endif
    Console.WriteLine("{0} - i:{2}; f:{3:0.000}", q, s, i, f);
}
static bool moreThen0 (int x){
    return x > 0;
}

}

---- End Of File:./cs/app.cs

    Командный файл для построения одного из тестов

---- File:./cs/cs.cmd

```

```
rem компиляция ответов на отдельные вопросы
rem 1 N
rem где N - номер вопроса

csc /define:Q%1 /debug /out:q%1.exe *.cs

---- End Of File:./cs/cs.cmd
```

14.2 Базовые Вопросы

1. Особенности процесса построения приложения для .Net.

Приложения (как частный случай сборки) для Платформы .Net написанные на одном из таких языков программирования как C#, Basic, C++ компилируются в промежуточный язык программирования Microsoft Intermediate Language - IL. Таким образом посроенное приложение содержит внутри себя не машинный код, а код промежуточного языка программирования. При первом запуске приложения вызывается компилятор JustInTime-Compiler, который проверяет и компилирует код промежуточного языка программирования в машинный код (команды процессора). Второй и последующие запуски приложения не влекут использование JustInTime-Compiler.

Приложение, созданное на основе платформы .Net, может быть написано на нескольких языках программирования и, обычно, управляется языковой средой CLR, которая следит за освобождением оперативной памяти, такой код называется управляемым. Кроме того обозначения встроенных простых типов языка, например, int или char заменяются на классы из библиотеки классов .Net Framework (в наших примерах - System.Int32 и System.Char соответственно). Для выполнения приложения, помимо главной сборки требуются дополнительные библиотеки классов и сборки сторонних производителей, необходимые приложению; библиотеки классов .Net Framework - например, System.dll; среда CLR - это коллекция динамически подключаемых библиотек, которые выполняют загрузку сборок, сбором мусора, обработкой исключительных состояний и JIT - компиляцию.

2. Что такое сборка в .Net? Перечислите основные типы файлов MS Visual Studio для построения приложения.

Сборка это основной элемент из которых состоит приложение на основе платформы .Net. Обычно представляют собой исполняемый файл (имеет расширение exe) или файл библиотеки динамической компоновки (по другому, динамически подгружаемой библиотеки, имеет расширение dll).

Для посроения сборки в MS Visual Studio требуется создать решение (файл с расширением sln), которое содержит по крайней мере один проект (файл с расширением csproj), несколько файлов исходного кода приложения (файлы с расширением cs), несколько файлов с ресурсами (расширение resx), файл AssemblyInfo.cs в каталоге Properties, который содержит метаописание сборки. В частности, там содержится номер версии, описание и название сборки, автора и торговую марку.

После построения приложения, рядом с файлом сборки может находиться файл с расширением `pbd`. Он содержит информацию для отладки приложения. В частности, в случае исключительной ситуации из этого файла берутся номера строчек, в котором произошел сбой приложения.

3. Перечислите области видимости.

Есть глобальная область видимости, глобальная именованная область видимости (пространство имен), область видимости класса (структуры), локальная область видимости. Глобальная - это строчки снаружи всех фигурных скобок. Глобальная именованная - это строчки снаружи всех фигурных скобок и внутри скобок относящихся к ключевому слову `namespace`. Локальная - это строчки, находящиеся внутри фигурных скобок какого либо метода, индекса или свойства. Область видимости класса - это строчки, находящиеся внутри фигурных скобок, относящихся к определению класса.

4. Из чего состоит программа на языке C#? Охарактеризуйте язык C#.

Программа на языке C# состоит из классов, хотя бы один из которых содержит точку входа `Main`. Классы обычно вкладываются в область видимости при помощи ключевого слова `namespace`. Язык C# - простой, современный объектно - ориентированный и типобезопасный язык программирования, поддерживающий одиночное наследование. Язык обеспечивает надежность и устойчивость приложений при помощи сборки мусора, которая автоматически освобождает память занятую неиспользуемыми объектами; обработки исключительных состояний; статической (переменная связывается с типом в момент объявления и тип не может быть изменен позже) и строгой (которая исключает возникновение ошибок согласования типов во время выполнения) типизации, помимо всего прочего она не позволяет обращаться к неинициализированным переменным, выходить за пределы массивов или выполнять неконтролируемое приведение типов. Все типы языка встроены в единую систему типов и наследуются от корневого типа `object`.

5. Перечислите и охарактеризуйте встроенные типы языка C#. На какие группы делятся все типы C#?

Все типы C# делятся на три группы: значимые типы (основные встроенные типы и структуры), ссылочные типы (классы и массивы) и строчки. Эти группы типов по разному передаются в методы (ссылкой или значением). Все типы языка встроены в единую систему типов и наследуются от корневого типа `object`. Операция преобразования переменной любого типа в переменную типа `object` называется упаковкой, а обратная - распаковкой.

6. Перечислите и охарактеризуйте операторы языка C#.

Оператор это метод языка программирования, для которого в языке существуют специальный символ (обозначения оператора). Операторы используются при построении выражения - правильно записанной последовательности констант, имен и обозначений оператора. Операторы, которые применяются к одному выражению (операнду), называются унарными, операторы, которые применяются к двум выражениям (операндам)

называются бинарными, и есть один тернарный оператор. Операторы упорядочены уровнем приоритета, то есть очередностью выполнения. В выражении

```
n = 11 - 2 * 4;
```

сначала выполняется умножение, потом отнимание, потом присвоение. Кроме того, операторы обладают ассоциативностью. Ассоциативность задает очередность выполнения операторов с одинаковым приоритетом. В таком случае вычисления выполняются либо слева направо (левая ассоциативность), либо справа налево (правая ассоциативность).

Операторами с наиболее высоким приоритетом являются операторы доступа к члену, оператор вызова метода (method call or delegate invocation) и оператор индексирования.

Отдельно надо упомянуть про операторы тестирования типа `is` и `as`:

- (a) оператор `is` проверяет совместимость типов. Возвращает значение `true`, если вычисленный левый операнд может быть приведен к типу, заданному правым операндом.
- (b) оператор `as` выполняет преобразование типов. Возвращает левый операнд, приведенный к типу, заданному правым операндом, но `as` возвращает `null`, где (T)x вызывает исключение.

7. Перечислите и охарактеризуйте инструкции языка C#.

Инструкции языка C# делятся на следующие группы: инструкция объявления, инструкция выражения, инструкции выбора (ветвления), инструкции итерации (цикла), инструкции перехода, инструкции обработки исключительных состояний, а также инструкции `checked`, `unchecked`, `lock` и `using`.

Инструкции `checked` и `unchecked` используются для управления контекстом при выполнении целочисленных арифметических операций и преобразований проверки переполнения.

Инструкция `lock` позволяет создать взаимоисключающую блокировку заданного объекта перед выполнением определенных операторов, а затем снять блокировку.

Инструкция `using` используется для получения ресурса перед определенным оператором, и для гарантированного удаления ресурса после его завершения.

8. Приведите примеры инструкций объявлений для основных встроенных типов.

```
int a;  
double b = 3.14159;  
System.Single [] c = new float [] {1,2,3};
```

```

bool    []    e = {true, true, true};
short  [,]    f = new short [2,3] {{1,2,3},{4,5,6}};
string      g = new string(new char[] {'a', 'b', 'c'});
System.String i = "hi";

```

9. Перечислите и охарактеризуйте инструкции выбора (ветвления).

К инструкциям выбора относятся инструкции `if - else` и `switch - case - default`. Условие в инструкции `if` в отличие от C++ должно иметь логический тип. Ветка `else` может отсутствовать, если она присутствует, то считается относящейся к ближайшей сверху инструкции `if` без ветки `else`.

```

if ( true)
    if ( true)      /// комментариями
        a = 1;      /// отмечен
    else            /// одна
        a = 2;      /// инструкция

```

Инструкция `switch` выбирает для выполнения одну из секций - кандидатов, начинаются текстом `'case константа :'` и заканчиваются обязательной инструкцией безусловного перехода `break`;

```

switch ((new Random()).Next(1, 5)){
    case 1: Console.WriteLine("one"); break;
    case 2: Console.WriteLine("two"); break;
    case 3: Console.WriteLine("three"); break;
    default:
        Console.WriteLine("too much:"); break;
}

```

Выражение в скобках рядом со ключевым словом `switch` (как и константы возле ключевого слова `case`) должно быть не нулевым (not null, а не 0). В ранних версиях C# присутствовало требование к типу. позволялись целые (`char`, `short`, `int`, `long`), строки или перечисления.

10. Перечислите и охарактеризуйте инструкции итерации (цикла).

Инструкции итерации используются для многократного выполнения некоторой другой инструкции (embedded statement). К этой группе итераций относятся инструкции `while`, `do`, `for` и `foreach`. Переменные, объявленные в области видимости цикла, не видны в следующей инструкции

```

for (int jj = 1; jj < 3; jj++) {
    int bb = 1;
}
// bb и jj тут не существует.

```

Особенно интересен оператор `foreach`, который может применяться к любому типу, имеющему публичный метод `GetEnumerator`, возвращающий класс или структуру и тип, который он возвращает имеет публичное свойство `Current` и метод `MoveNext`, возвращающий логические значения. Обычно такими особенностями обладают коллекции - термин обобщающий массивы, списки, стеки, очереди и тому подобное.

```
static int Main(string []args){  
    foreach (string s in args)  
        Console.WriteLine(s);  
}
```

Причем в цикле нельзя менять объекты из свойства `Current`, в данном примере это строчка `s`;

С инструкциями цикла используются инструкции выхода из цикла - `break` и инструкция перехода к следующей итерации - `continue`.

11. Перечислите и охарактеризуйте инструкции безусловного перехода.

Инструкции перехода используются для передачи управления инструкции отличной от следующей за текущей. К этой группе относятся операторы `break`, `continue`, `goto`, `throw`, `return` и `yield`.

Инструкция `goto` позволяет переходить к другой инструкции помеченной соответствующей меткой в области видимости метода.

Инструкции `break` и `continue` используются внутри циклов для перехода к инструкции следующей за циклом (в случае `break`) и к следующей итерации (в случае `continue`). `break`, кроме того, используется в инструкции выбора `switch`.

Инструкция `throw` позволяет переход к содержащему её блоку инструкций следующему за ключевым словом `catch`. Подбор блока `catch` выполняется по типу объекта, который порождается в `throw`; Этот тип должен быть наследован от класса `Exception`. Передача управления, в отличие от `goto` может выполняться к любому методу, непосредственно или через другие методы, вызывавшему метод, в котором встретилась инструкция `throw`.

Инструкция `return` прекращает работу текущего метода и передает управление методу из которого вызывался текущий. При повторном вызове метода все локальные переменные инициализируются начальными значениями.

Инструкция `yield return` прекращает работу текущего метода и передает управление методу из которого вызывался текущий. При этом значения локальных переменных текущего метода сохраняются со времени прошлого вызова.

12. Перечислите и охарактеризуйте ключевые слова, используемые для обработки исключительных состояний.

Инструкция `try...catch` позволяют перехватывать исключения, создаваемые при выполнении блока кода, а инструкция `try...finally` используется для указания кода завершения, который выполняется всегда, независимо от появления исключений.

```
try {  
    if (true)  
        throw new IndexOutOfRangeException();  
}
```



```

        else
            throw new OverflowException();
    }
    catch (OverflowException ex){
        Console.WriteLine("this is else branch");
    }
    catch (IndexOutOfRangeException ex){
        Console.WriteLine("this is then branch");
    }
    finally{
        Console.WriteLine("end ");
    }
}

```

13. Что такое класс и структура. Различия между классами и структурами. Из чего состоит класс или структура?

Типы, которые создаются при помощи слова `class` или `struct`. Обычно состоят из констант, полей, методов, свойств и индексов, перегруженных операторов. Поля, методы, свойства и индексы могут называться членами класса. В сложных случаях могут содержать внутри себя объявления других классов, других структур и типов - делегатов. Структуры относятся к группе значимых типов, классы - к ссылочным. Кроме того, структуры могут наследовать только интерфейсы, у структур нельзя объявлять конструктор без параметров. При копировании объекта структуры создается копия исходной структуры, при копировании объекта класса создается копия указателя. Пример описания класса с одной константой и одним полем:

```

class a {
    public const double e = 2.718281828;
    public int b ;
    public int // не знаешь какую функцию писать
    eqv(int f) {return f;} // пиши функ. эквивалентности
}

```

14. Статические и не статические члены классов или структур.

Члены классов и структур могут иметь модификатором `static` и, в таком случае, будут статическими. В этом случае они находятся в области видимости класса (или структуры) но принадлежат скорее самому типу, а не объекту этого типа. Это означает, что в области видимости класса к ним можно обращаться из статических и нестатических методов, свойств и индексов. Но, наоборот, из статических методов, свойств и индексов нельзя обращаться к не статическим членам класса, без создания нового объекта. Далее, из внешней по отношению к классу области видимости (в случае наличия модификатора доступа `public`) к ним можно обращаться через имя типа и без создания объекта данного типа. К нестатическим членам из внешней области видимости можно обращаться только через созданный объект. Это можно трактовать так, что статические члены класса хранятся в типе, а нестатические члены класса - в объекте.

15. Как выполняется передача фактических параметров в методы? Ключевые слова `ref` и `out`.

Передача фактических параметров в вызываемые методы для значимых типов выполняется при помощи передачи значения (то есть, копии объекта) и изменение копии не влияет на, собственно, фактический параметр. Передача фактических параметров в вызываемые методы для ссылочных типов выполняется при помощи передачи указателя (ссылки) на исходный объект. Он меняется в случае изменений вызванном. Если планируется, что вызываемый метод должен менять объект значимого типа, то необходимо использовать ключевые слова `ref` или `out`. Ключевое слово `out` заставит компилятор убедиться, что объект менялся. В случае ссылочных типов ключевые слова `ref` или `out` позволят создать новый объект (захватывать новую память) и вернуть его в вызывающий метод (см. [2.3](#) и [10.4](#)).

16. Какой класс в .Net предоставляет математические функции. Примеры функций.

Тригонометрические (`Sin`, `Cos`), логарифмические (`Log`, `Exp`) и другие математические (`Sqrt`, `Cbrt` - кубический корень, `Pow`) функции в .Net собраны в статическом классе `System.Math`. Кроме того, с хорошей точностью в нем представлены две константы:

```
public const double E = 2.71828182845905;  
public const double PI = 3.14159265358979;
```

17. Какие структуры в C# используются для работы со временем?

Структура `DateTime` представляет моменты времени обычно выражаемыми как дата и момент дня с различных точек зрения. Например, день можно рассматривать как день года (`DayOfYear`), день месяца (`Day`) или день недели (`DayOfWeek`). Свойство `Now` может использоваться для оценки времени работы приложения. При этом разница между двумя моментами времени имеет тип структуры `TimeSpan` - интервал. В этой структуре естественно нет, например, дня недели. Интервал можно рассматривать как, например, часы, минуты и секунды либо как общее количество секунд (`TotalSeconds`) или минут (`TotalMinutes`).

18. Методы и свойства, которые предоставляет класс `string`?

Объект класса `string` является неизменяемым и содержит последовательность объектов класса `char`, в кодировке UTF-16. Строки инициализируются следующим образом:

```
string o = "c:\\Program Files\\Microsoft Visual Studio 8.0";  
string n = @"c:\Program Files\Microsoft Visual Studio 9.0";  
  
string t = @"My pensive SARA ! thy soft cheek reclined  
Thus on mine arm, most soothing sweet it is  
To sit beside our Cot,...";
```

В строке, начинающаяся с символа @, отсутствуют escape-последовательности.

Класс предоставляет большой набор методов (ToLower - преобразовать все символы в нижний регистр , ToUpper - в верхний регистр , Trim - отсечь пробельные символы слева и справа , TrimStart - отсечь пробельные символы слева , TrimEnd - отсечь пробельные символы справа , Substring - выделить подстроку , Replace - заменить символы на другие , Remove - удалить часть строки , Contains - содержит ли строка заданный параметр , LastIndexOf - вернуть индекс последнего вхождения символа (справа) , IndexOf - вернуть индекс первого вхождения символа (слева)).

Особенно надо выделить методы для разделения строки на подстроки (Split), соединения подстрок в одну строку (Join) и преобразование чего угодно в строку (Format). Неизменяемость объекта означает, что после любого изменения предыдущий объект забывается и вместо него появляется новый. При этом происходит захват новой памяти. Это означает что построение длинной строки в цикле из большого количества маленьких строчек при помощи последовательной конкатенации является очень неэффективным способом. Это нужно делать при помощи метода Join.

```
string s = "ONE,,TWO,,THREE,,";
char [] seps = new char[] {' ',' '};
string [] flds = s.Split(seps); // длина массива = 8
                               // в массиве будут строки длиной 0.
s = " ONE TWO \tTHREE\t ";
seps = new char[] {' ',' ','\t'};
flds = s.Split(seps
, StringSplitOptions.RemoveEmptyEntries);
// длина массива = 3
```

19. Методы и свойства, которые предоставляет структура char?

Структура char используется для хранения отдельных символов в кодировке UTF-16. Структура предоставляет большой набор методов, облегчающий разбор текстов. Например, методы вида IsXXXXX - отвечают на вопрос является ли символ цифрой (IsDigit), управляющим (IsControl), буквой (IsLetter), знаком пунктуации (IsPunctuation), пробелом (IsWhiteSpace), буквой нижнего регистра (IsLower) и так далее. Методы начинающиеся на To, предназначены для преобразования, Например, ToUpper - буква преобразуется в верхний регистр.

Как и для числовых типов существуют методы Parse и TryParse:

```
Console.WriteLine(char.Parse("A")); // Output: 'A'
Console.WriteLine(char.Parse("\uD800")); // Output: '??'
Console.WriteLine(char.Parse("\\")); // Output: 'A'
```

20. Какие escape-последовательности символов доступны в C# ?

```
\' - одинарная кавычка, нужна для символьных литералов;
\" - двойная кавычка, нужна в строковых литералах;
\\ - обратный слеш
\0 - завершающий ноль-символ
```

```

\b - бекспейс
\f - form feed (прогон страницы)
\n - новая строка
\r - возврат каретки
\t - горизонтальная табуляция
\v - вертикальная кавычка
\u - юникод последовательность
\U - юникод последовательность для суррогатных пар (UTF-16)
\x - юникод последовательность, похожа на
    \u, исключаящую переменную длину.

```

21. Какие методы, не приводящие и приводящие к исключительным состояниям, используются для преобразования строчек в числовые типы? И наоборот, числовых типов в строчки?

Для преобразования строчек в числовые типы используются статические методы соответствующего класса `Parse` и `TryParse`. `Parse` - приводит к исключительным состояниям. Например, к `FormatException` - это состояние возникает если строка не представляет константу ожидаемого типа, или `OverflowException` - это состояние возникает если величина константы ожидаемого типа из строки выходит за допустимые границы значений (см. поля `MinValue` и `MaxValue` в типе).

```

int a = int.Parse("0"); // тут все хорошо.
int b = int.Parse("null"); // тут - FormatException
short c = short.Parse("2000000000"); // тут - OverflowException

```

Методы `TryParse` - не приводят к исключительным состояниям, а возвращают признак удалось или не удалось выполнить преобразование.

```

double x= 0.0;
bool a = double.Parse("3.14159", out x);

```

Выполнение этого примера зависит от того какая культура используется в данном контексте по умолчанию (см. `CultureInfo Class`). Знак для отделения дробной части числа от целой может быть запятой или точкой.

Для преобразования числа в строчку в каждом типе используется метод `ToString`.

22. Какие типы и методы используются для ввода - вывода текстовых файлов?

Для ввода текстовых файлов используется класс `StreamReader` (как наследник абстрактного класса `TextReader`) с методами `Read` - для чтения следующего символа, `ReadLine` - для чтения следующей строчки, `ReadToEnd` - для чтения всех оставшихся символов в файле.

Для вывода в тестовые файлы используется класс `StreamWriter` (как наследник абстрактного класса `TextWriter`) и методы `Write` и `WriteLine`. Методы `WriteLine` завершают свой вывод символом новой строки, методы `Write` - нет. Могут выводит непосредственно объекты встроенных типов, либо с использованием строки формата, содержащей спецификаторы места вывода.

```

Console.Write(2.71828);
Console.WriteLine(
    "this is e: {0}, and this is pi:{1:0.000}, and e again: {0:0.0} "
    , 2.71828, 3.14159);

```

В текущем примере первый спецификатор места вывода для числа e (0) не задает количество знаков после запятой, для числа π (1:0.000)- задает три знака после запятой и третий, опять для числа e (0:0.0)- задает один знак после запятой.

Все рассмотренные классы принадлежат именованной области видимости (пространству имен) System.IO. Кроме того, они являются наследниками интерфейса IDisposable, и, значит, могут использоваться в инструкции using. Для метода Dispose существует синоним Close.

23. Какие типы и методы используются для ввода - вывода двоичных файлов?

В двоичных файлах, в отличие от текстовых, данные хранятся в таком же виде, как и в оперативной памяти, то есть, при вводе - выводе не изменяются. Для вывода используется перегруженные для всех простых типов методы Write класса BinaryWriter. Для перемещения по файлу используется метод Seek.

Для ввода данных существуют набор методов ReadXXXX (ReadBoolean - для bool, ReadInt16 - для short, ReadDouble - для double и так далее) класса BinaryReader.

Конструкторы обоих файлов в качестве формального параметра принимают класс FileStream. Объект такого класса можно создавать при помощи конструкторов класса или, как это делается чаще, при помощи статических методов Open или Create класса File:

```

int i;
using (BinaryReader br = new BinaryReader(
    File.Open("1.bin", FileMode.Open))){
    i = br.ReadInt32();
}

```

Все рассмотренные классы принадлежат именованной области видимости (пространству имен) System.IO. Кроме того, они являются наследниками интерфейса IDisposable, и, значит, могут использоваться в инструкции using. Для метода Dispose существует синоним Close.

24. Что такое метод класса? Примеры переопределение методов класса.

Метод класса (или структуры) это функция (то есть, четверка: тип возвращаемого значения, имя, возможно пустой список формальных параметров в круглых скобках и составная инструкция), которая принадлежит области видимости класса (или структуры). У методов - конструкторов нет типа возвращаемого значения, и после круглых скобок со списком формальных параметров и перед составной инструкцией может присутствовать обращение к конструктору родительского типа (ключевое слово base), или к другому конструктору своего типа (ключевое слово this)

25. Что такое конструктор класса? Как вызывается конструктор родительского класса. Примеры перегрузок конструкторов класса. Конструктор по умолчанию.

Конструктор класса - это метод класса, который не имеет типа возвращаемого значения и имя которого совпадает с именем класса. Конструктором по умолчанию называется конструктор, у которого пустой список формальных параметров. Конструктор может быть статический, он применяется для инициализации статических полей класса, вызывается неявно, до первого использования класса. Конструкторы вызываются

- при использовании оператора new;
- при использовании ключевого слова base с параметрами;
- при использовании ключевого слова this с параметрами;

26. Что такое свойство класса?

Свойством называют специальные методы доступа (get и set), при обращении к которым нет необходимости использовать оператор вызова метода (имя с круглыми скобками). Обращение к этим методам имеет вид выражения с оператором присвоения.

```
class a{
    int b;
    public int B    {
        get        {
            Console.WriteLine("get");
            return b;
        }
        set        {
            Console.WriteLine("set");
            b = value;
        }
    }
}

static void Main (){
    a x = new a();
    x.B = 1;    // тут вызывается метод set
    int c = x.B; // тут вызывается метод get
}
}
```

Ключевое слово value - является входным параметром метода set. Его тип совпадает с типом свойства. Внутри методов можно вставлять дополнительные инструкции. Разрешается дописывать модификаторы доступа, если они усиливают ограничения. Одно из свойств можно опускать. В общем, можно считать, что свойство - это методы, которые делают вид, что они - поле.

27. Что такое индексатор класса?

Индексатор класса - это специальные методы (подобно свойствам), которые вызываются при помощи оператора индексирования, а не оператора вызова метода.

```

class a{
    uint [] b = new uint [10];
    public int this[int i]      {
        get {
            Console.Write("get");
            if (i >=0 && i < b.Length)
                return (int) b[i];
            else
                return -1;
        }
        set {
            Console.Write("set");
            if (i >=0 && i < b.Length)
                b[i] = (uint)value;
        }
    }
    static void Main (){
        a x = new a();
        x[2] = 1;                // тут вызывается метод set
        Console.WriteLine(x[2]); // тут вызывается метод get
    }
}

```

Тут value - это входной параметр для метода set. Один из методов можно опускать, также можно усиливать модификатор доступа. Индексатор может иметь тип отличный от int и размерность больше чем 1 как в примере.

28. Что такое коллекция в .Net? Привести два примера коллекции, и их основные методы и свойства.

Коллекция в .Net это понятие для обобщения стеков, списков, очередей, хэш - таблиц (они же отображения, они же словари). Коллекции похожи на массивы, но отличаются от них тем, что коллекции предназначены для динамического изменения. Длина коллекции хранится в свойстве Count, один из часто используемых методов - Add - нужен для добавления нового элемента в коллекцию. Для использования шаблонов коллекций, нужно подключить именованную область видимости (пространстве имен) System.Collections.Generic. Там представлены словари (списки пар (ключ,значение), Dictionary<TKey,TValue>), списки (List<T>), очереди (Queue<T>), сортированные списки (SortedList<TKey,TValue>), и стеки (Stack<T>), которые предназначены для работы с объектами заданного типа T (TKey,TValue). С другой стороны в области видимости System.Collections существуют списки (ArrayList), хэш таблица (HashTable) - аналог словаря, (Queue) и стек (Stack), которые предназначены для работы с объектами типа object. На коллекциях, как и на массивах определена операция индексирования.

Ограничимся перечислением часто используемых методов из списка List<T>

- Clear - удаляет все элементы списка;
- Contains - проверяет, входит ли элемент в список;

- Exists - проверяет, есть ли элементы в списке, которые удовлетворяют некоторому условию;
- Find - аналог предыдущего метода, но возвращает первое вхождение, которое удовлетворяет условию;
- IndexOf - индекс элемента;
- Insert - вставить элемент в список, в указанное место;
- Remove - удалить первое вхождение указанного объекта из списка;
- RemoveAt - удалить элемент с указанным индексом;
- Sort - отсортировать список;
- ToArray - построить массив из коллекции.

См. пример:

---- File:./cs/list/p.cs

```
using System;
using System.IO;
using System.Collections.Generic;

class Program{
    static int Main() {
        List <char> ls = new List<char> ();
        char [] x = {'z','w','o'};
        ls.AddRange (x);
        ls.AddRange (new char[] {'h','o','p','a'});
        ls.Add('r');

        Console.WriteLine ("count/index of w/4th elem: {0}/{1}/{2}"
            ,ls.Count, ls.IndexOf('w'), ls[4]);
        ls.Remove('o');
        ls.Sort();
        Console.WriteLine ("count/index of w/4th elem: {0}/{1}/{2}"
            ,ls.Count, ls.IndexOf('w'), ls[4]);
        ls.Reverse();
        foreach (char e in ls)
            Console.Write("'{0}' ", e);
        // answer is
        //count/index of w/4th elem: 8/1/'o'
        //count/index of w/4th elem: 7/5/'r'
        //'z' 'w' 'r' 'p' 'o' 'h' 'a'
        return 0;
    }
}
```

---- End Of File:./cs/list/p.cs

29. Что такое инструкция using?

Существуют классы, требующие явного освобождения некоторых ресурсов, к ним относятся файлы, потоки команд (нити, thread), соединения

с базой данных, сокет и так далее. Они наследуются от специального абстрактного класса (объекты абстрактных классов нельзя объявлять), называемого интерфейсом - IDisposable и, поэтому, обязаны иметь метод Dispose, который и освобождает соответствующие ресурсы. Для гарантированного вызова этого метода и создана инструкция using, которая добавит вызов метода Dispose независимо от того, код внутри using выполнен успешно или возникло исключительное состояние. Пример использования инструкции using:

```
using ( BinaryWriter bw =
        new BinaryWriter(File.Open("1.bin", FileMode.Create)))
{
    bw.Write("just test");
    bw.Write(10);
    bw.Write(true);
}
```

30. Массивы в языке C#, инициализация массивов, одномерные и многомерные массивы. Основные свойства и методы массивов.

Массив - это последовательность переменных с одинаковым типом и одним именем. Доступ к отдельной переменной осуществляется при помощи операции индексирования (символ операции - прямоугольные скобки []). Массивы могут иметь больше чем одну размерность. Все массивы наследуются от класса System.Array, значит, тоже являются классами и относятся к ссылочному типу, наследуют свойства и методы этого класса. Например, свойство Length возвращает полное число элементов во всех размерностях массива, Rank - количество размерностей массива.

```
int    []  a = {-1, 1, 2, 3, -4};    // разная инициализация
        a = new int[3] {11,-22,-33}; // не надо освобождать
char   [,] d = new char [3,2]      // массив из 6 char
        {{'c','d'},{'s','a'},{'q','w'}};
double [][] c = new double [2] []; // массив из 2х массивов.
c[0]    = new double [] {1.0, 2.0};
c[1]    = new double [] {1.1, 2.2, 3.3};

Console.WriteLine
    ("rank of a/d/c: {0}/{1}/{2}", a.Rank, d.Rank, c.Rank);
Console.WriteLine
    ("length of a/d/c: {0}/{1}/{2}", a.Length, d.Length, c.Length);
Console.WriteLine
    ("length of a: {0}; ", a.Length);
//answer is
//rank of a/d/c: 1/2/1
//length of a/d/c: 3/6/2
//length of a: 3;
```

Кроме того имеется большое количество методов:

- Clear - метод в массиве выбранным элементам задает нулевое значение (соответствующее типу элемента);

- Copy - выполняет неглубокое копирование значений из одного массива в другой. Неглубокое - означает, что в случае массива ссылочных типов, копируются указатели. Это означает, что соответствующие друг другу элементы из одного и второго массива будут ссылаться на один объект;
- IndexOf - находит индекс заданного элемента;
- Find - находит элемента, удовлетворяющий условию;
- Find - находит элемента, удовлетворяющий условию;
- Resize - изменяет размер массива;
- Sort - несколько две группы перегруженных методов, для сортировки массива из одной размерности. Одна группа, если элементы массива сравнимые (то есть, тип элементов массива, был наследован от интерфейса IComparable, и на элементах определена операция меньше равно). Если элементы массива не сравнимые, то в метод нужно передавать специальный класс, наследованный от интерфейса IComparer (сравниватель) - вторая группа перегруженных методов.

Кроме того, массивы можно использовать в цикле foreach. Примеры вызова некоторых методов и цикла foreach:

```
Array.Sort (a);
Console.WriteLine
    ("first more then 0 and its index: {0}/{1}; "
     , Array.Find(a, moreThen0)
     , Array.IndexOf(a, Array.Find(a, moreThen0)));
foreach (char e in d) {
    Console.Write("{0} ", e );
}
//answer is
// first more then 0 and its index: 11/2;
// c d s a q w
```

При этом, moreThen0 это имя следующей функции:

```
static bool moreThen0 (int x){
    return x > 0;
}
```

14.3 Дополнительные Вопросы

1. Из каких разделов состоит сборка?

Сборка состоит из четырех разделов: манифеста или раздел метаописания, раздел кода приложения на промежуточном языке программирования, раздел типов сборки, которые могут использовать другие приложения, раздел ресурсов (указателей мыши, иконок и т.д.).

2. Основные директивы препроцессора C#. Для чего они используются?

Препроцессор C#. имеет следующие директивы:

- `#define`: что бы добавить (определить) новый макрос.
- `#undef`: что бы удалить (как бы разопределить == сделать неопределенным ранее определенный) ранее определенный макрос.
- `#pragma`: что бы управлять работой компилятора, например, отключать сообщения и предупреждения об ошибках.

```
#pragma warning disable 642
```

Отключить 642 предупреждение.

- `#if`, `#else`, `#elif`, `#end`: что бы пользоваться условной компиляцией. Пример использования условной компиляцией можно найти в разделе [14.1](#)).

3. Что такое абстрактный класс?

Класс называется абстрактным если в нем не реализован хотя бы один из методов (Но присутствует его сигнатура отмеченная ключевым словом `abstract`). Класс тоже отмечается ключевым словом `abstract` Нет возможности создавать объекты этого класса.

4. Приведите и охарактеризуйте примеры нескольких типов, которые используются для обработки исключительных состояний.

Для обработки исключительных состояний часто используются следующие типы:

- `Exception` - родительский тип для наследования всеми остальными специализированными типами. Содержит самую общую информацию для описания ошибки. Следующие поля её объясняют: `Message` - текст, описывающий исключительное состояние, `TargetSite` - метод, который вызвал исключительное состояние, `StackTrace` - стек вызовов методов, начиная с `Main`. Если приложение скомпилировать с ключем `/debug`

```
csc *.cs /debug
```

То стек вызовов будет содержать имя файла и строчку, которая вызвала ошибку. Что то вроде:

```
в GCExample.aa.f() в .\foo\1.cs:строка 15
в GCExample.Program.f() в .\foo\1.cs:строка 27
в GCExample.Program.Main() в .\foo\1.cs:строка 34
```

Иногда бывает полезно свойство `HRESULT` - код ошибки.

- `FormatException` - исключение обычно возникает при конвертации данных из текстового представления во двоичное. Например, при попытке преобразовать строку "1a2" в целое (при помощи метода `Parse`).
- `IndexOutOfRangeException` - это исключение влечет попытка обратиться к несуществующему элементу массива.
- `NullReferenceException` - это исключение влечет попытка использовать переменную, содержащую ссылку на объект `null`.

5. Реализуйте метод CompareTo в следующем классе

```
class S : IComparable {
    bool    first;
    string fio;
    uint    age;
}
```

для сортировки коллекций или по полю fio, или по полю age.

```
class S : IComparable {
    bool    first;
    string fio;
    uint    age;
    public CompareTo
    public int CompareTo (object other)
    {
        if (first)
            return fio.CompareTo(((S)other).fio);
        else
            return age.CompareTo(((S)other).age);
    }
}
```

Недостаток метода - возможность получения исключительной ситуации.

6. Добавьте в класс

```
class S {
    string fio;
    uint    age;
}
```

конструктор и статическое свойство, для подсчета количества созданных объектов.

```
class S {
    static private uint count = 0;
    string fio;
    uint    age;
    public S (string f="", uint a = 0) {
        fio = f;
        age = a;
        count ++;
    }
}
```

7. Что такое перегрузка операторов? Охарактеризуйте перегрузку операторов в C#.

Перегрузка операторов (функций) означает определение нескольких функций с одинаковыми именами. Операторы отличаются от функций, в основном, только тем, что применяются в специальном виде. Например, оператор сложения применяется не в виде:

```
int a = +(1, 2);
```

а в виде

```
int a = 1+2;
```

Кроме того, в C# требуется, что такие методы должны быть статическими и открытыми, возвращать объект своего класса и не иметь параметров с модификаторами `out` или `ref`.

Пример (чисто иллюстративный, придуманный по мотивам сложения комплексных чисел, но достаточно бессмысленный для координат МГС84) перегрузки бинарного оператора сложения для координат:

```
class coor {
    double ltt;
    double lng;
    public coor ( double nord, double east)
        { ltt = nord; lng = east;}
    public static coor operator+ (coor a, coor b){
        return new coor (a.ltt + b.ltt, a.lng + b.lng);
    }
}
```

8. Для класса

```
class coor {
    double ltt;
    double lng;
}
```

Перегрузите оператор преобразования в строку.

Для общедоступных приложений принято использовать шесть знаков после запятой. Выбрано неявное преобразование, так как оператор не предусматривает возникновение исключительной ситуации:

```
public static
implicit operator string (coor r) { // записи в строчку
    string val = "wrong coors";
    bool letters = true;
    if ( 0.0 <= r.ltt && r.ltt <= 90.0
        && 0.0 <= r.lng && r.lng <= 180.0
    ) {
        if (letters)
            val = String.Format("N{0:000000} E{1:0000000}"
                                , r.ltt, r.lng);
        else
            val = String.Format("{0:0000000} {1:0000000}"
                                , r.ltt, r.lng);
    }
    return val;
}
```

9. Какой механизм используется для реализации полиморфизма?

Программа называется мономорфной, если каждое выражение в ней имеет один тип. Программа называется полиморфной, если в ней существуют выражения имеющие несколько типов. Тогда полиморфизм это свойство языка позволяющее записывать выражения нескольких типов. Согласно Стрейчи (который впервые использовал это слово в 1967 году) и, позднее, Люку Карделии - перегрузка функций и неявное приведение типов относятся к такому виду полиморфизма. к специальному виду полиморфизма (ad-hoc полиморфизм). При этом код программы годится только для тех типов, которые известны на момент написания кода.

К истинному полиморфизму относится полиморфизм включения (в смысле включения типов), при котором код программы годится для всех типов сразу (и тех, которые будут получены потом при помощи наследования) или параметрический полиморфизм, который используется шаблонами.

Пример создания полиморфных (в смысле включения) функций `X.txt2bin` и `X.str2rec` смотри в разделе 15.2. При полиморфизме включения часто применяется механизм позднего связывания (описатели `virtual` и `override`).

Таким образом, для реализации полиморфизма используется неявное приведение типов, наследование, перегрузка функций и операций, виртуальные функции и позднее связывание, шаблоны.

10. Позднее и раннее связывание. Виртуальные функции и переопределение функций. Ключевые слова `virtual` и `override`.

Наследование, позволяющее построение классы наследников на основе классов родителей, приводит к тому, объекты класса наследника имеют тип класса родителя и класса наследника. Это означает возможность присваивать переменной родительского класса объект из любого класса наследника без преобразования.

```
class P { void f(){};
class C:P { new void f(){};
Main(){
    P p = new C();
    p.f();
}
```

При этом строчка `p.f()`; использует родительскую версию метода `f`, хотя сам объект `p` на деле имеет тип наследника. Такой способ подбора метода для объекта называется ранним связыванием. В случае

```
class P { virtual void f(){};
class C:P { override void f(){};
Main(){
    P p = new C();
    p.f();
}
```

строка `p.f()`; использует версию метода `f` из класса наследника. Такой способ подбора метода для объекта называется поздним связыванием.

---- File:./cs/override/program.cs

```
//define TEST

using System;
using System.Data;
using System.Threading;
using System.Net;
using System.IO;

class app {
    class coor {
        double ltt;
        double lng;
        virtual
        public void write(){
            Console.Write("{0:0.00000} г.с.ш. {1:0.00000} г.в.д.", ltt, lng);
        }
    }

    class coor3: coor {
        double alt;
        override
        public void write(){
            base.write();
            Console.Write(" {0:0.00000} м. высоты над уровнем моря", alt);
        }
    }

    static int Main(string[] ars)
    {
        coor c = new coor();
        Console.Write("родитель: ");
        c.write();
        Console.WriteLine("");
        c = new coor3();
        Console.Write("наследник: ");
        c.write();
        Console.WriteLine("");
        return 0;
    }
}
```

---- End Of File:./cs/override/program.cs

В результате работы приложение выдаст следующие строки

```

родитель: 0,00000 г.с.ш. 0,00000 г.в.д.
наследник: 0,00000 г.с.ш. 0,00000 г.в.д.
              0,00000 м. высоты над уровнем моря
    
```

Хотя и первая, и вторая строка выводились одним методом переменной родительского типа

```
c.write();
```

11. Что такое атрибуты (System.Attribute)? Примеры часто используемых атрибутов.

Атрибуты - это специальный инструментарий, позволяющий добавлять некоторое метаописание к классам, интерфейсам или методам и полям класса. Кроме того, атрибуты позволяют управлять работой компилятора. Два важных атрибута: [STAThread] - ставится перед точкой входа в приложение Main и приводит к тому, что компилятор создает одно-нитевое приложение (что позволяет пользоваться классами из области видимости System.Windows.Forms); [Serializable] - ставится перед классом, объекты которого можно сериализовать.

12. Как из типа перечисление (Enum) получить текстовые представления всех его значений и строчку преобразовать в значение из типа перечисление?

В качестве примера такой конвертации можно рассмотреть пример из проекта журналирования [25]. Получение массива из всех элементов заданного перечисления __перечисление__ выполняется методом Enum.GetValues(typeof(__перечисление)). Обратная конвертация выполняется методом Enum.Parse. Смотри пример ниже.

```

public enum IMPORTANCELEVEL {
    Spam          ///< мусорные сообщения
    , Debug        ///< отладка
    , Warning      ///< предупреждения
    , Stats        ///< менее важная информация
    , Error        ///< ошибки приложения
    , FatalError   ///<катастрофические ошибки, делающие
                  ///< невозможным функционирование приложения
    , Info         ///<очень важная информация
    , Ignore       ///< для отсутствия вывода вообще
};

// метод выдает список уровней важности для подсказки оператору
public static string ILList() {
    Array all = Enum.GetValues(typeof(IMPORTANCELEVEL));
    List<string> a = new List<string>();
    foreach (object i in all)
        a.Add(i.ToString());
    return String.Join(" ", a);
}
    
```



```

public static IMPORTANCELEVEL strtolvl(string code) {
    IMPORTANCELEVEL x = IMPORTANCELEVEL.Error;
    try {
        x = (IMPORTANCELEVEL) Enum.Parse(typeof(IMPORTANCELEVEL), code);
    }
    catch {
        x = IMPORTANCELEVEL.Error;
    }
    return x;
}

```

13. Какой класс в .Net используется для создания окон? Какие методы в .Net используются для демонстрации окон оператору? Типы окон.

Для создания окон в .Net используется класс Form из области видимости System.Windows.Forms. Для демонстрации окна используется три различных метода: Form.ShowDialog - для демонстрации модальных окон (блокирующих доступ к другим окнам приложения); Form.Show - для демонстрации дочерних окон (не блокирующих доступ к другим окнам приложения); Application.Run - для демонстрации главного окна приложения.

14. Какой класс в .Net используется для создания управляющих элементов окон? Каким способом управляющие элементы связываются с окном?

Все управляющие элементы окна наследуются из класса Control.

```
Controls.Add(new Button());
```

15. Что такое файловая система? Какие классы используются для управлением файловой системы (создание, копирование, удаление файлов и директорий)?

файловая система - это набор требований и соответствующая им часть операционной системы, которая отвечает за создание, хранение, чтение, редактирование, уничтожение данных на носителях информации и за управление доступом к этим данным. В операционной системе Windows данные на носителях информации представляют собой древовидную структуру из каталогов и файлов. File - класс, который предоставляет программисту возможности для управления файлами и их атрибутами. Directory и DirectoryInfo - классы, который предоставляет программисту возможности для управления каталогами и их атрибутами. Причем методы из Directory - статические и применяются без создания объекта, а методы из DirectoryInfo - не статические и применяются после создания объекта и лучше применять их в случае нескольких действий с файловой системой.

16. Ключевое слово yield и пример его использования.

Инструкция создает итератор, позволяющий использовать инструкцию цикла foreach. Применение yield return декларирует, что данный метод возвращает последовательность IEnumerable, элементами которой являются результаты выражений каждого из yield return. Причем с возвращением значения, yield return передает управление вызывающей стороне

и продолжает исполнение метода после запроса следующего элемента. Значения переменных внутри метода с `yield` сохраняются между запросами.

```
static void Main() {
    // Display powers of 2 up to the exponent of 8:
    foreach (int i in Power(2, 8))
        Console.Write("{0} ", i);
}

public static
System.Collections.Generic.IEnumerable<int>
Power(int number, int exponent) {
    int result = 1;
    for (int i = 0; i < exponent; i++){
        result = result * number;
        yield return result;
    }
}
```

(см. <https://habr.com/ru/post/311094/>).

17. Для чего используется класс File.

Класс File используется для выполнения различных действий с файлами, таких как копирование, перемещение, переименование, создание, открытие, удаление и добавление к одному файлу за раз. Можно также использовать класс File для получения и задания атрибутов файла или сведений DateTime, связанных с созданием, доступа и записи в файл. Если вы хотите выполнять операции с несколькими файлами, необходимо использовать Directory.GetFiles или DirectoryInfo.GetFiles.

Некоторые методы из класса File, возвращают объект для ввода - вывода данных. например, Open или Create - возвращает типа FileStream, который можно использовать для дальнейшей работы с файлом, в частности, для создания объектов BinaryReader и BinaryWriter. Эти объекты используются для ввода- вывода двоичных данных. А метод CreateText - возвращает объект типа StreamWriter, а OpenText - объект типа StreamReader. Они используются для ввода вывода текстовых данных.

18. Получение случайных чисел в .Net.

Для генерации случайных чисел в .Net используется класс Random из пространства имен System. конструктор по умолчанию (без параметров) создает объект с использованием системного времени. Само случайное число генерируется при помощи метода Next.

```
Random r = new Random();
int i = r.Next();
i      = r.Next(0, 10); // сгенерировать число от 0 до 10
r      = new Random(245);
```

Для создания последовательности псевдослучайных чисел можно использовать не системное время как в примере (в этом случае последовательности получаются разными), а заданное число. Тогда последовательность окажется одинаковой и программу будет легче отлаживать.

19. Что такое интерфейс?

Интерфейс - это конструкция языка, позволяющая задавать набор методов, которые обязаны иметь наследники интерфейса. Задается ключевым словом `interface`. В фигурных скобках, относящихся к определению интерфейса перечисляются сигнатуры методов, естественно, без тела. Пример интерфейса, который требует методы, обеспечивающие навигацию по карте:

```
public interface iMapZoom {
    // гарантировать попадание заданной точки в область видимости
    void mustShow ( double b, double l);
    // сделать масштаб по оси X и по оси Y одинаковыми
    void mkZmEqual();
    // сдвиг области просмотра по линии юг север
    void moveNordSouth (
        int shift // проценты сдвига области видимости,
                  // >0 -- на север, <0 -- на юг
    );

    // сдвиг области просмотра по линии запад - восток
    void moveEastWest (
        int shift // проценты сдвига области видимости,
                  // >0 -- на восток, <0 -- на запад
    );
    // изменение размеров области видимости
    void zoom (
        int shift // >0 --увеличение области видимости (мелкий масштаб),
                  // <0 уменьшение области видимости (укрупнение масштаба)
    );
}
```

20. Для чего используется класс `Directory`?

Класс `Directory` предоставляет статические методы для действий над каталогами: `CreateDirectory` - создать каталог; `Delete` - удалить каталог; `Move` - переместить каталог; `Exists` - проверить существование каталога; `EnumerateDirectories` - извлечь список подкаталогов; `EnumerateFiles (GetFiles)` - извлечь список файлов.

Пример который извлекает имена файлов и имена подкаталогов из корневого каталога диска `c:`

```
string [] fileEntries = Directory.GetFiles("c:\\");
foreach(string fileName in fileEntries)
    Console.WriteLine("next file: '{0}'",fileName);
string [] subdirectoryEntries
    = Directory.GetDirectories("c:\\");
foreach(string subdirectory in subdirectoryEntries)
    Console.WriteLine(
        "next directoty: '{0}'", subdirectory);
```

21. Для чего используется класс `DirectoryInfo`?

Класс `DirectoryInfo` предоставляет нестатические методы для действий над каталогами: `Create`, `CreateSubdirectory` - создать каталог; `Delete` - удалить каталог; `MoveTo` - переместить каталог в указанное место; `EnumerateDirectories` - извлечь список подкаталогов; `EnumerateFiles (GetFiles)` - извлечь список файлов. Существование каталога проверяется свойством (а не методом) `Exists`;

Пример который извлекает имена файлов и имена подкаталогов из корневого каталога диска c:

```
DirectoryInfo di = new DirectoryInfo("C:\\");
string [] fileEntries = di.GetFiles();
foreach(string fileName in fileEntries)
    Console.WriteLine("next file: '{0}'", fileName);
string [] subdirectoryEntries
    = di.GetDirectories();
foreach(string subdirectory in subdirectoryEntries)
    Console.WriteLine(
        "next directoty: '{0}'", subdirectory);
```

22. Что такое автоматически реализуемые свойства (auto-implemented properties)? Приведите пример.

Автоматически реализуемые свойства - это особенность компилятора, которая позволяет существенно сократить громоздкость при описании свойств класса (или структуры), в случае, когда от методов доступа (accessors) не требуется дополнительных действий.

Например, автоматически генерируемое свойство следующего класса

```
class a{
    public int B { get; set; }
}
```

заменяется компилятором на следующий код: `public int B [CompilerGenerated] get return this.kBackingField;[CompilerGenerated]setthis. < B > kBackingField=value; < B > kBackingField;`

К одному из методов доступа можно дописать более жесткий модификатор доступа чем заданный у поля. В данном случае можно добавить `private` или `protected`. Например, так:

```
class a{
    public int B { get; protected set; }
}
```

23. Что такое класс `StringBuilder`?

Класс `StringBuilder` - это аналог класса `String` с изменяемой последовательностью символов. Определен в пространстве имен `System.Text`. То есть, он хранит последовательности символов в кодировке UTF-16, но эту последовательность, в отличие от класса `String`, можно изменять.

Имеет такое же свойство `Length` - текущая длина, имеет новое свойство `Capacity` - максимальная длина строки, хранить в этом месте без перезахвата памяти. Индексатор в классе `String` позволяет только вынимать символ из строки. То есть, для него задан только метод доступа `get`. Индексатор в классе `StringBuilder` позволяет вынимать и вставлять символ в строку. То есть, для него заданы методы доступа `get` и `set`.

```
StringBuilder a = new StringBuilder("absd");
a[2]='2';          // если бы переменная a имела тип String
```

В классе `StringBuilder` отсутствуют методы `IndexOf` или `StartsWith` для поиска в последовательности символов. Вместо метода `Format` - для форматированного вывода в строку, класс `StringBuilder` имеет метод `AppendFormat` - добавить в конец строки форматированный вывод.

24. Что такое лямбда - выражение? Примеры использования.

Лямбда - выражение - выражение, которое при помощи символа операции `=>` создает безымянную функцию. Записывается в виде лямбда - выражения или в виде лямбда - оператора.

```
(input-paramaters) => expression

(input-paramaters) => { statements-list }
```

У входных параметром можно задавать типы, а можно не задавать. Если входной параметр один, то скобки можно пропустить. Созданную безымянную функцию (она называется замыканием) можно присвоить переменной - делегату или передавать в качестве фактического параметра в другой метод. Рассмотрим следующий пример:

```
---- File:./cs/lambda/p.cs
```

```
using System;
using System.Collections.Generic;

class p{
    static void Main()    {
        int n, j = 0;
        n = Int32.Parse(Console.ReadLine());
        List<int> a = new List<int>(); //создаем список типа int
        DateTime date1 = DateTime.Now; //фиксируем начальное время программы
        for (int i = 2; i < n; i++)    {
            if (a.FindAll(x => i % x == 0).Count == 0) //если i не делится на любое число x из
                a.Add(i); //добавляем в список значение i
        }
        a.ForEach(x => Console.WriteLine(x)); //выводим все значения из списка
        Console.Error.WriteLine((DateTime.Now - date1).TotalSeconds);
    }
}
```

```
---- End Of File:./cs/lambda/p.cs
```

В строчке

```
List<int> a = new List<int>();
```

создается список целых чисел *a*. Метод `FindAll` из списка извлекает все целые числа, которые являются делителем текущего проверяемого числа *i* лямбда - выражение

```
x => i % x == 0
```

задает функцию, которая в качестве формального параметра принимает *x* (текущее целое число из списка *a*) и пытается на него разделить *i*, если список делителей пустой, то число простое и его добавляем в список *a*. Функция - замыкание имеет доступ к переменной *i* из окружающих её строчек. Такая переменная в MSDN называется захваченной переменной.

25. Какие типы и методы используются для низкоуровневого ввода - вывода файлов?

Для работы с файлами на низком уровне используется класс `FileStream`.

Этот класс предоставляет поток данных для файла операционной системы, поддерживающий синхронные и асинхронные чтение и запись. В классе есть несколько свойств для `CanRead` - можно ли читать из него, `CanWrite` - можно ли писать в него, `CanSeek` - можно ли менять позицию в потоке данных, `Name` - абсолютный путь к файлу, `Length` - длина файла, `Position` - текущая позиция в потоке данных. И наиболее важные методы: `Read` - прочитать блок байт с текущей позиции, `Write` - записать блок байт с текущей позиции, `Seek` - задать текущую позицию в потоке данных. Класс наследует интерфейс `IDisposable`.

26. Что такое регулярное выражение?

Регулярное выражение это гибкий, удобный и эффективный способ обработки текстов. Регулярные выражения используются при помощи класса `Regex` из области видимости `System.Text.RegularExpressions`. В самом простом случае требуется иметь текст (строчку) в которой ищется требуемая подстрока и шаблон, который задает искомую подстроку. Шаблоны задаются специальным языком. В этом языке многие символы (кроме символов `'[', ']', '\', \'`).

Зададим следующие регулярные выражения:

```
Regex r1=new Regex("abc");    //шаблону подходит текст abc
Regex r2=new Regex("^abc");    // подходит текст abc в начале строки
Regex r3=new Regex("[0-9]");   // подходит любая цифра
Regex r4=new Regex(@"\d");      // подходит любая цифра
Regex r5=new Regex(@"\d{2}$"); // две цифры в конце строки
```

Тогда следующие выражения будут равны true:

```
r1.IsMatch("zzzz abc a")
r2.IsMatch("abc a")
r3.IsMatch("abc 5 a")
r4.IsMatch("ab 9 a")
r5.IsMatch("ab 9 a35")
```

Метод Match - вернет информацию о первом слева вхождении подходящей строки:

```
Match m = r5.Match("ab 9 a35");
```

При этом m.Value примет значение '35', а m.Value - 6.

27. Привести пример и объяснить использования ключевых слов checked, unchecked.

Арифметические операции и операцию преобразования типа C# выполняет в контексте, который может быть контекстом checked (проверять наличие переполнения) или контекстом unchecked (не проверять наличие переполнения). В первом случае переполнение вызывает исключительную ситуацию и выполнение приложения прекращается. Во втором случае, просто отбрасываются старшие биты, в зависимости от типа. Эти ключевые слова можно использовать в двух видах: как блок инструкций либо похоже на применение функции.

```
---- File:./cs/checked/p.cs
```

```
using System;

class p{
    static void Main()    {
        byte b = 255;
        byte c = unchecked(b++); // в контексте unchecked
        b=255;                  // выполняется только операция инкремента
        unchecked { // блок инструкций
            b=255; // в контексте unchecked выполняются
            b++;  // все операторы
            b=255; //
            b++;  //
        }
        b=255; //
        b++;   // exception is here
    }
}
```

```
---- End Of File:./cs/checked/p.cs
```

Контекстом можно управлять так же при помощи ключа компилятора

/checked[+|-] Сгенерировать проверки переполнений

Например, компиляция приведенного выше примера, выполненная следующим образом:

```
csc /debug /checked+ /out:a.exe /unsafe *.cs /nologo
```

приведет к исключительной ситуации на последней инструкции в 16 строке.

```
Необработанное исключение:
System.OverflowException:
Переполнение в результате выполнения арифметической операции.
в p.Main() в c:\cs\checked\r.cs:строка 16
```

28. Что такое динамический тип (dynamic)? Приведите отличия от типа object и типа var.

dynamic - это обозначение для статического типа, для объектов которого не выполняется проверка типа. В большинстве случаев проверка рассматривает объекты такого типа, как имеющие тип object. Но во время компиляции считается, что к таким объектам можно применять любые операторы и любые методы. Если операторы или методы применимы к текущему объекту, то они нормально завершатся на шаге выполнения. Если нет - то попытка их применения приводит к исключительному состоянию. Переменной типа dynamic можно присваивать переменные разных типов в течении выполнения приложения. В то время, как переменной типа var можно присваивать переменные только одного типа. Тогда её тип определяется начальным присвоением, что называется неявной типизацией. Переменной типа object можно присваивать переменную любого типа (упаковка), так как класс object является родителем любого типа, но без явного приведения к исходному типу (распаковка) нельзя использовать ни операторы, ни методы типа - наследника.

29. Примеры интерфейсов в C#.

Наверно, одни из самых широко используемых интерфейсов в C# являются интерфейсы IEnumerable (и IEnumerator), IDisposable и IComparable.

Для перебирания объектов из коллекций (эти объекты описаны классами - наследниками интерфейса IEnumerable (и IEnumerator)) существует специальная инструкция foreach. Например:

```
int [] a = new int [4] {1,2,3,44};
foreach (int z in a)     // в этом операторе нельзя менять
    Console.WriteLine(z); // переменную z
```

Объекты, созданные при помощи класса - наследника интерфейса IDisposable, называются высвобождаемыми. Настоятельно рекомендуется классы, использующие файлы, соединения с БД, сокеты, именованные каналы (named pipe), потоки команд (thread) наследовать от интерфейса IDisposable и освобождать перечисленные сущности (ресурсы) в методе Dispose. Объекты таких классов желательно использовать в операторе using:


```

using (BinaryWriter bw
    = new BinaryWriter(File.Open("1.bin", FileMode.Create))){
    bw.Write(1);
    bw.Write(3.141592);
}

```

При этом, на закрывающей операторной скобке, гарантированно вызывается метод `Dispose`, независимо от того, нормально выполнились ли все инструкции внутри оператора `using` или произошла исключительная ситуация.

Для объектов из классов - наследников интерфейса `Comparable` требуется создания методов сравнения

```

class Xxx: Comparable
{
    int CompareTo (object o){
        Xxx x = 0 as Xxx; // распаковка тут
        if (x != null) {
            int rc = 0;
            // выполнить сравнение this и x
            return rc;
        }
        else
            throw new Exception("I can't compare this object");
    }
}

```

Если код возврата `rc < 0`, то текущий объект (`this`) меньше чем объект `o`. Если `rc == 0`, то объекты совпадают. Если `rc > 0`, то объект `o` меньше объекта `this`. Коллекции из таких объектов можно сортировать методами `Sort` из класса коллекции. Далее, для этого интерфейса требуется выполнение распаковки и не гарантируется, что объекты коллекции будут удачно преобразованы к нужному типу. Что бы не сталкиваться с такими трудностями, можно использовать шаблон - интерфейс `Comparable<T>`. тогда приведенный выше пример будет выглядеть следующим образом:

```

class Xxx: Comparable<Xxx>
{
    int CompareTo (Xxx o){
        int rc = 0;
        // выполнить сравнение this и o
        return rc;
    }
}

```

В разделе [15.2](#) можно посмотреть полный пример для сортировки.

30. Что такое класс `CultureInfo` и как она влияет на работу приложения?

Класс `CultureInfo` - возвращает или задает язык и региональные параметры для приложения. К таким параметрам, например, относится знак отделяющий числитель десятичной дроби от целой части - этот знак может быть запятой или точкой задается свойством `NumberFormatInfo.NumberFormat.NumberDecimalSeparator`

Этот параметр влияет на работу методов `double.Parse`, `double.TryParse`, `double.ToString`. Так же представляет интерес формат для записывание дат и времени - `DateTimeFormat`; и символ денежной единицы - `NumberFormatInfo.NumberFormat.CurrencySymbol`.
Пример для получения десятичного разделителя:

```
string NumberDecimalSeparator
= Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator;
```

14.4 Создание Оконных Приложений

1. Какие свойства класса `Form` используются для создания многодокументного интерфейса?

У главного окна приложения свойство `IsMdiContainer` установить в `true`:

```
IsMdiContainer = true;
```

Свойству `MdiParent` дочернего окна указать родителя мультидокументного приложения (главное окно приложения).

Коллекцию `MdiChildren` можно использовать для проверки было ли уже создано дочернее окно. Это позволяет избежать повторного открытия окна.

Одному из элементов меню (обычно предпоследний справа) Установить свойство `MdiList` в `true`.

2. Напишите метод для выбора и активизации уже открытого дочернего окна.

```
bool shouldIOpen (string text){
    int i = 0;
    for (int i = 0; i < MdiChildren.Count(); i++)
    {
        if (this.MdiChildren[i].Text == text)
        {
            MdiChildren[i].Activate();
            return false;
        }
    }
    return true;
}
```

3. Какой класс используется для создания окон?

Для создания окон используется классы `Form` и `Control` из области видимости `System.Windows.Forms`. При помощи `Form` создаются объекты для управления окнами, при помощи наследников класса `Control` - управляющие элементы внутри окон.

4. Какие методы используются для демонстрации окон?

Для демонстрации окна используется три различных метода: `Form.ShowDialog` - для демонстрации модальных окон (блокирующих доступ к другим окнам приложения); `Form.Show` - для демонстрации дочерних окон (не блокирующих доступ к другим окнам приложения); `Application.Run` - для демонстрации главного окна приложения. `Activate` - для активизации уже открытого дочернего окна.

5. Какие свойства связаны с кнопками управления окном?

При помощи следующих свойств окна можно прятать или показывать соответствующие кнопки управления: `MinimizeBox`, `MaximizeBox`, `ControlBox`.

6. Какие свойства и перечисления используются при создании диалоговых окон?

Свойства `MinimizeBox`, `MaximizeBox`, `ControlBox` - установить в `false`. Свойствам `AcceptButton` и `CancelButton` задать правильные кнопки окна.

Кнопкам окна задать соответствующие значения - результаты диалога.

```
butOk.DialogResult = DialogResult.OK;  
butCancel.DialogResult = DialogResult.Cancel;
```

Стиль границы окна задать в `FormBorderStyle.FixedDialog`.

```
FormBorderStyle = FormBorderStyle.FixedDialog;
```

7. Перечислите и охарактеризуйте стандартные диалоговые окна.

Диалоговое окно `MessageBox` - используется для демонстрации сообщений оператору и последующего выбора оператором одной из нескольких альтернатив. `OpenFileDialog` и `SaveFileDialog` используются для стандартного открытия новых файлов или сохранения файлов. Важными свойствами являются свойства `RestoreDirectory` и `InitialDirectory`

```
openFileDialog1.RestoreDirectory = true;  
openFileDialog1.InitialDirectory  
= Path.GetDirectoryName(  
    Assembly.GetExecutingAssembly().Location);
```

8. Как `event delegate` используется при разработке оконного интерфейса?

Для каждого события, которое может произойти с объектом класса `Form` разработчики .NET разработали набор типов, которые используются парами. Первым типом описали соответствующее поле класса `Form`. Вторым тип используется для передачи параметров в обработчик события. Например, для события, которое возникает перед закрытием окна созданы:

- тип `FormClosingEventHandler`, которым описано поле `FormClosing` - используемое для вызова обработчиков события (методов для обработки события).

- тип `FormClosingEventArgs`, который обеспечивает передачу параметров в обработчик.

Для обработки события прикладной программист должен написать метод правильного типа (в текущем примере `FormClosingEventHandler`) и добавить его к полю `FormClosing`. Пример обработчика приводится в ответе на следующей вопрос.

9. Какое событие нужно использовать перед закрытием окна? Приведите пример обработчика.

Событие `FormClosing` используется перед закрытием окна, позволяет отменить закрытие.

```
void formClosing(Object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        e.Cancel = true;
        if (MessageBox.Show(
            "Вы уверены что хотите закрыть программу?",
            "Выйти?",
            MessageBoxButtons.YesNo) == DialogResult.Yes)
            e.Cancel = false;
    }
}
```

10. Какие классы, свойства и методы используются для загрузки - сохранения XML документа?

Загрузка - выгрузка документа:

```
XmlDocument d = new XmlDocument();
d.LoadXml(
    "<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n<Tbl></Tbl>");
d.Save("out.xml");
d.Load("in.xml");
```

11. Какие классы, свойства и методы используются для извлечения узлов из XML документа?

Выбрать из документа первый узел:

```
XmlNode tbl = d.DocumentElement;
```

Список узлов с заданным именем (выбрать по имени или выбрать всех детей):

```
XmlNodeList rs = d.GetElementsByTagName("Rec");
XmlNodeList r = rs[0].ChildNodes;
```

Взять текст узла:

```
r[0].InnerText;
```

12. Какие классы, свойства и методы используются для добавления узла в другой XML узел?

Создать два узла и один добавить к другому:

```
XmlNode nd = d.CreateElement("Rec");  
XmlNode fld = d.CreateElement("fld");  
fld.InnerText = "field";  
nd.AppendChild(fld);
```

13. Какие классы, свойства и методы используются для добавления атрибута к XML узлу?

Создать атрибут узла и добавить к узлу:

```
XmlNode attr = d.CreateNode(XmlNodeType.Attribute, "id", null);  
attr.Value = 25.ToString();  
nd.Attributes.SetNamedItem(attr);
```

14. Назовите несколько управляющих элементов. Как управляющий элемент добавляется к окну?

```
Button myButton = new Button();  
myButton.Location = new Point(20, 20);  
myButton.Text = "button";  
this.Controls.Add(myButton);  
Label my = new Label();  
my.Location = new Point(20, 40);  
my.Text = "label";  
this.Controls.Add(my);  
TextBox my = new TextBox();  
my.Location = new Point(20, 60);  
my.Text = "textbox";  
this.Controls.Add(my);
```

15. Какое ключевое слово предназначено для создания метода с переменным числом параметров? Приведите пример.

Для создания метода, принимающего переменное число однотипных параметров используется ключевое слово `params`. Примером метода, который принимает переменное число параметров, является метод `Console.WriteLine`.

16. Назовите управляющие элементы, предназначенные для автоматического размещения содержащихся в них управляющих элементов.

Управляющий элемент `FlowLayoutPanel` предназначен для горизонтального или вертикального расположения управляющих элементов. Управляющий элемент `TableLayoutPanel` предназначен для размещения управляющих элементов в виде таблицы.

17. Приведите пример использования шаблона для создания словаря.

Пример создание словаря из полей ввода и умолчательных значений к ним.

```
TextBox a = new TextBox();
TextBox b = new TextBox();
Dictionary<TextBox, string> flds =
    new Dictionary<TextBox, string> ();
flds.Add(a, "Pupkin");
flds.Add(a, "22");
```

18. Для чего используется свойство Padding?

Свойство Padding у панелей Panel используется для того, что бы задать отступы от внешней границы панели до её внутренних контролов.

```
Panel p = new Panel();
// left, top, right, bottom
p.Padding = new Padding (0,0,0,8);
```

19. Для чего можно использовать управляющий элемент Panel?

Управляющий элемент можно использовать для группирования внутри него других управляющих элементов, например, полей ввода - TextBox и меток - Label.

20. Какое свойство и перечисление используется для размещения управляющих элементов?

Для размещения управляющих элементов используется свойство Dock и перечисление DockStyle. При расположении в окне или другом управляющем элементе, управляющий элемент можно прижимать вверх, вниз, влево, вправо или (DockStyle.Top, DockStyle.Bottom, DockStyle.Left, DockStyle.Right, DockStyle.Fill) полностью заполнять им свободное пространство.

21. Какие классы используются для просмотра табличных данных?

Для просмотра табличных данных используются следующие классы: DataGridView - таблица, DataGridViewColumn - колонка таблицы, DataGridViewRow - строка таблицы, DataGridViewCell - ячейка таблицы, DataGridViewCellCollection - коллекция ячеек в строке.

22. Приведите пример настройки управляющего элемента для просмотра табличных данных.

```
DataGridView
dgv = new DataGridView();
dgv.ReadOnly = true;
dgv.AllowUserToAddRows = false;
dgv.BackgroundColor = Color.White;
dgv.AutoSizeColumnsMode
    = DataGridViewAutoSizeColumnsMode.Fill;
```

23. Приведите пример добавления в управляющий элемента для просмотра табличных данных трех колонок.

```
DataGridView
dgv = new DataGridView();
dgv.ColumnCount = 3;
dgv.Columns[0].Name = String.Format("Pole{0}", 1);
dgv.Columns[1].Name = String.Format("Pole{0}", 2);
dgv.Columns[2].Name = String.Format("Pole{0}", 3);
```

24. Какая коллекция и метод используется для добавления строчек в таблицу?

Для добавления строчек в управляющий элемент для просмотра табличных данных DataGridView используется коллекция `dgv.Rows` и её метод `Add()`.

```
DataGridView      dgv = new DataGridView();
dgv.ColumnCount = 3;
string [] flds = new string[3];
flds[0] = "aa" ;   flds[1] = "bb" ; flds[2] = "cc" ;
dgv.Rows.Add(flds);
```

25. Как гарантировать видимость заданной ячейки в управляющем элементе для просмотра табличных данных DataGridView ?

```
DataGridView
dgv = new DataGridView();
dgv.CurrentCell = dt.Rows[0].Cells[0];
```

26. Какие свойства, классы, методы и события применяются для использования главного меню в окне?

Для свойства окна `Menu` используются классы `MainMenu` и `MenuItem`; метод `Add()` - для добавления одного элемента меню к другому; событие `Click` - для обработки нажатия на элемент меню. Для свойства окна `MainMenuStrip`; классы `MenuStrip` и `ToolStripMenuItem`; Обработчики события `Click` создаются для каждого элемента меню.

27. Какие классы и свойства применяются для использования панели инструментов в окне?

Для использования панели инструментов применяется классы `ToolBar` и `ToolStripButton` (или более новые классы: `ToolStrip`, `ToolStripButton`). Для размещения панели инструментов используются метод `Add()` коллекции окна `Controls` и свойство `Dock`, которое позволяет размещать в панель одной из сторон окна. Обработчик события `Click` создается для всей панели, а не для каждой кнопки. В классе `ToolStripButtonClickEventArgs` свойство `Button` содержит кнопку, которую нажимал оператор.

28. Приведите пример информационного управляющего элемента, отличного от Label.

Информацию, описывающую состояние окна, удобно демонстрировать при помощи класса StatusStrip, объект которого может содержать несколько различных меток класса ToolStripStatusLabel.

```
StatusStrip statStrip = new StatusStrip();
statStrip.Items.Add(new ToolStripStatusLabel());
statStrip.Items[0].Text = "Hello";
Controls.Add(statStrip);
```

29. При помощи какого ключевого слова гарантируется освобождение захваченных программных ресурсов (имеются ввиду файлы, нити (потoki команд), сокеты и соединения с базой данных)? Приведите пример.

Для гарантированного освобождения ресурсов (вызова у соответствующего объекта метода Dispose()) необходимо использовать третью форму ключевого слова using.

```
using (TextReader trdr
    = File.OpenText(@"./example.cs")){
    string str = trdr.ReadToEnd();
    Console.WriteLine(">>>\n{0}\n<<<", str);
}
```

30. Какой класс используется для создания (или чтения, или вывода) текстов в заданной кодировке? Пример создания и использования кодировки 1251.

Для того, что бы явно задавать кодировки текста, предназначен класс Encoding.

```
Encoding enc = Encoding.GetEncoding(1251);
string str;
using(StreamReader sr
    = new StreamReader("./1251.txt", enc)){
    while ((str = sr.ReadLine()) != null) {
        Console.WriteLine(str);
    }
}
```

14.5 Дополнительные Возможности Программирования В .Net

1. Какие средства существуют в .Net для определения типа во время выполнения?

- операция typeof (тип) - возвращает объект класса System.Type (тип) по заданному обозначению типа.

```
Type t = typeof(int);
```


- метод GetType() - возвращает тип выражения

```
Type t = (1+1).GetType();
```

- оператор is - возвращает true если тип выражения слева находится в дереве наследования типа, заданного справа.

```
Console.WriteLine("result is {0}", 1 is int);
```

2. Какая утилита используется для генерации ресурсов?

Утилита resgen.exe используемая с ключом /compile из XML - файла генерирует откомпилированный файл ресурсов с расширением .resources.

3. Какой класс используется для генерации ресурсов?

Для создания XML - файлов с ресурсами (расширение resx) используется класс ResXResourceWriter

```
ResXResourceWriter rw =
    new ResXResourceWriter("onePicture.resx");
Image i = Image.FromFile("picture.png", true);
rw.AddResource("picture", i);
rw.Close();
```

4. При помощи какого класса осуществляется доступ к ресурсам?

Доступ к отдельным файлам в ресурсах осуществляется при помощи менеджера ресурсов

```
global::System.Resources.ResourceManager rm
    = new global::System.Resources.ResourceManager(
        "onePicture" // имя файла ресурсов
        , typeof(MyProgram).Assembly);
// имя файла-картинки
Image arrImage = (Image)rm.GetObject("picture");
```

5. Какие методы используются для преобразования целых чисел и сточек (string) в цвет (Color)?

```
Color clr = Color.FromArgb(
    100 // прозрачность
    ,10 // красный
    ,20 // зеленый
    ,30 // голубой
);
Color c = Color.FromName("DarkRed");
```

6. Назовите и охарактеризуйте основные классы, которые используются для рисования в окнах .Net.

Класс Graphics предоставляет методы для рисования основных примитивов (линии, кривые, эллипсы, многоугольники, прямоугольники и так

далее) в окнах. В обработчик события Paint объект этого типа передается в параметре PaintEventArgs. Этот объект можно себе представлять виртуальную бумагу. Классы Pen (карандаш) и Brush (кисточка) предоставляют объекты двух типов, при помощи которых выполняется рисование.

7. Создайте красный карандаш при помощи словесного описания цвета.

```
Pen P1
    = new Pen(Color.FromName("Red"), 1);
```

8. Создайте полупрозрачную синюю кисточку.

```
Brush trnsBlueBrush = new SolidBrush(
    Color.FromArgb(128, 0, 0, 255));
```

9. Какой метод панели следует вызывать, что бы перерисовать панель?

Для перерисовки панели (равно как и управляющего элемента) или её части следует вызывать метод Invalidate() или Invalidate(Rectangle)

```
Panel pan = new Panel();
pan.Invalidate();
```

10. Приведите пример обработчика события Paint для рисования ломаной.

Рисование ломаной выполняется при помощи метода DrawLines (Pen, Point[]).

```
void Paint1(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Point[] px1 = new Point[3];
    px1 [0]      = new Point (10, 10);
    px1 [1]      = new Point (50, 50);
    px1 [2]      = new Point (10, 90);
    g.DrawLines(Pens.Blue, px1);
}
```

11. Как создаются всплывающие подсказки на элементах управления (например, на ToolBarButton)?

Чтобы создать всплывающую подсказку на кнопке панели инструментов надо присвоить соответствующий текст свойству ToolTipText.

```
ToolBarButton tlbExit = new ToolBarButton("Exit");
tlbExit.ToolTipText =
    "Нажмите для выхода из приложения.";
```

12. Каким методом измеряют размер текста в пикселях? Приведите пример.

```
String      s = "Hello, world!";
Font        f = new Font("Times New Roman", 14);
StringFormat sf =
    new StringFormat(
//      StringFormatFlags.DirectionVertical
      StringFormatFlags.DirectionRightToLeft
    );
SizeF sizef = g.MeasureString(s, f,
    Int32.MaxValue, sf);
```

13. Каким методом выводят текст в области окна (без объекта класса Label)?
Какие дополнительные классы для этого требуются?

Для вывода заданного текста *s* в заданном месте требуется задать шрифт *f*, способ расположения текста *sf* и прямоугольник *rf*, в котором текст должен выводиться.

```
String      s = "Hello, world!";
Font        f = new Font("Times New Roman", 14);
StringFormat sf =
    new StringFormat(
//      StringFormatFlags.DirectionVertical
      StringFormatFlags.DirectionRightToLeft
    );
RectangleF rf = new RectangleF
    // x  y  width height
    (20, 20, 100, 30);
g.DrawString(s, f, Brushes.Black, rf, sf);
```

14. Какой класс используется для перемещения отображаемых в окне объектов в вертикальном направлении? Приведите пример инициализации.

```
VScrollBar vSB = new VScrollBar();
vSB.Maximum = 91;      // максимальное число
vSB.Value    = 45;      // начальное
vSB.Dock     = DockStyle.Right;
```

15. Какой класс используется для перемещения отображаемых в окне объектов в горизонтальном направлении? Приведите пример инициализации.

```
HScrollBar hSB = new HScrollBar();
hSB.Maximum = 91;      // максимальное число
hSB.Value    = 45;      // начальное
hSB.Dock     = DockStyle.Top;
```

16. В каком классе содержатся обычные математические функции? Приведите пример.

Для вычисления некоторых математических функций написаны статические методы класса *Math* с соответствующими названиями.

```
double a = Math.Cos(0);
```

17. Какой класс используется для низкоуровневого (побайтного) ввода - вывода? Приведите пример.

Для низкоуровневого ввода - вывода используется класс `Stream`. Следующий код выведет в стандартный вывод символ 'я' в узкой кодировке 1251.

```
byte[] b = new byte[1];
Stream so = Console.OpenStandardOutput();
b[0]=(byte)255;
so.Write(b, 0, 1);
```

18. Какой класс используется для выполнения `Http` - запросов? Приведите пример скачивания файла.

Для выполнения `Http` - запросов используется классы `HttpWebRequest` и `HttpWebResponse`. Объект класса `HttpWebRequest` можно создать при помощи статического метода `WebRequest.Create(string)`. Для получения ответа используется метод `GetResponse()` класса `HttpWebRequest`.

```
byte[] b = new byte[1]; int ch ;
Stream so = Console.OpenStandardOutput();
HttpWebRequest req= (HttpWebRequest)
    WebRequest.Create("http://google.ru");
HttpWebResponse resp
= (HttpWebResponse) req.GetResponse();
Stream = resp.GetResponseStream();
while (-1 != (ch=istrm.ReadByte()) ){
    b[0]=(byte)ch;
    so.Write(b, 0, 1);
}
```

19. Как можно узнать положение координат указателя мышки 1) на экране, 2) в окне?

Положение указателя курсора можно хранится в объекте `Cursor.Position`. Для пересчета в окно (или панель) необходимо использовать метод `Control.PointToClient(Point)` или, например, аргументы обработчика события `MouseClick` - `MouseEventArgs.X` и `MouseEventArgs.Y`.

```
void _MouseClicked(Object sender, MouseEventArgs e)
{
    Console.WriteLine(
        "MouseEventArgs x/y: {0}/{1}", e.X
        , e.Y);

    Console.WriteLine(
        "Position x/y: {0}/{1}", Cursor.Position.X
        , Cursor.Position.Y);

    Point mouseLoc =
        this.PointToClient(Cursor.Position);
    Console.WriteLine(
        "mouseLoc x/y: {0}/{1}", mouseLoc.X
        , mouseLoc.Y);
}
```

20. Какие классы, свойства и методы используются для создания контекстного меню?

Для создания контекстного меню используются либо класс `ContextMenuStrip`, свойство элемента управления `ContextMenuStrip`; либо класс `ContextMenu` и свойство элемента управления `ContextMenu`.

```
#if OLD
    ContextMenu = new ContextMenu();
#else
    ContextMenuStrip = new ContextMenuStrip();
#endif
```

21. Какие классы, свойства и методы используются для изменения (добавления, удаления элементов) контекстного меню?

В случае контекстного меню типа `ContextMenuStrip`, элементы меню имеют тип `ToolStripMenuItem` и добавляются в коллекцию `Items` (удаляются из `Items`).

```
ContextMenuStrip.Items.Clear();
ToolStripMenuItem mi
    = new ToolStripMenuItem("in Region");
ContextMenuStrip.Items.Add(mi);
```

В случае контекстного меню типа `ContextMenu`, элементы меню имеют тип `MenuItem` и добавляются в коллекцию `MenuItems` (удаляются из `MenuItems`).

```
ContextMenu.MenuItems.Clear();
MenuItem mi = new MenuItem("in Region");
ContextMenu.MenuItems.Add(mi);
```

22. Приведите пример создания контекстного меню.

В конструктор окна добавить код

```
ContextMenuStrip = new ContextMenuStrip();
ContextMenuStrip.Items.Add("item 1");
ContextMenuStrip.Items.Add("item 2");
MouseDown += Control1_MouseClick;
```

И обработчик нажатия на клавишу мышки должен содержать следующую строчку.

```
void _MouseDown(Object sender, MouseEventArgs e)
{
    ContextMenuStrip.Show(Cursor.Position);
}
```

23. Какими средствами можно проверить принадлежит ли заданная точка сложной геометрической фигуре?

Для проверки попадает ли заданная точка в сложную геометрическую фигуру требуется использовать классы `GraphicsPath`, `Region` и его метод `IsVisible(Point)`. Геометрическую фигуру требуется разделить на простые элементы (ломанные, прямоугольники, сектора, кривые и так далее). Каждую из таких простых фигур добавлять в объект класса `GraphicsPath`, а из него позднее создать объект класса `Region`.

```
// pnts ломаная
GraphicsPath gp = new GraphicsPath();
for (int i = 0; i < pnts.Length - 1; i++)
{
    gp.AddPolygon(mkPolygon(pnts[i]
        , pnts[i+1])
    );
}
Region r = new Region(gp);
if (r.IsVisible(new Point(10, 20)))
;
else
;
```

24. Что такое процесс и какой класс используется для управления выполняемыми на компьютере процессами и запуска новых (дочерних) процессов?

Процесс это скомпилированное JIT компилятором и загруженное в оперативную память приложение, которому операционная система присвоила `Process Idendicator (pid)` Для управления и запуска процессов используется классы `Process` и `ProcessStartInfo`. Можно получить список выполняющихся в текущий момент процессов, список модулей (exe и dll) заданного процесса, изменить приоритет процесса, прервать его работу. Пример запуска нового процесса (редактор `notepad` откроет находящийся в текущем каталоге исходный код `Program.cs`):

```
ProcessStartInfo sI = new ProcessStartInfo ();
sI.FileName = "notepad.exe";
sI.Arguments = "Program.cs";
int rc;
using (Process p = Process.Start(sI)) {
    p.WaitForExit();
    rc = p.ExitCode;
}
```

25. Что такое код завершения приложения? Каким способом приложение может задать код завершения (или код возврата)?

Код завершения процесса (приложения) - это целое число, которое передается в запустивший приложение процесс. Традиционно 0 означает, что приложение отработало нормально. Код завершения задается либо в методе `int Main()` оператором `return`; либо в любом другом месте при помощи вызова метода `Exit(int)`;

```
// что-то пошло не так.
Environment.Exit(1);
```

26. Что такое поток команд (нить)? Какой класс используется для запуска и управления нитями?

Поток команд (нить) - это последовательность команд процесса, которой операционная система выдает время процессора. Для запуска и управления нитями используется класс `System.Threading.Thread`. Нить должна иметь точку входа, начинает выполняться при помощи метода `Start(object)`. Основная нить при помощи метода `Join()` может ждать завершения запущенной нити. Методы `Suspend()` и `Resume()` позволяют приостанавливать и продолжать выполнение команд нити. Свойство `IsBackground` определяет будет ли нить основной или фоновой. Метод `Main` будет ждать завершения основных нитей и оборвет работу фоновых (при этом не гарантируется выполнение операторов из секций `finally` фоновых нитей).

27. Какой класс используется для организации исключительного доступа нитей к, например, целым переменным?

Для исключительного доступа к целым (а также вещественным) переменным используется класс `System.Threading.Interlocked`. Он содержит несколько статических методов, которые позволяют изменять значения и сравнивать целые и вещественные переменные. Например:

- метод `int Increment (ref int a)` - увеличивает переменную на 1.
- метод `int Decrement (ref int a)` - уменьшает переменную на 1.
- метод `int Add (ref int a, int b)` - увеличивает или уменьшает переменную `a` на `b`.
- метод `int Add (ref int a, int b)` - увеличивает или уменьшает переменную `a` на `b`.

28. Что такое блок синхронизации и какие классы и методы предназначены для работы с ними.

Блоки синхронизации - средство .Net, позволяющее гарантировать исключительный доступ к любым объектам. Для работы с ним используется класс `System.Threading.Monitor` и его статические методы `Enter(object o)` и `Exit(object o)`;

```
class p {
    public int    numbers = 0;
    public void work(){
        Monitor.Enter (this);
        numbers ++;
        Monitor.Exit (this);
    }
}
```

29. Для чего используется оператор `lock`? Приведите пример его применения.

Оператор `lock` используется так же как и методы `Monitor.Enter(object o)` / `Monitor.Exit (object o)` и, в отличие от методов класса `Monitor`, он гарантирует выход из блока синхронизации, то есть, вызов метода `Monitor.Exit (object o)`.

```
class p {
    public int    numbers = 0;
    public void work(){
        lock (this){
            numbers ++;
        }
    }
}
```

30. Как организовать исключительный доступ нитей к статическим переменным?

Что бы организовать исключительный доступ нитей к статическим переменным в сочетании с методами `Monitor.Enter(object o)` / `Monitor.Exit (object o)` или оператором `lock` необходимо использовать операцию `typeof`. Пример:

```
static class p {
    public int    numbers = 0;
    public void work(){
        lock (typeof(p)){
            numbers ++;
        }
    }
}
```

15 НЕКОТОРЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И ПРИМЕРЫ

Для лучшего понимания тонкостей использования основополагающих понятий типа, класса и объекта дополнительно рекомендуется ознакомиться с монографией Люки Карделли и Мартина Абади 'Теория Объектов', см. [1]. Для некоторых глав из неё существует перевод см. [34]. Скорее всего, беглый взгляд на эту работу может привести читателя к мысли, что для освоения темы требуются существенные усилия. И это правда, даже такие классики как Страуструп (который под классом понимал пользовательский тип, то есть, класс считал типом), не совсем осознавали трудности связанные с этими понятиями. Наверно, одним из первых результатов, указывающий на различие между понятиями класса и типа, равно как и понятиями наследования и выделения подтипа, является принцип подстановки Лисков. В нем явно указываются условия, при которых один тип можно считать подтипом другого. Пример использования операции наследования, который приводит к нарушению свойства быть подтипом (класс ребенка нельзя трактовать как имеющий тип класса родителя)

можно найти в работе Роберта Мартина (см. [4]), обсуждение этого примера с формальной стороны, можно найти в [24]. Более сложное и неожиданное следствие о том, что тип класса целых чисел не является подтипом класса вещественных чисел (если целые наследуются от вещественных) и наоборот (при попытке наследования вещественных от целых) обсуждается в работе [32]. В ней же можно ознакомиться с несколькими другими мнениями по поводу взаимоотношений типов и классов. Но окончательная формулировка решения данного вопроса принадлежит Бертрону Мейеру (см. [5, Основы объектно-ориентированного программирования]): Тип - это спецификация (то есть, набор требований), а класс - это реализация спецификации на одном из языков программирования.

15.1 Исходные текст всех примеров и лабораторных

Исходные тексты всех примеров и лабораторных можно скачать по адресу: http://agp1.adr.com.ua/docs/cs_1.zip либо http://agp1.adr.com.ua/docs/cs_1.z.

15.2 Конвертация файлов и абстрактные классы

Следующий пример показывает идею, которая снижает накладные расходы на сопровождение уже разработанного программного обеспечения. В данном примере - разработан метод Program.txt2bin для конвертации текстового файла в двоичный с возможной сортировкой результата. Будем считать, что метод пишется в терминах абстрактного класса X из предположений, что текстовый файл содержит одну запись на строку, записи могут быть отсортированы, объекты из строки с записью создаются методом X.str2rec, а выводятся в двоичный файл методом X.rec2bin. Кроме этих предположений о структуре текстового и двоичного файлов, равно как и свойствах записей не известно ничего. См.

```
---- File:./cs/2bin/p.cs

using System;
//
using System.Collections.Generic;
//using System.Linq;
//using System.Text;
//using System.Collections;
using System.IO;

namespace ToBin
{
    abstract
    class X:IComparable<X> {
        static
        protected string[] sep = { " ", "\t" };    // separators
        abstract
        public X str2rec(string rec);

        abstract
```

```

        public void rec2bin(BinaryWriter bw);

        virtual
        public int CompareTo(X o) {
            return 0;
        }
    }

}

class Program
{
    static void Main(string[] args)
    {
        //
        X rec = new i2();
        // X rec = new r2();
        using (BinaryWriter bw = new BinaryWriter(File.Open("1.bin", FileMode.Create))){
            txt2bin ( Console.In, bw, true, rec);
        }
    }

    static
    void txt2bin (TextReader tr, BinaryWriter bw, bool sort, X rec ){

        string r;
        List <X> ls = new List<X> ();
        while ((r = tr.ReadLine())!=null){
            ls.Add(rec.str2rec(r));
        }
        if (sort)
            ls.Sort();
        for (int i = 0; i < ls.Count; i++)
            ls[i].rec2bin(bw);
    }
}

}

---- End Of File:./cs/2bin/p.cs

```

После окончания разработки (и рассылки ста тысячам миллионам клиентов ;)) динамически подгружаемой библиотеки с кодом метода Program.txt2bin) можно наследовать сколько угодно конкретных классов (тут приводится два: i2 и r2), при этом исходную динамически подгружаемую библиотеку менять нет необходимости. Требуется только создать объект правильного типа, в нашем примере это строки

```
X rec = new i2();
```

или

```
X rec = new r2();
```

и передать его в обсуждаемый метод:

```
txt2bin ( Console.In, bw, true, rec);
```

Пример наследования записи из двух целых чисел:

```
---- File:./cs/2bin/i2.cs
```

```
using System;
using System.Collections.Generic;
using System.IO;

namespace ToBin{
    class i2: X {
        int a;
        int b;

        public i2(): this(0,0){}

        public i2(int x, int y){
            a=x;
            b=y;
        }
        override
        public int CompareTo(X o){
            return a.CompareTo(((i2)o).a);
        }
        override
        public X str2rec(string rec){
            string [] flds = rec.Split(sep, StringSplitOptions.RemoveEmptyEntries);

            if (flds.Length >= 2) {
                a = int.Parse(flds[0]);
                b = int.Parse(flds[1]);
            }
            return new i2(a, b);
        }
        override
        public void rec2bin(BinaryWriter bw){
            bw.Write(a);
            bw.Write(b);
        }
    }
}
```

```
---- End Of File:./cs/2bin/i2.cs
```

Пример наследования записи из двух вещественных чисел:

```
---- File:./cs/2bin/r2.cs
```

```
using System;
using System.Collections.Generic;
using System.IO;
```

```

namespace ToBin
{
    class r2: X {
        double a;
        double b;

        public r2(): this(0.0,0.0){}

        public r2(double x, double y){
            a=x;
            b=y;
        }
        override
        public int CompareTo(X o){
            return a.CompareTo(((r2)o).a);
        }
        override
        public X str2rec(string rec){
            string [] flds = rec.Split(sep, StringSplitOptions.RemoveEmptyEntries);

            if (flds.Length >= 2) {
                a = double.Parse(flds[0]);
                b = double.Parse(flds[1]);
            }
            return new r2(a, b);
        }
        override
        public void rec2bin(BinaryWriter bw){
            bw.Write(a);
            bw.Write(b);
        }
    }

}

---- End Of File:./cs/2bin/r2.cs

```

Более жизненный примеры разработки программного обеспечения в абстрактном и конкретном стиле можно найти в переводе 'Неполное Руководство по SQLite для пользователей Windows' раздел 3.6 Ado.Net провайдер (см. [35, п. 3.6.1.1].).

15.3 Пример использования лямбда выражений

В данном примере два раза используется лямбда выражение:

- `a.FindAll(x => i % x == 0).Count == 0`
 тут выбирается подсписок делителей текущего проверяемого числа *i*, если он пустой, то число *x* - простое;

- `a.ForEach(x => Console.WriteLine(x));`
 тут для каждого элемента списка выполняется вывод его в стандартный вывод.

```

---- File:./cs/lambda/p.cs

using System;
using System.Collections.Generic;

class p{
    static void Main()    {
        int n, j = 0;
        n = Int32.Parse(Console.ReadLine());
        List<int> a = new List<int>(); //создаем список типа int
        DateTime date1 = DateTime.Now; //фиксируем начальное время программы
        for (int i = 2; i < n; i++)    {
            if (a.FindAll(x => i % x == 0).Count == 0) //если i не делится на любое число x из списка
                a.Add(i); //добавляем в список значение i
        }
        a.ForEach(x => Console.WriteLine(x)); //выводим все значения из списка
        Console.Error.WriteLine((DateTime.Now - date1).TotalSeconds);
    }
}

---- End Of File:./cs/lambda/p.cs

```

15.4 Явно заданная продолжительность тика

Пример импортирует из системной библиотеки `winmm.dll` две функции: `timeBeginPeriod` и `timeEndPeriod`. Между вызовами этих функций продолжительность системного тика меняется на заданное в функциях значение. Стандартная продолжительность тика 1/64 секунды. Затем при помощи `Stopwatch` засекается общая продолжительность сна нити для тысячи засыпаний. На компьютерах с одним одноядерным процессором два цикла в примере выдают существенно разные времена, второе время более длинное. Это означает, что точность измерений времени при стандартном тике хуже.

```

---- File:./cs/timer/timer.cs

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading;

class MMTimersTest
{
    [DllImport("winmm.dll")]
    private static extern uint timeBeginPeriod(uint period);
    [DllImport("winmm.dll")]
    private static extern uint timeEndPeriod(uint period);
}

```

```

static void Main(string []args)
{
    if (args.Length < 1)
    { Console.WriteLine("usage: \n timer XXXX\n where XXXX msec - period of sleeping (1000 time
    else if (args[0] == "-?")
    { Console.WriteLine("usage: \n timer XXXX\n where XXXX msec - period of sleeping"); }
    else
    {
        int l = Int32.Parse(args[0]);
        uint period = 1;

        uint rc = timeBeginPeriod(period); //<<<<<====
        Stopwatch stopWatch = new Stopwatch(); stopWatch.Start();
        for (int i = 1; i <= 1000; i++) // первый цикл
            Thread.Sleep(l);
        stopWatch.Stop();
        rc = timeEndPeriod(period); //<<<<<====
        Console.WriteLine(" timeEndPeriod, rc = {0}", rc);

        TimeSpan ts = stopWatch.Elapsed;
        string elapsedTime
            = String.Format("{0:0.000} secs",ts.TotalMilliseconds/1000 );

        Console.WriteLine("RunTime with non standart tick: " + elapsedTime);

        Stopwatch stopWatch1 = new Stopwatch(); stopWatch1.Start();
        for (int i = 1; i <= 1000; i++) // первый цикл
            Thread.Sleep(l);
        stopWatch1.Stop();
        ts = stopWatch1.Elapsed;
        elapsedTime
            = String.Format("{0:0.000} secs" , ts.TotalMilliseconds/1000 );
        Console.WriteLine("RunTime with standart tick:" + elapsedTime);
    }
}
}

---- End Of File:./cs/timer/timer.cs

```

15.5 Явно заданное кодирование файла

В разделе расположены примеры в который кодировка файлов задается явно. Простая версия. Строки со стандартного ввода читаются в кодировке 866 страницы (MS DOS) и выводятся в кодировке страницы 1251 (Windows 95). Класс Encoding расположен в области видимости System.Text.

```

---- File:./cs/encoding2/866to1251.cs

```

```

using System;
using System.IO;
using System.Text;

```

```

class app {
    static int Main() {
        // она и так 866, я хотел явно записать
        Console.InputEncoding = Encoding.GetEncoding(866);

        Console.OutputEncoding = Encoding.GetEncoding(1251);
        // после этого портится кодировка консоли
        string str;

        while ((str = Console.ReadLine()) != null) {
            Console.WriteLine(str);
        }
        return 0;
    }
}

```

---- End Of File: ./cs/encoding2/866to1251.cs

Второй пример показывает как открывать новый StreamReader с заданной кодировкой. В частности страница 65001 задает кодировку UTF-8

---- File: ./cs/encoding/utf8.cs

```

using System;
using System.Web;
using System.Threading;
using System.IO;
using System.Net;
using System.Text;
using System.Text.RegularExpressions;
using System.Collections.Specialized;

class app {
    static int Main() {

        Console.WriteLine("type is {0}", (1+1).GetType());
        Console.WriteLine("result is {0}", 1 is int);
        // Console.WriteLine("result is {0}", typeof(1));

        Encoding enc = Encoding.GetEncoding(65001);
        string str;
        using(StreamReader sr = new StreamReader("./utf-8.txt", enc)){
            while ((str = sr.ReadLine()) != null) {
                Console.WriteLine(str);
            }
        }
        return 0;
    }
}

```

```
}
```

```
---- End Of File:./cs/encoding/utf8.cs
```

15.6 Контекстное меню

Раздел [13.4](#) содержит объяснения к примеру:

```
---- File:./cs/menustrip/form1.cs
```

```
//#define OLD

using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace gdiDrawLine
{
    public partial class Form1 : Form
    {
        Point[] points;
        Pen pen;
        Pen pen1;
        public Form1() {
            pen = new Pen(Brushes.Black, 5);
            pen1 = Pens.Red;
            Paint += Draw_Paint;
#if OLD
            ContextMenu = new ContextMenu();
            Console.WriteLine("ContextMenu version");
#else
            ContextMenuStrip = new ContextMenuStrip();
            Console.WriteLine("ContextMenuStrip version");
#endif
            MouseClick += Control1_MouseClick;
            Point[] po = {
                new Point(5,5), //5, 10
                new Point(50,50),
                new Point(100,5),
                new Point(150, 50),
                new Point(200, 5),
            };
            points = po;
        }

        void ContextMenuStrip_Click(object sender, EventArgs e)
        {
            Console.WriteLine("MenuItemen have been clicked");
        }
    }
}
```



```

    }

    void Control1_MouseClick(Object sender, MouseEventArgs e)
    {
/*
        Console.WriteLine("Position x/y: {0}/{1}", Cursor.Position.X
                           , Cursor.Position.Y);
        Console.WriteLine("MouseEventArgs x/y: {0}/{1}", e.X    // 8[] возвращает прошлые координаты
                           , e.Y);    //
        Console.WriteLine("MouseEventArgs Location x/y: {0}/{1}", e.Location.X
                           , e.Location.Y); // 8[] возвращает прошлые координаты
        Point mouseLoc = PointToClient(Cursor.Position);
        Console.WriteLine("mouseLoc x/y: {0}/{1}", mouseLoc.X
                           , mouseLoc.Y);

        mouseLoc = e.Location;    */
        Region r = mkRgn();
        if (r.IsVisible(e.Location))
        {
#if OLD
            ContextMenu.MenuItems.Clear();
            MenuItem mi = new MenuItem("in Region");
            ContextMenu.MenuItems.Add(mi);
#else
            ContextMenuStrip.Items.Clear();
            ToolStripMenuItem mi = new ToolStripMenuItem("in Region");
            ContextMenuStrip.Items.Add(mi);
#endif
            mi.Click += ContextMenuStrip_Click;
        }
        else
        {
#if OLD
            ContextMenu.MenuItems.Clear();
            MenuItem mi = new MenuItem("Not in Region");
            ContextMenu.MenuItems.Add(mi);
#else
            ContextMenuStrip.Items.Clear();
            ToolStripMenuItem mi = new ToolStripMenuItem("Not in Region");
            ContextMenuStrip.Items.Add(mi);
#endif
            mi.Click += ContextMenuStrip_Click;
        }
    }

    public void Draw_Paint(object sender, PaintEventArgs ea)
    {
        Graphics g = ea.Graphics;
        Text = "gdiDrawLine";
        // нарисовал жирную черную
        g.DrawLine(pen, points);
        for (int i = 0; i < points.Length - 1; i++)
        {
            // рисую красную ломаную рядом
            g.DrawPolygon(pen1, mkPolygon(points[i]

```

```

        , points[i+1])
    );
}
}

Region mkRgn()
{
    GraphicsPath gp = new GraphicsPath();
    int i;
    for (i = 0; i < points.Length - 1; i++)
    {
        gp.AddPolygon(mkPolygon(points[i]
            , points[i+1])
        );
    }
    Region r = new Region(gp);
    return r;
}

Point [] mkPolygon (Point a, Point b){
    Point[] m = new Point[5];
    m[0] = a;
    m[1] = b;
    m[2] = new Point(b.X + 4, b.Y);
    m[3] = new Point(a.X + 4, a.Y);
    m[4] = a;
    return m;
}
}
}

```

---- End Of File:./cs/menustrip/form1.cs

15.7 Скачивание файла

---- File:./cs/wget/wget.cs

```

//define TEST

using System;
using System.Data;
using System.Threading;
using System.Net;
using System.IO;

class application1
{
    [STAThread]
    static int Main(string[] ars)
    {
        int ch ;
        byte []b =new byte[1];
        Stream istrm=null;
    }
}

```

```

Stream    so  = Console.OpenStandardOutput();

{
    HttpWebRequest req  =
        (HttpWebRequest) WebRequest.Create("http://google.ru");
    HttpWebResponse resp =
        (HttpWebResponse) req.GetResponse();
    istrm  = resp.GetResponseStream();
    while (-1 != (ch=istrm.ReadByte()) ){
        b[0]=(byte)ch;
        so.Write(b, 0, 1);
    }

}
return 0;
}
}

---- End Of File:./cs/wget/wget.cs

```

15.8 Клик снаружи или внутри многоугольника?

Приложение демонстрирует обработку кликов мышки (левого и правого) и простой алгоритм для определения попал ли клик мышки внутрь многоугольника. Многоугольник задается левыми кликами мышки, правый клик мышки тестируется на попадание в многоугольник.

```

---- File:./cs/inpolygon/program.cs

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;

namespace module2_p
{
    class Program : Form
    {
        [STAThread]
        static void Main(string[] args)
        {
            Application.Run(new Program());
        }

        List<Point> points = new List<Point>();

        Panel canvas;
        public Program()
        {
            canvas = new Panel();
            canvas.Dock = DockStyle.Fill;

```

```

        canvas.BackColor = Color.Gray;
        Controls.Add(canvas);
        canvas.MouseClick += Canvas_MouseClick;
        canvas.Paint      += _paint;
    }

    void _paint (object sender, PaintEventArgs e){
        Graphics G = e.Graphics;
        if (points.Count >= 3) {
            G.FillPolygon(new SolidBrush(Color.Blue), points.ToArray());
            Console.WriteLine("count: {0}", points.Count);
        }
    }

    void Canvas_MouseClick(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            points.Add(new Point(((MouseEventArgs)e).X, ((MouseEventArgs)e).Y));
            canvas.Invalidate();
        }
        else if (e.Button == MouseButtons.Right)
        {
            Point click = new Point(((MouseEventArgs)e).X, ((MouseEventArgs)e).Y);
            if (AngleTest(click, points))
                Console.WriteLine("in");
            else
                Console.WriteLine("out");
        }
    }

    bool AngleTest(Point visitor, List<Point> figure)
    {
        double a = 0;
        for (int i = 0; i < figure.Count - 1; i++)
            a -= Angle(figure[i], visitor, figure[i + 1]);

        a -= Angle(figure[figure.Count - 1], visitor, figure[0]);
        Console.WriteLine("sum: {0}", a);

        if (Math.Abs((Math.Abs(a) - 360)) < 0.1) return true;
        else return false;
    }

    double Angle(Point A, Point B, Point C)
    {
        PointF vector1 = new PointF(A.X - B.X, A.Y - B.Y);
        PointF vector2 = new PointF(C.X - B.X, C.Y - B.Y);
        double sin = vector1.X * vector2.Y - vector2.X * vector1.Y;
        double cos = vector1.X * vector2.X + vector1.Y * vector2.Y;
        return Math.Atan2(sin, cos) * (180 / Math.PI);
    }

```

```

    }
}

---- End Of File: ./cs/inpolygon/program.cs

```

15.9 Теннис для двух игроков

Приложение является хорошей иллюстрацией использования таймеров, типов GraphicsPath и Region. Кроме того, приложение является примером работы на клавиатуре двух операторов, левый игрок использует клавиши s и w, правый - клавиши вверх и вниз. Для этого используется обработка событий KeyPressed и KeyReleased. Пример принадлежит Пасечнику М.А.

```

---- File: ./cs/tennis/program.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace SMP_2_2
{
    class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            Application.Run(new Form1());
        }
    }
    class Form1 : Form
    {
        Rectangle player1;
        Rectangle player2;
        Rectangle ball;
        Region player1_reg;
        Region player2_reg;
        Region ball_reg;

        int player1_points;
        int player2_points;
        bool player1_up;
        bool player1_down;
        bool player2_up;
        bool player2_down;
        int delta_x;
        int delta_y;
    }
}

```

```

Timer gameTime;
Timer moveTime;

public Form1() {
    MinimizeBox = false;
    MaximizeBox = false;
    gameTime = new Timer();
    gameTime.Interval = 20;
    moveTime = new Timer();
    moveTime.Interval = 20;
    player1_points = 0;
    player2_points = 0;

    this.Height = 600;
    this.Width = 800;
    this.BackColor = Color.White;
    this.Paint += new PaintEventHandler(Draw1);
    this.KeyDown += new KeyEventHandler(KeyPressed );
    this.KeyUp += new KeyEventHandler (KeyReleased);
    gameTime.Tick += new EventHandler(OnGameTick);
    moveTime.Tick += new EventHandler(OnMoveTick);

    if (MessageBox.Show(
        " Press 'OK' to start."
        ,"Starting the game",MessageBoxButtons.OK
        )==DialogResult.OK) {
        ResetGame();
    }
}

public void Draw1(object sender, PaintEventArgs e) {
    Graphics myGraphics = e.Graphics;
    myGraphics.FillRegion(Brushes.Blue, player1_reg);
    myGraphics.FillRegion(Brushes.LimeGreen, player2_reg);
    myGraphics.FillRegion(Brushes.Red, ball_reg);
}

public void OnGameTick(object sender, EventArgs e) {
    MoveBall();
    if (ball.IntersectsWith(player2))
    {
        delta_x *= -1;
        if (player2_up) {
            delta_y -= 2;
        }
        if (player2_down) {
            delta_y += 2;
        }
    }
    if (ball.IntersectsWith(player1)) {
        delta_x *= -1;
        if (player1_up)
            delta_y -= 2;
        if (player1_down)
            delta_y += 2;
    }
    if (ball.Y > Height-60)

```

```

        delta_y *= -1;
    if (ball.Y < 0)
        delta_y *= -1;
    if (ball.X < -10) {
        player2_points++;
        StopGame();
    }
    if (ball.X > Width) {
        player1_points++;
        StopGame();
    }
}

public void KeyPressed(object sender, KeyEventArgs e) {
    switch (e.KeyCode) {
        case Keys.W:
            player1_up = true;
            break;
        case Keys.S:
            player1_down = true;
            break;
        case Keys.Up:
            player2_up = true;
            break;
        case Keys.Down:
            player2_down = true;
            break;
    }
    if (player1_up || player1_down || player2_up || player2_down) {
        moveTime.Start();
    }
}

public void KeyReleased(object sender, KeyEventArgs e) {
    switch (e.KeyCode) {
        case Keys.W:
            player1_up = false;
            break;
        case Keys.S:
            player1_down = false;
            break;
        case Keys.Up:
            player2_up = false;
            break;
        case Keys.Down:
            player2_down = false;
            break;
    }
}

public void ResetGame() {
    player2 = new Rectangle(Width - 85, Height / 2 - 50, 15, 100);
    player2_reg = new Region(player2);
    player1 = new Rectangle(70, Height / 2 - 50, 15, 100);
    player1_reg = new Region(player1);
    ball = new Rectangle(Width / 2 - 10, Height / 2 - 10, 20, 20);
    GraphicsPath gp = new GraphicsPath();

```

```

gp.AddEllipse(ball);
ball_reg = new Region(gp);
Invalidate();

delta_x = 10;
delta_y = 0;
player2_up = false;
player2_down = false;
player1_up = false;
player1_down = false;
gameTime.Start();
}
public void MoveBall() {
    Rectangle temp = ball;
    ball.Location = new Point(ball.Location.X + delta_x, ball.Location.Y + delta_y);
    GraphicsPath gp = new GraphicsPath();
    gp.AddEllipse(ball);
    ball_reg = new Region(gp);
    Invalidate(ball);
    Invalidate(temp);
}
public void OnMoveTick(object sender, EventArgs e) {
    bool working = false;
    if (player1_up && player1.Y > 2) {
        Rectangle temp = player1;
        player1.Location = new Point(player1.X, player1.Y - 10);
        player1_reg = new Region(player1);
        Invalidate(player1);
        Invalidate(temp);
        working = true;
    }
    if (player1_down && player1.Y < Height - 140) {
        Rectangle temp = player1;
        player1.Location = new Point(player1.X, player1.Y + 10);
        player1_reg = new Region(player1);
        Invalidate(player1);
        Invalidate(temp);
        working = true;
    }
    if (player2_up && player2.Y > 2) {
        Rectangle temp = player2;
        player2.Location = new Point(player2.X, player2.Y - 10);
        player2_reg = new Region(player2);
        Invalidate(player2);
        Invalidate(temp);
        working = true;
    }
    if (player2_down && player2.Y < Height - 140) {
        Rectangle temp = player2;
        player2.Location = new Point(player2.X, player2.Y + 10);
        player2_reg = new Region(player2);
        Invalidate(player2);
        Invalidate(temp);
        working = true;
    }
}

```



```

    }
    if (!working)
        moveTime.Stop();
}

void StopGame(){
    gameTime.Stop();
    moveTime.Stop();

    DialogResult result = MessageBox.Show(
        "Score is " + player1_points
        + ":" + player2_points + "\nWant to continue?"
        , "Continue?", MessageBoxButtons.YesNo);
    if (result == DialogResult.Yes)
        ResetGame();
    else if (result == DialogResult.No)
    {
        string ending_str = "";
        if (player1_points > player2_points)
            ending_str = "Player 1 won!";
        else if (player1_points < player2_points)
            ending_str = "Player 2 won!";
        else
            ending_str = "It's a tie!";
        MessageBox.Show(ending_str, "Game is finished");
        this.Close();
    }
}

}

}

}

---- End Of File:./cs/tennis/program.cs

```

15.10 Построение графиков при помощи управляющего элемента Chart

В окне рисуются два графика различными цветами.

```
---- File:../cs/chart/1.cs
```

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
//using System.Linq;

using System.Windows.Forms.DataVisualization.Charting;
using System.Windows.Forms.DataVisualization;

namespace MethodObch_L1
{
    static class Program
    {
        [STAThread]
```

```

        static void Main()
        {
            Application.Run(new Form1());
        }
    }

class Form1 : Form
{
    public Form1()
    {
        //создаем элемент Chart
        Chart myChart = new Chart();
        //кладем его на форму и растягиваем на все окно.
        Controls.Add(myChart);      // 1 способ добавить упр.элемент на форму

        //      myChart.Parent = this;    /// 2 способ добавить чарт на форму
        myChart.Dock = DockStyle.Fill;
        //добавляем в Chart область для рисования графиков, их может быть
        //много, поэтому даем ей имя.
        myChart.ChartAreas.Add(new ChartArea("Math functions"));
        //Создаем и настраиваем набор точек для рисования графика, в том
        //не забыв указать имя области на которой хотим отобразить этот
        //набор точек.
        Series mySeriesOfPoint = new Series("Sinus");
        mySeriesOfPoint.ChartType = SeriesChartType.Line;
        mySeriesOfPoint.ChartArea = "Math functions";

        for (double x = -Math.PI; x <= Math.PI; x += Math.PI / 10.0)
        {
            mySeriesOfPoint.Points.AddXY(x, Math.Sin(x));
        }

        Series mySeriesOfPoint1 = new Series("Cosinus");
        mySeriesOfPoint1.ChartType = SeriesChartType.Line;
        mySeriesOfPoint1.ChartArea = "Math functions";

        for (double x = -Math.PI; x <= Math.PI; x += Math.PI / 10.0)
        {
            mySeriesOfPoint1.Points.AddXY(x, Math.Cos(x));
        }
        //Добавляем созданный набор точек в Chart
        myChart.Series.Add(mySeriesOfPoint);
        myChart.Series.Add(mySeriesOfPoint1);
    }
}

---- End Of File:./cs/chart/1.cs

```

Для построения приложения требуется явно указать динамически подключаемую библиотеку с нужным управляющим элементом:

```
---- File:./cs/chart/compile.cmd
```

```
csc /r:System.Windows.Forms.DataVisualization.dll -out:a.exe 1.cs
---- End Of File:./cs/chart/compile.cmd
```

15.11 Использование трехмерной графики

Пример не готов.

15.12 Порождение дочернего процесса

```
---- File:./cs/run/program.cs

using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using System.Threading;

namespace app2
{

#pragma warning disable 642
    class Program
    {
        static int Main(string[] args)
        {
            ProcessStartInfo sI = new ProcessStartInfo ();
            sI.FileName = "notepad.exe";
            sI.FileName = "cp.exe";
            sI.Arguments = "Program.cs .Program.cs";
            int rc;
            using (Process p = Process.Start(sI)) {
                p.WaitForExit();
                rc = p.ExitCode;
            }
            Console.WriteLine ("rc = {0}", rc);
            //Environment.Exit(rc);
            return rc;
        }
    }
}

---- End Of File:./cs/run/program.cs
```

15.13 Операция инкремента в двух нитях

В приложении приводится пример синхронизации двух нитей, которые по очереди выполняют операцию инкремента на одной переменной. С помощью условной компиляции можно выполнить синхронизацию потоков (то есть, в данном примере это гарантированное увеличение значения переменной обеими потоками по очереди) при помощи нескольких способов. При помощи класса `Interlocked`, при помощи класса `Monitor`, который облегчает использование так называемой критической секции и при помощи двух способов использования инструкции `lock`.

---- File:./cs/threads/program.cs

```
// #define INTERLOCKED
// #define MONITOR
// #define LOCK
//
#define LOCKSTATIC

using System;
using System.IO;
using System.Threading;
namespace test {
    class m {
        static public int    max = 5; // пять секунд работать
        static public int    sNumbers = 0; ///< количество целых чисел, проверенных средом
        public int numbers = 0;
        [STAThread]
        public static void Main()
        {
            m someMemory      = new m(); // память общая для обеих потоков
            p b = new p("ThreadPriority.Lowest ", someMemory);
            p a = new p("ThreadPriority.Highest", someMemory, ThreadPriority.Highest);

            b.t.Join(); // ожидаем завершение потока b
            a.t.Join(); // ожидаем завершение потока a

            string s;

            #if INTERLOCKED
                s = "interlocked";
            #elif MONITOR
                s = "monitor ";
            #elif LOCK
                s = "lock ";
            #elif LOCKSTATIC
                s = "lockStatic ";
            #else
                #error You have to set at least one macro
            #endif

            Console.WriteLine("\n--{0} version, total(nonstatic/static): {1}/{2}\n", s
```

```

        , someMemory.numbers
        , m.sNumbers
    );
    Console.WriteLine("thread '{0}' finished with {1} numbers"
        , a.name, a.numbers);
    Console.WriteLine("thread '{0}' finished with {1} numbers"
        , b.name, b.numbers);
    Console.WriteLine("\n--{0} version, difference(nonstatic/static): {1}/{2}\n"
        , s
        , someMemory.numbers - a.numbers - b.numbers
        , m.sNumbers - a.numbers - b.numbers
    );
    }
}

class p {
    public Thread t = null;    ///  

    public int numbers = 0;
    public string name = null;

    public p (string o
        , m sharedMemory
        , ThreadPriority p=ThreadPriority.Lowest
    ){
        name = o;
        t = new Thread(work);
        t.Priority = p;
        t.Start(sharedMemory);    // передали общую память в поток
    }

    public void work(object o){
        m mm = (m) o;    // получили общую память в потоке
        DateTime st = DateTime.Now;
        for (DateTime cur = DateTime.Now
            ; (cur - st).TotalSeconds < m.max
            ; cur = DateTime.Now) {
#if INTERLOCKED
            Interlocked.Increment(ref mm.numbers);

#elif MONITOR
            Monitor.Enter (mm);
            mm.numbers ++;
            Monitor.Exit (mm);
#elif LOCK
            lock(mm) { // переменная типа класс, а не структуры
                mm.numbers++;
            }
#elif LOCKSTATIC
            lock(typeof(m)) {
                m.sNumbers ++;    // это общая переменная
            }
#endif
            numbers++;    // это переменная среда, что бы узнать

```

```

        }
    }
}

---- End Of File: ./cs/threads/program.cs

```

Не слишком точные измерения показали, что операции что статические методы класса Interlocked работают существенно быстрее остальных трех способов и метод при помощи класса Monitor работает чуть быстрее, чем метод при помощи ключевого слова lock:

```

--interlocked version, total: 25483276/0
--monitor      version, total: 16512926/0
--lock         version, total: 16084766/0
--lockStatic  version, total: 0/16190122

```

15.14 Пример мертвой блокировки

При помощи класса Monitor можно добиться мертвой блокировки приложения, а инструкция lock в той же ситуации не блокирует работу одного.

```

---- File: ./cs/deadblock/p.cs

---- End Of File: ./cs/deadblock/p.cs

```

15.15 Пример для генерации CSV файла из GPX - файла

Специальный вид XML файлов. Бытовые навигаторы (например, Navitel) в таких файлах сохраняют свои треки. Перевод документации по GPX формату можно посмотреть тут: <https://www.sql.ru/forum/actualutils.aspx?action=gotomsg&tid=1241910&msg=19981792>.

```

---- File: ./cs/gpx2csv/p.cs

using System;
using System.Reflection;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.IO;
using System.Diagnostics;
//using ut;

namespace test
{
    class Program

```

```

{

static void Main(string[] args)
{
    string flNm    = "small.gpx";

    int rc = 1;

    DateTime st = DateTime.Now;
    string s = "csv file is ready!";
    {
        if (File.Exists(flNm)) //P $\Delta$ 0 $\mu$ ,0 „0  $\gg$  ',0 $\mu$ ,0 $\mu$  ,0  $\pm$  ¶ $\mu$   $\mu\mu$  ,0 $\mu$ 
        {

            string p = Path.GetDirectoryName(flNm);
            if (p.Length < 1) p = ".";
            string outFile = p
                + "/" + Path.GetFileNameWithoutExtension(flNm)
                + ".{0}.csv";

            StreamWriter sw = null;
            Console.Error.WriteLine(
                "output file name if '{0}'!", outFile);
            XmlDocument xml = new XmlDocument();
xml.Load(flNm);
            XmlNode gpx = xml.DocumentElement;
            XmlNodeList lst = gpx.ChildNodes;
            XmlNodeList trkLst = null;
            Console.Error.WriteLine(
                "metadata '{0}'/{1}!", lst[0].Name, lst[0].InnerText);
            int i = 0;
            string tmp;
            for ( i = 1; i < lst.Count; i++) {
                XmlNode trk = lst[i];
                trkLst = trk.ChildNodes;
                if (trkLst.Count > 0 ){
                    string ff;
                    if (trkLst[0].Name == "name"){
                        // PSP°C $\ddagger$ P°P»C $\acute{f}$ C $\grave{u}$  PSPsPIC<P $\ddagger$ 
                        Console.Error.WriteLine(
                            "next track/file: {0}'/{1}'", trkLst[0].InnerText
                                , string.Format(outFile,trkLst[0].InnerText.Replace(" ", "_")));
                        if (sw !=null)
                            sw.Dispose();
                        sw = new StreamWriter(string.Format(outFile,trkLst[0].InnerText.Replace(" "
                            if(sw !=null) {
                                Console.Error.WriteLine(
                                    "stream writer is opened");
                                sw.WriteLine(
                                    "#stream writer is opened");
                            }
                        }
                    }
                }
            }
            if (trkLst.Count > 1 && trkLst[1].Name == "trkseg"){ // PSPsPIC<P $\ddagger$  C $\acute{f}$ P $\mu$ i $j\mu$ ,0 ,0
                XmlNodeList pnts = trkLst[1].ChildNodes;

```

```

if (pnts.Count > 0) {
    XmlNode pnt1 = pnts[0];
    if (pnt1.Name == "trkpt") {
        if (sw!=null)
            sw.Write("#    ");
        else
            Console.Write("#    ");

        foreach( XmlAttribute a in pnt1.Attributes){
            if (sw!=null)
                sw.Write("{0}    ", a.Name);
            else
                Console.Write("{0}    ", a.Name);
        }
        foreach ( XmlNode pp in pnt1.ChildNodes) {
            if (sw!=null)
                sw.Write("{0}    ", pp.Name);
            else
                Console.Write("{0}    ", pp.Name);
        }
    }
    foreach( XmlNode pnt in pnts){
        if (pnt.Name == "trkpt") {
            if (sw!=null)
                sw.WriteLine("");
            else
                Console.WriteLine("");
            foreach( XmlAttribute a in pnt.Attributes){
                if (sw!=null)
                    sw.Write("{0} ", a.InnerText);
                else
                    Console.Write("{0} ", a.InnerText);
            }
            foreach ( XmlNode pp in pnt.ChildNodes) {
                if (pp.Name == "time") {
                    tmp = pp.InnerText.Replace("T", " ").Replace("Z", " ");
                    Console.Error.WriteLine(
                        "time: {0}", tmp);
                }
                else
                    tmp = pp.InnerText;
                if (sw!=null)
                    sw.Write("{0} ", tmp);
                else
                    Console.Write("{0} ", tmp);
            }
        }
    }
    Console.WriteLine("\n");
}
if (sw !=null)
    sw.Dispose();
}

```



```

        if (trkLst == null || trkLst.Count < 1){
            Console.Error.WriteLine(
                "no any track");
        }
    }
    else {
        Console.Error.WriteLine(
            "no such file: '{0}'!", (string)flNm);
    }
    Console.WriteLine(s);
    DateTime fn = DateTime.Now;
    Console.Error.WriteLine( "time of work is {0} secs;"
        , (fn - st).TotalSeconds);
}
}
}

---- End Of File:./cs/gpx2csv/p.cs

```

15.16 Скачивание карт Яндексa и ОпенСтритМап

В примере демонстрируется применение функций для прямого и обратного преобразования градусов Мировой Геодезической Системы Координат WGS-84 в пиксели некоторой условной карты. Функции были написаны по примеру <http://doro.poltava.ua/articles/tcl/geo2tiles.html> и использованы в нескольких проектах.

Приложение строит запросы для скачивания тайла (кусочка большой карты 18 масштаба) содержащего заданную точку:

```

---- File:./cs/map/p.cs

#pragma warning disable 642

using System;
using System.Globalization;
using System.Reflection;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using System.Diagnostics;

namespace geo
{
    static class Program
    {
        static double[] resolution;
        static Program (){
            resolution = new double[19];

```

```

        resolution[0]= 156543.03392804097 ;
        resolution[1]= 78271.51696402048 ;
        resolution[2]= 39135.75848201024 ;
        resolution[3]= 19567.87924100512 ;
        resolution[4]= 9783.93962050256 ;
        resolution[5]= 4891.96981025128 ;
        resolution[6]= 2445.98490512564 ;
        resolution[7]= 1222.99245256282 ;
        resolution[8]= 611.49622628141 ;
        resolution[9]= 305.748113140705 ;
        resolution[10]= 152.8740565703525 ;
        resolution[11]= 76.43702828517625 ;
        resolution[12]= 38.21851414258813 ;
        resolution[13]= 19.109257071294063 ;
        resolution[14]= 9.554628535647032 ;
        resolution[15]= 4.777314267823516 ;
        resolution[16]= 2.388657133911758 ;
        resolution[17]= 1.194328566955879 ;
        resolution[18]= 0.5971642834779395 ;
    }
    static int getZoom (int meters, ref int divisor){ //сколько тайлов в заданной длине
        if (divisor < 2) divisor = 2;
        else if (divisor > 20) divisor = 20;
        int zm = 18;
        for (; zm > 0; zm--) {
            if (resolution[zm]*256.0 > meters/divisor ) { // длина стороны тайла в метрах на местн
                if (zm < 18)
                    zm++;
                return zm;
            }
        }
        return zm;
    }

    [STAThread]
    static void Main(string[] args)
    {
        //Param
        DateTime st = DateTime.Now;
        // int ii = 1;
        // string title = "";
        {
            uint tilesX;
            uint tilesY;
            uint picX ;
            uint picY ;
            double flatX;
            double flatY;
            double lat1 = 49.437328 ;
            double lon1 = 32.060387 ;
            int zoom = 18; // пересчитать в масштаб

            Console.Error.WriteLine( "latitude/longitude/size {0}/{1}/{2}"

```

```

        , coor2str(lat1), coor2str(lon1), zoom);

    {
        yandexLonLat2XY (out tilesX, out tilesY, out picX, out picY, out flatX, out flatY,
            lat1, lon1, (short)zoom);
        Console.WriteLine(
            "wget \"http://vec01.maps.yandex.net/tiles?l=map&v=2.28.0&y={2}&x={1}&z={0}&lang=ru\"
            , zoom , tilesX, tilesY);
    }
    {
        LonLat2XY (out tilesX, out tilesY, out picX, out picY, out flatX, out flatY,
            lat1, lon1, (short)zoom);
        Console.Error.WriteLine(
            "numbers of tile x/y: {0}/{1}; pixel coordinates x/y: {2}/{3}; zoom: {4}"
            , tilesX, tilesY, picX, picY, zoom);
        Console.Error.WriteLine(
            "flat coor x/y: {0}/{1}; "
            , coor2str(flatX), coor2str(flatY));
        Console.WriteLine(
            "wget http://a.tile.openstreetmap.org/{0}/{1}/{2}.png -O osm.{0}.{1}.{2}.png", zoom
        )
    }
}

static void yandexLonLat2XY (out uint tilesX
    , out uint tilesY
    , out uint picX
    , out uint picY
    , out double flatX
    , out double flatY
    , double lat
    , double lon
    , short z
    ){
    double pi = 3.1415926535897932;
    double latrad = (lat*pi)/180.0;
    double lonrad = (lon*pi)/180.0;

    double a = 6378137.0;
    double k = 0.0818191908426;

/*
    tan($pi / 4 + $latrad / 2)
    /
    (tan ($pi / 4
        + asin($k * sin($latrad))
        / 2
    )
    ) ** $k
*/

    double f =
        Math.Tan( pi/4.0 + latrad/2.0)
        /
        Math.Pow( Math.Tan(pi/4.0

```

```

        + Math.Asin(k * Math.Sin(latrad) )
        /2.0
    ) , k);

    flatX = (20037508.342789 + a *lonrad)      *53.5865938 / Math.Pow(2, 23-z) ;
    flatY = (20037508.342789 - a *Math.Log(f))*53.5865938 / Math.Pow(2, 23-z) ;
    Console.Error.WriteLine("flatY {0}/{1}",flatY, coor2str(flatY));

    tilesX = (uint)(flatX/256);
    tilesY = (uint)(flatY/256);
    picX   = ((uint)flatX)%256;
    picY   = ((uint)flatY)%256;
}
static void LonLat2XY (out uint tilesX
                      , out uint tilesY
                      , out uint picX
                      , out uint picY
                      , out double flatX
                      , out double flatY
                      , double lat
                      , double lon
                      , short zoom
                      ){
    double pi = 3.1415926535897932;
    double latrad = (lat*pi)/180.0;
    Console.Error.WriteLine("lat radian/input {0}/{1}",latrad, lat);
    double lonrad = (lon*pi)/180.0;

    // double a = 6378137.0;
    // double k = 0.0818191908426;
    // double bm0 = 256 * Math.Pow( 2.0, (double)(zoom/2));
    double bm0 = 256 * Math.Pow( 2.0, (double)zoom)/2.0;
    Console.Error.WriteLine("bm0/zoom {0}/{1}",bm0, zoom);

    flatX = (bm0 * (1+ lonrad/pi));
    flatY = (bm0 * (1- 0.5 * Math.Log(
        (1+ Math.Sin(latrad))
        /
        (1-Math.Sin(latrad))
        )/pi));
    Console.Error.WriteLine("flatY {0}/{1}",flatY, coor2str(flatY));

    tilesX = (uint)(flatX)/256;
    tilesY = (uint)(flatY)/256;
    picX   = (uint)(flatX)%256;
    picY   = (uint)(flatY)%256;
}
static string coor2str(double coor){
    return string.Format("{0:##.#####}", coor);
}
}

```

```
---- End Of File:./cs/map/p.cs
```

После выполнения приложения должно получиться нечто вроде:

```
---- File:./cs/map/wget.cmd
```

```
wget "http://vec01.maps.yandex.net/tiles?l=map&v=2.28.0&y=89750&x=154417&z=18&lang=ru-RU" -O yandex.  
wget http://a.tile.openstreetmap.org/18/154417/89538.png -O osm.18.154417.89538.png
```

```
---- End Of File:./cs/map/wget.cmd
```

Строку http-запроса можно скопировать в браузер либо использовать утилиту wget.

Далее, приводятся тексты всех четырех функций:

```
---- File:./cs/map/1.txt
```

```
/// <summary>  
/// Переводит из координат точки на тайле Yandex-а в реальные координаты  
/// </summary>  
/// <param name="tilesX">Колонка тайла</param>  
/// <param name="tilesY">Строка тайла</param>  
/// <param name="picX">X координата точки на тайле</param>  
/// <param name="picY">Y координата точки на тайле</param>  
/// <param name="z">Зум карты</param>  
/// <param name="latitude"></param>
```

```

/// <param name="longitude"></param>

protected void yandexXY2LonLat ( double tilesX
                                , double tilesY
                                , double picX
                                , double picY
                                , int z
                                , out double latitude
                                , out double longitude)
{
    // double e = 2.7182818284590452;
    double pi = 3.1415926535897932;

    double a = 6378137.0;
    double c1 = 0.00335655146887969;
    double c2 = 0.00000657187271079536;
    double c3 = 0.00000001764564338702;
    double c4 = 0.00000000005328478445;

    double flatX = tilesX * 256 + picX;
    double flatY = tilesY * 256 + picY;

    double mercX = flatX * Math.Pow(2, (23 - z)) / 53.5865938 - 20037508.342789;
    double mercY = 20037508.342789 - (flatY * Math.Pow(2, (23 - z))) / 53.5865938;
    double g = pi / 2 - 2 * Math.Atan(1.0 / Math.Exp(mercY / a));
    double f = g + c1 * Math.Sin(2 * g) + c2 * Math.Sin(4 * g) + c3 * Math.Sin(6 * g) + c4 *

    latitude = f * 180.0 / pi;
    longitude = mercX / a * 180.0 / pi;
}
/// <summary>
/// Преобразование из реальных координат в координаты тайла для Yandex-a
/// </summary>
/// <param name="tilesX">Колонка тайла</param>
/// <param name="tilesY">Строка тайла</param>
/// <param name="picX">X координата точки на тайле</param>
/// <param name="picY">Y координата точки на тайле</param>
/// <param name="flatX">X координата точки в плоскости проекции Земли</param>
/// <param name="flatY">Y координата точки в плоскости проекции Земли</param>
/// <param name="lat">Широта</param>
/// <param name="lon">Долгота</param>
/// <param name="z">Зум карты</param>
protected void yandexLonLat2XY(out int tilesX
                                , out int tilesY
                                , out int picX
                                , out int picY
                                , out double flatX
                                , out double flatY
                                , double lat
                                , double lon
                                , int z
                                )
{
    double pi = 3.1415926535897932;

```

```

double latrad = (lat * pi) / 180.0;
double lonrad = (lon * pi) / 180.0;

double a = 6378137.0;
double k = 0.0818191908426;
double f = Math.Tan(pi / 4.0 + latrad / 2.0) / Math.Pow(Math.Tan(pi / 4.0 + Math.Asin(k

flatX = (20037508.342789 + a * lonrad) * 53.5865938 / Math.Pow(2, 23 - z);
flatY = (20037508.342789 - a * Math.Log(f)) * 53.5865938 / Math.Pow(2, 23 - z);

tilesX = (int)(flatX / 256);
tilesY = (int)(flatY / 256);
picX = ((int)flatX) % 256;
picY = ((int)flatY) % 256;
}

/// <summary>
/// Преобразование из реальных координат в координаты тайла для OpenStreetMap-a
/// </summary>
/// <param name="tilesX">Колонка тайла</param>
/// <param name="tilesY">Строка тайла</param>
/// <param name="picX">X координата точки на тайле</param>
/// <param name="picY">Y координата точки на тайле</param>
/// <param name="flatX">X координата точки в плоскости проекции Земли</param>
/// <param name="flatY">Y координата точки в плоскости проекции Земли</param>
/// <param name="lat">Широта</param>
/// <param name="lon">Долгота</param>
/// <param name="z">Зум карты</param>
/// <summary>
void osmLonLat2XY(out int tilesX
    , out int tilesY
    , out int picX
    , out int picY
    , out double flatX
    , out double flatY
    , double lat
    , double lon
    , int zoom
)
{
    double pi = 3.1415926535897932;
    double latrad = (lat * pi) / 180.0;
    double lonrad = (lon * pi) / 180.0;
    double bm0 = 256 * Math.Pow(2.0, (double)zoom) / 2.0;

    flatX = (bm0 * (1 + lonrad / pi));
    flatY = (bm0 * (1 - 0.5 * Math.Log((1 + Math.Sin(latrad)) / (1 - Math.Sin(latrad)))) / pi

    tilesX = (int)(flatX) / 256;
    tilesY = (int)(flatY) / 256;
    picX = (int)(flatX) % 256;
    picY = (int)(flatY) % 256;
}

```

```

/// <summary>
/// Переводит из координат точки на тайле OpenStreetMap-а в реальные координаты
/// </summary>
/// <param name="tilesX">Колонка тайла</param>
/// <param name="tilesY">Строка тайла</param>
/// <param name="picX">X координата точки на тайле</param>
/// <param name="picY">Y координата точки на тайле</param>
/// <param name="z">Зум карты</param>
/// <param name="latitude"></param>
/// <param name="longitude"></param>
/// <summary>
void osmXY2LonLat(double tilesX, double tilesY, double picX, double picY, int z,
                  out double latitude,
                  out double longitude)
{
    double e = 2.7182818284590452;
    double pi = 3.1415926535897932;
    double bm0 = 256 * Math.Pow(2, z) / 2.0;

    double flatX = tilesX * 256 + picX;
    double flatY = tilesY * 256 + picY;
    double lonrad = pi * (flatX - bm0) / bm0;
    double c1 = 2 * pi * (bm0 - flatY) / bm0;
    double latrad = Math.Asin((Math.Pow(e, c1) - 1) / (Math.Pow(e, c1) + 1));

    latitude = 180 * latrad / pi;
    longitude = 180 * lonrad / pi;
}

```

---- End Of File:./cs/map/1.txt

15.17 Утилита для генерации ресурсов

Приложение является иллюстрацией использования класса ResXResourceWriter - писателя ресурсов в XML файл. Кроме того, в приложении используется библиотека обработки командной строки (см. [19]) и библиотека журналирования работы приложения (см. [20]).

Исходный тест утилиты:

---- File:./cs/resxgen/ut/program.cs

```

#pragma warning disable 642

using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Globalization;
using System.Resources;
using System.Drawing.Imaging;

```



```

using System.Drawing;

using Args;
using Logger;

namespace test
{
    class Program
    {
        static public ArgFlg hlpF ;
        static public ArgFlg dbgF ;
        static public ArgFlg vF ;
        static public ArgIntMM logLvl ;
        static public ArgStr flNm ;
        static public ArgStr resx ;

        static Program (){
            hlpF = new ArgFlg(false, "?", "help", "to see this help");
            vF = new ArgFlg(false, "v", "verbose", "additional info");
            dbgF = new ArgFlg(false, "d", "debug", "debug mode");
            logLvl = new ArgIntMM(1, "l", "log", "log level", "LLL");
            logLvl.setMin(1);
            logLvl.setMax(8);
            flNm = new ArgStr("none", "f", "file", "data file, if not set - list (resourceNm, file");
            resx = new ArgStr(".r.resx", "r", "resx", "resource file", "RFNM");
        }

        static public void usage(){
            Args.Arg.mkVHelp("to make resx file from list of files", "resourceNm", vF

                ,hlpF
                ,dbgF
                ,vF
                ,logLvl
                ,flNm
                ,resx
            );
            Environment.Exit(1);
        }

        /*
        static Program(){
            var format = new System.Globalization.NumberFormatInfo();
            format.NumberDecimalSeparator = ".";
        } */

        [STAThread]
        static void Main(string[] args)
        {
            string rNm = "";

            for (int i = 0; i<args.Length; i++){
                if (hlpF.check(ref i, args))

```

```

        usage();
    else if (dbgF.check(ref i, args))
    ;
    else if (vF.check(ref i, args))
    ;
    else if (logLvl.check(ref i, args))
    ;
    else if (flNm.check(ref i, args))
    ;
    else if (resx.check(ref i, args))
    ;
    else
        rNm = args[i];
    }
    if (rNm == "" && flNm != "none")
        usage();
    DateTime st = DateTime.Now;
    using (LOGGER Logger = new LOGGER(LOGGER.uitoLvl(logLvl))){
if (vF)
    Logger.cnsllvl = IMPORTANCELEVEL.Spam;
string extension;
Logger.WriteLine(IMPORTANCELEVEL.Stats
    , " resource name/file resx name: '{0}'/'{1}'", rNm, (string)flNm);
    //ResXResourceSet resSet = new ResXResourceSet(resx);
        Image i;
        byte [] bs;
        ResXResourceWriter rw = //new ResXResourceWriter(resSet);
                                new ResXResourceWriter(resx);
Logger.WriteLine(IMPORTANCELEVEL.Stats
    , "ResXResourceWriter is here! ");
    if (flNm != "none"){
        extension = Path.GetExtension(flNm);
        if (extension.ToLower() == ".jpg") {
            i = Image.FromFile(flNm, true);
            rw.AddResource(rNm, i);
        }
        else if (extension.ToLower() == ".png") {
            i = Image.FromFile(flNm, true);
            rw.AddResource(rNm, i);
        }
        else if (extension.ToLower() == ".bmp") {
            i = Image.FromFile(flNm, true);
            rw.AddResource(rNm, i);
        }
        else {
            bs = File.ReadAllBytes(flNm);
            rw.AddResource(rNm, bs);
        }
        Logger.WriteLine(IMPORTANCELEVEL.Stats
            , "Image from file is here! ");
    }
else {
        int lineno=1;
        for (string str = Console.ReadLine());

```

```

        str != null;
        str = Console.ReadLine(), lineno++) {
    Logger.WriteLine(IMPORTANCELEVEL.Stats
        , "line no {0}:{1}'", lineno, str);
    string[] values = str.Split(new char[] { ' ', '\t' }
        , StringSplitOptions.RemoveEmptyEntries);
    if (values.Length > 1) {
        try{
            extension = Path.GetExtension(values[1]);
            if (extension.ToLower() == ".jpg") {
                i = Image.FromFile(values[1], true);
                rw.AddResource(values[0], i);
            }
            else if (extension.ToLower() == ".png") {
                i = Image.FromFile(values[1], true);
                rw.AddResource(values[0], i);
            }
            else if (extension.ToLower() == ".bmp") {
                i = Image.FromFile(values[1], true);
                rw.AddResource(values[0], i);
            }
            else {
                bs = File.ReadAllBytes(values[1]);
                rw.AddResource(values[0], bs);
            }
        }
        catch{
            Logger.WriteLine(IMPORTANCELEVEL.Error
                , "skipped file in line no {0}:{1}', length:{2}", lineno, str, valu
            }
        }
        else
            Logger.WriteLine(IMPORTANCELEVEL.Error
                , "wrong line no {0}:{1}', length:{2}", lineno, str, values.Length);
    }
}
rw.Close();
DateTime fn = DateTime.Now;
Logger.WriteLine(IMPORTANCELEVEL.Stats, "time of work with file '{1}' is {0} secs"
    , (fn - st).TotalSeconds, (string)flNm);
}
}
}

```

```
---- End Of File:./cs/resxgen/ut/program.cs
```

Список картинок из которых будет строится файл ресурсов:

```
---- File:./cs/resxgen/ut/pics.lst
```

```
pic1          ./pics/pen.png
pic2          ./pics/settings.png
pic3          ./pics/skull.png
pic4          ./pics/3.png
```

```
---- End Of File:./cs/resxgen/ut/pics.lst
```

Файлы с картинками находятся в каталоге ./cs/resXGen/ut/pics.

Пример выполнения утилиты, для построения файла ресурсов:

```
---- File:./cs/resxgen/ut/test.cmd
```

```
resXGen.exe -l 3 -v <pics.lst -r picslst.resx
```

```
---- End Of File:./cs/resxgen/ut/test.cmd
```

Построенный файл eyelst.resx используется в лабораторной [6.1](#) для построения приложения с подмигивающим глазом.

15.18 Примеры файла AssemblyInfo.cs

По умолчанию Visual Studio при создании проекта сама добавляет файл AssemblyInfo.cs к проекту. Его можно найти в каталоге Properties. Другие примеры этих файлов можно найти в проектах [\[19\]](#) и [\[20\]](#).

15.19 Проблема с разделителем дробной части

В различных культурах дробная часть вещественного числа отделяется от целой разными символами. В качестве таковых могут выступать, в частности, символы запятая ',' или точка '.'. Символ задается в региональных настройках компьютера. Доступ к ним из программы можно получить при помощи свойства

```
using System.Threading;
...
Thread.CurrentThread.CurrentCulture
```

Для чтения тех или иных версий вещественных чисел можно использовать следующие статические методы с явно задаваемым разделителем дробной части:

```
public
static double Parse( string val
                    , char dP = '.'          ///< дополнительный резделитель дробной части
                    ){    ///

    string v = val.Replace(dP,
        Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator[0]);
    return double.Parse(v);
}

public
static bool TryParse ( string val
                      , out double number
                      , char dP          ///< дополнительный резделитель дробной части
                      ){    ///

    string v = val.Replace(dP,
        Thread.CurrentThread.CurrentCulture.NumberFormat.NumberDecimalSeparator[0]);
    return double.TryParse(v, out number);
}
```

15.20 Сериализация и десериализация

Сериализация - это процесс преобразования объекта в последовательность (серию байт), десериализация - обратный процесс, восстановление объекта из последовательности байт. Примеры различных видов сериализации (xml, двоичная и SOAP) и десериализации можно увидеть в следующем приложении:

---- File:./cs/serialization/program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization.Formatters.Soap;
using System.Xml.Serialization;

using Args;

namespace csv2xml
{
    enum Formater {Xml, Bin, Soap};
```

```

class Program
{
    static public ArgStr iPath;
    static public ArgStr oPath;
    static public ArgFlg hlpF;
    static public ArgFlg vF;
    static public ArgStr fld;
    static public ArgStr frmt ;
    static public ArgStr sep;
    static public Dictionary<string, string> fldDict;

    static Program()
    {
        hlpF = new ArgFlg(false, "?", "help", "to see this help");
        vF = new ArgFlg(false, "v", "verbose", "additional output");
        iPath = new ArgStr(".", "i", "input", "input xml path", "PATH");
        iPath.show = false;
        oPath = new ArgStr(".", "o", "output", "output xml path", "PATH");
        oPath.show = false;
        sep = new ArgStr(";", "s", "separator", "csv separator", "CHAR");
        fld = new ArgStr("field", "f", "node attribute", "key:value", "LISTDICT");
        frmt = new ArgStr("Xml", "ser", "serialization", "Xml/Bin/Soap: serialization param", "SSS");
    }

    static public void usage()
    {
        Args.Arg.mkVHelp(
            "to demo serialization "
            , "-i PATH | -o PATH " , vF, vF, hlpF, frmt, iPath, oPath, sep );
        Environment.Exit(1);
    }

    public static void xml2csv(string inPath, string separator, Formater f)
    {
        switch (f){
            case Formater.Xml:    xml2csv (inPath, separator);    break;
            case Formater.Bin:    bin2csv (inPath, separator);    break;
            case Formater.Soap:    soap2csv(inPath, separator);    break;
        }
    }

    public static void xml2csv(string inPath, string separator)
    {
        XmlSerializer formatter = new XmlSerializer(typeof(L));

        using (FileStream fs = new FileStream(inPath , FileMode.Open))
        {
            // PrPIP° PsP±μ,0Δ..,0' >> ...

```

```

        L lst = (L) formatter.Deserialize(fs);
        foreach (Record r in lst )
            Console.Out.WriteLine(r.ToString());
    }
}

public static void soap2csv(string inPath, string separator)
{
    SoapFormatter formatter = new SoapFormatter();

    using (FileStream fs = new FileStream(inPath , FileMode.Open))
    {
        // PrPIP° PsP±μ,0Δ...,0' >>...

        L lst = (L) formatter.Deserialize(fs);
        foreach (Record r in lst )
            Console.Out.WriteLine(r.ToString());
    }
}

public static void bin2csv(string inPath, string separator)
{
    BinaryFormatter formatter = new BinaryFormatter();

    using (FileStream fs = new FileStream(inPath , FileMode.Open))
    {
        // PrPIP° PsP±μ,0Δ...,0' >>...

        L lst = (L) formatter.Deserialize(fs);
        foreach (Record r in lst )
            Console.Out.WriteLine(r.ToString());
    }
}

public static void csv2xml(string outputPath, string separator, Formater f)
{
    using (FileStream fs = new FileStream(outputPath , FileMode.Create))
    {
        L lst = new L();

        string str = Console.In.ReadLine();
        while (str != null)
        {
            string[] values = str.Split(separator[0]);
            uint a = 0;
            double b = 0.1;
            if (uint.TryParse(values[1], out a)) ;
            if (double.TryParse(values[2], out b)) ;
            Record r = new Record(values[0], a, b);
            lst.Add(r);
            str = Console.In.ReadLine();
        }

        switch (f){
            case Formater.Xml:    csv2xml(lst, fs, separator);    break;
            case Formater.Bin:    csv2bin(lst, fs, separator);    break;
        }
    }
}

```

```

        case Formater.Soop:  csv2soap(lst, fs, separator);    break;
    }
}
}

```

```

public static void csv2xml(L lst, FileStream fs, string separator)
{
    XmlSerializer formatter = new XmlSerializer(typeof(L));
    formatter.Serialize(fs, lst);
}
public static void csv2bin(L lst, FileStream fs, string separator)
{
    BinaryFormatter formatter = new BinaryFormatter();
    formatter.Serialize(fs, lst);
}
public static void csv2soap(L lst, FileStream fs, string separator )
{
    SoapFormatter formatter = new SoapFormatter();
    formatter.Serialize(fs, lst);
}

```

```

public static void splitFiled(string field)
{
    if (Object.ReferenceEquals(fldDict, null))
        fldDict = new Dictionary<string, string>();
    string[] val = field.Split(':');
    if (fldDict.ContainsKey(val[0]))
        return;
    fldDict.Add(val[0], val[1]);
}

```

```

static void Main(string[] args)
{
    Formater f;
    for (int i = 0; i < args.Length; i++)
    {
        if (hlpF.check(ref i, args))
            usage();
        else if (iPath.check(ref i, args))
            ;
        else if (oPath.check(ref i, args))
            ;
        else if (sep.check(ref i, args))
            ;
        else if (frmt.check(ref i, args))
            ;
        else if (fld.check(ref i, args))
            ;
    }
}

```



```

{

    [Serializable]
    public class L: List<Record>
    {
    }

    [Serializable]
    public class Record
    {
        public string Word;
        public uint Amount;
        public double xx;
        private int sss;
        private static int count;
        static Record(){
            count = 0;
        }
        public Record(string _Word, uint _Amount, double _xx)
        {
            Word = _Word;
            Amount = _Amount;
            xx = _xx;
            count ++ ;
            sss = count;
        }
        public Record() : this("", 0, 1.1) { }
        public override string ToString(){
            return string.Format("{0};{1};{2};{3}", Word, Amount, xx, sss);
        }
    }
}

```

---- End Of File:./cs/serialization/record.cs

Примеры входных/выходных файлов приложения

---- File:./cs/serialization/1.txt

слово11;222;33,33
слово2;111;999,9

---- End Of File:./cs/serialization/1.txt

---- File:./cs/serialization/1.xml

```

<?xml version="1.0"?>
<ArrayOfRecord xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
  <Record>
    <Word>слово11</Word>
    <Amount>222</Amount>
    <xx>33.33</xx>
  </Record>
  <Record>
    <Word>слово2</Word>
    <Amount>111</Amount>
    <xx>999.9</xx>
  </Record>
</ArrayOfRecord>

```

```
<Word>слово2</Word>
<Amount>111</Amount>
<xx>999.9</xx>
</Record>
</ArrayOfRecord>

---- End Of File:./cs/serialization/1.xml
```

Объекты некоторых типов сериализовать нельзя. См. <https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/feature-details/types-supported-by-the-data-contract>

16 СПИСКИ ЛАБОРАТОРНЫХ РАБОТ ДЛЯ СОКРАЩЕННЫХ КУРСОВ

Раздел содержит списки лабораторных работ для сокращенных курсов и методические рекомендации для их выполнения.

16.1 Методические указания для курса Технологии .NET В Разработке ИС

- см. 2.1 вычисление простых чисел;
- см. 2.3 передача фактических параметров в методы;
- см. 2.4 обработка строчек;
- дополнительная лабораторная по изучению передачи аргументов командной строки;
- лабораторная 2.5 разделяется на две: 1 - выбор двух столбцов целых чисел (метод Split и оформление CSV-файлов), 2 - чтение координат трека (метод Parse и ключевые слова try-catch-finally);
- лабораторная 5.5 - можно просто заполнить DataGridView без использования главного меню приложения.
- лабораторная 6.2 - нарисовать в окне приложения немасштабированную ломаную.

16.1.1 Вычисление простых чисел

Постановку задачи см. в разделе 2.1 , методические рекомендации см. в 10.2 .

16.1.2 Передача фактических параметров в методы

Постановку задачи см. в разделе 2.3 , методические рекомендации см. в 10.4 .

16.1.3 Обработка строчек

Постановку задачи см. в разделе 2.4 , методические рекомендации см. в 10.5 .

16.1.4 Передача аргументов командной строки в приложение

Аргументы командной строки - это цепочки символов, разделенные пробелами. В методических рекомендациях [10.5](#) они уже начали обсуждаться. Осталось дополнить, что аргумент командной строки, который начинается на символ тире ('-') или наклонная черта ('/') будет называться ключ. Ключ может иметь параметры или может их не иметь. Заглавные и прописные буквы в аргументах командной строки обычно не различаются. Например, ключ '-?' или '/?' параметра не имеет и традиционно приводит к выдаче приложением краткой инструкции об его использовании и немедленно завершает его работу.

Под обработкой аргументов обычно подразумевается цикл, который просматривает массив args (см. ниже) слева направо и, в зависимости от наличия ключей и их параметров, присваивает некоторые значения переменным различного типа и выдает экран подсказки.

```
static void Main(string[] args)    {
    //код здесь
}
```

Никаких других действий в этом цикле обычно не требуется выполнять. Далее, эти переменные используются для работы приложения. Например, в примере ниже, это следующие переменные:

```
int x=0;           // координаты левого верхнего
int y=0;           // угла управляющего элемента окна
string w="text";   // текст на управляющем элементе окна
int f = 0;         // признак, какой управляющий элемент показать
```

Цикл, который ищет только ключ помощи ('-?') может выглядеть как то так:

```
for (int i = 0; i < args.Length; i++) {
    if (args[i].Equals("-?")){
        Console.WriteLine("  Spravka. Programma dlya vstavki v formu knopki, nadpisi ili tekstovo
        Console.WriteLine("  app.exe [-?] [-x ***] [-y ***] [-w ***] -k <button/label/textbox>");
        Console.WriteLine("  -?  spravka;");
        Console.WriteLine("  -x  otstup obyekta ot levogo kraya;");
        Console.WriteLine("  -y  otstup obyekta ot verhnego kraya;");
        Console.WriteLine("  -w  text na obyekte ;");
        Console.WriteLine("  -k  vibor obyekta(button,label ili textbox);");
        return 1;           // код работы приложения должен быть отличен от 0
    }
}
```

Тут можно увидеть, что все ключи, кроме ключа помощи имеют параметр. Обработка этого параметра требует дополнительных усилий, что бы избежать исключительных состояний. В примере ниже намеренно такая проверка сделана только для ключа '-v':

```
else if (args[i].ToLower().Equals("-k")) {
    if (i+1 < args.Length) {           /*** проверка существования параметра
```

Ключи с квадратными скобками в подсказке считаются не обязательными и оператор может их не задавать. Ключи без квадратных скобок считаются обязательными, в нашем примере - это ключ '-k'.

16.1.4.1 Законченный пример, написанный только средствами языка

В этом примере не используется никаких дополнительных возможностей, кроме языка C#:

---- File:./cs/labs/mod3lab1/p.cs

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

namespace m3l1{
    class Program    {
        static
        int Main(string[] args)    {
            int x=0;                // координаты левого верхнего
            int y=0;                // угла управляющего элемента окна
            string w="text";        // текст на управляющем элементе окна
            int f = 0;              // признак, какой управляющий элемент показать
            if (args.Length == 0)    {
                Console.Error.WriteLine("Error: Nothing to do!");
                return 1;
            }
            //идем по ключам
            for (int i = 0; i < args.Length; i++) {
                if (args[i].Equals("-?")){
                    Console.WriteLine("    Spravka. Programma dlya vstavki v formu knopki, nadpisi
app.exe [-?] [-x ***] [-y ***] [-w ***] -k <button/label
                    Console.WriteLine("    -? spravka;");
                    Console.WriteLine("    -x otstup obyektu ot levogo kraya;");
                    Console.WriteLine("    -y otstup obyektu ot verhnego kraya;");
                    Console.WriteLine("    -w text na obyekte ");
                    Console.WriteLine("    -k vibor obyektu(button,label ili textbox);");
                    return 1;        // код работы приложения должен быть отличен от 0
                }
                else if (args[i].ToLower().Equals("-x"))
                    x = Convert.ToInt32(args[i+1]);
                else if (args[i].ToLower().Equals("-y"))
                    y = Convert.ToInt32(args[i+1]);
                else if (args[i].ToLower().Equals("-w"))
                    w = args[i+1];
                else if (args[i].ToLower().Equals("-k")) {
                    if (i+1 < args.Length) {        /*** проверка существования параметра
                        if (args[i+1].ToLower().Equals("button"))
                            f = 1;
                        else if (args[i+1].ToLower().Equals("label"))
                            f = 2;
                        else if (args[i+1].ToLower().Equals("textbox"))
                            f = 3;
                        else {
                            Console.Error.WriteLine("Error: wrong parameter!");
                            return 1;
                        }
                    }
                }
            }
        }
    }
}
```

```

    }
    else {
        Console.Error.WriteLine("Error: missed parameter!");
        return 1;
    }
}

}

Application.Run(new wnd(x, y, f, w));
return 0;
}

}
}

```

```
---- End Of File:../cs/labs/mod3lab1/p.cs
```

16.1.4.2 Законченный пример 2

Для обработки аргументов командной строки можно пользоваться библиотекой из проекта [19]. Простой пример можно посмотреть тут:

```
---- File:../cs/args/p.cs
```

```
#pragma warning disable 642
// pragma отключает предупреждение компилятора о пустом операторе в if
using System;
using System.IO;
using Args;

class a{
    static
    void Main(string[] args)    {
        ArgFloat    // ключ плавающего типа
        perCent     = new ArgFloatMM(0.05, "p", "percent", "percent for something", "PPP");
        ArgStr       // ключ типа строка
        flNm        = new ArgStr("somefile.dat", "f", "file", "data file", "FLNM");
        ArgFlg       //
        vFlag       = new ArgFlg(false, "v", "verbose", "to show additional information");
        ArgFlg       //
        hFlag       = new ArgFlg(false, "?", "help", "to show usage page");

        for (int i = 0; i<args.Length; i++){

            if (hFlag.check(ref i, args))    {
                Arg.mkVHlp("to test command line arguments" // подсказка по утилите
                    , ""
                    , vFlag
                    , Arg.mkArgs(
                        hFlag
                        ,vFlag
                        ,perCent
                        ,flNm
                    )
                )
            }
        }
    }
}
```

```

        );
        Environment.Exit(1);

    }
    else if (vFlag.check(ref i, args))
        ; // без прагмы тут будет предупреждение о возможной ошибке
    else if (perCent.check(ref i, args))
        ;
    else if (flNm.check(ref i, args))
        ;
    }
    Console.WriteLine("Вы ввели: {0}/{1}/{2}/{3}"
        , (bool) hFlag
        , (bool) vFlag
        , (double) perCent
        , (string) flNm
    );
}
}

---- End Of File: ./cs/args/p.cs

```

Выбор ключа и проверку существования у него параметра выполняет метод `check`, формирование экрана подсказки - метод `mkVHelp` и конструкторы переменных. Кроме того, метод `check` является примером применения полиморфизма, а `mkVHelp` - примером метода с переменным числом параметров. По примеру должно быть ясно, что тип `ArgFlg` используется для обработки ключей без параметра (аналог логического типа), тип `ArgFloat` - для обработки ключей с параметром в виде вещественного числа, тип `ArgStr` - для обработки ключей с параметром в виде строки.

Более сложный пример обработки параметров командной строки можно посмотреть в приложении

```

---- File: ./cs/labs/2xml/p.cs

#pragma warning disable 642
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.IO;
using Args;

namespace csv2xml
{
    partial class p
    {
        static public ArgStr oPath;
        static public ArgFlg hlpF;
        static public ArgChar sep;
        static public ArgStr enc;
    }
}

```

```

static public ArgStr fNm;

static public Dictionary<int, string> fldDict;

static p(){
    hlpF      = new ArgFlg(false, "?", "help", "to see this help");
    oPath      = new ArgStr(".", "o", "output", "output xml path", "PATH");
    oPath.required = true;          // нет квадратных скобок
    sep        = new ArgChar(',', 's', "separator", "csv separator", "CHAR");
    enc        = new ArgStr("866", "e", "encoding", "encoding of standart input (866, 1251, UTF-8)");
    fNm        = new ArgStr("field", "f", "fieldName", "to set name of field(number:name)", "FIELD");
    fNm.show = false;
    fldDict = new Dictionary<int, string>();
}

static public void usage(){
    Args.Arg.mkVHelp(
        "to convert some csv-file with specified encoding to xml-file "
        , "[ -f FLDNM1 [ -f FLDM2 ] ... ]" , false
        , hlpF, oPath, fNm, enc, sep);
    Environment.Exit(1);
}

static
void Main(string[] args) {
    bool vFlag=false;
    for (int i = 0; i < args.Length; i++) {
        if (hlpF.check(ref i, args))
            usage();
        else if (oPath.check(ref i, args))
            ;
        else if (sep.check(ref i, args))
            ;
        else if (enc.check(ref i, args))
            ;
        else if (fNm.check(ref i, args))
            addNm(fNm, fldDict);
    }
    string enco = "cp866";
    switch (((string)enc).ToLower()){
        case "866": enco = "cp866"; break;
        case "1251": enco = "windows-1251"; break;
        case "utf-8": enco = "utf-8"; break;
    }
}

Console.InputEncoding = Encoding.GetEncoding( enco );
if (vFlag) {
    string []flds = getRec("");
    if (flds == null)
        Console.Error.WriteLine("isNull");
    else
        Console.Error.WriteLine("length {0} ", flds.Length);
}
csv2xml((char)sep);
}

```



```

static
string [] getRec (string line, char sep= ' '){
    string [] flds=null;
    char [] seps;
    if (line != null) {
        // удалить комментарий, лидирующие и завершающие пробелы
        int lastP = line.IndexOf('#');
        if (lastP >= 0) // удалили комментарий
            line = line.Substring(0, lastP);
        line = line.Trim(); // отсеки пробелы

        if (sep == ' ') { // в случае пробела добавить
            // табуляцию
            seps = new char[2];
            seps[0] = ' ';
            seps[1] = '\t';
            flds = line.Split
                (seps, StringSplitOptions.RemoveEmptyEntries);
        }
        else { // каждый символ разделителя создает отдельное поле
            seps = new char[1];
            seps[0] = sep;
            flds = line.Split (seps);
        }
    }
    return flds;
}

}

---- End Of File:./cs/labs/2xml/p.cs

```

Также, в разделе [10.6](#) тоже приводятся объяснения обработки командной строки при помощи проекта [\[19\]](#).

16.1.5 Выбор двух столбцов CSV файла

Сначала выполняется упрощенная версия лабораторной [2.5](#), из CSV - файла извлекается лексемы в заданных столбцах. При этом не требуется преобразование строчек в численный тип и не обязательно использование исключительных состояний. Выяснять существует ли заданный столбец в текущей строки можно явным сравнением. Для выполнения лаботарной достаточно рекомендаций из раздела [10.6](#) вплоть до примера функции getRec.

Как совсем простая версия выполнения - можно просто взять пример метода getPoints для чтения первых двух столбцов целых чисел:

```

public Point [] // чтение массива точек
getPoints(TextReader tr
, bool vFlag=false // выводить подробности об ошибках
)
{

```

```

List <Point> ps = new List <Point> ();
int errs=0;
int lineNo=0;
int a, b;
string r;
string[] sep = { " ", "\t" }; // separators
for ( ;(r = tr.ReadLine())!=null; lineNo++){
    string[] flds = r.Split(sep, StringSplitOptions.RemoveEmptyEntries);
    if (flds.Length >= 2)
    {
        try {
            a = int.Parse(flds[0]);
            b = int.Parse(flds[1]);
            ps.Add (new Point(a, b));
        }
        catch (Exception ex) {
            errs++;
            if (vFlag)
                Console.Error.WriteLine (
                    "lineNo/r/error: {0}/{1}/{2}", lineNo, r, ex.Message);
        }
    }
    else if (flds.Length == 1) {
        errs++;
        if (vFlag)
            Console.Error.WriteLine (
                "lineno/r/error: {0}/{1}/{2}", lineNo, r, "not enough data");
    }
}
if (errs>0)
    Console.Error.WriteLine ("There was {0} errors", errs);
return this.ps;
}

```

При этом, данный метод не должен выполнять разбор CSV - файлов, содержащих ошибки набора данных.

16.1.6 Чтение координат трекера

При выполнении лабораторной требуется использование блоков try - catch - finally. Методические рекомендации смотри в заключительной части раздел [10.6](#) (после метода getRec).

16.1.7 Представление табличных данных

Методические указания см. в разделе [12.5.1](#)

16.1.7.1 Законченный пример

Класс для представления табличных данных без управляющих элементов:

---- File:./cs/labs/datagrid/wnd.cs

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

namespace dtGrd
{
    public class wnd : Form    {
        string fileName;
        protected
        DataGridView dgv = new DataGridView();

        public wnd(string fnm)    {
            fileName = fnm;
            DataGridViewColumn clm = new DataGridViewColumn();
            clm.Name = "Col 1";

            dgv.ReadOnly = true;
            dgv.Dock = DockStyle.Fill;
            dgv.AllowUserToAddRows = false;
            dgv.BackgroundColor = Color.White;
            dgv.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
            Load += fLoad;
            Controls.Add(dgv);
        }

        void fLoad(object sender, EventArgs a)    {
            Console.WriteLine("Load");
            ReadFile(fileName);
        }

        public void ReadFile(string path) {
            string str;
            string[] fields;
            bool firsttime = true;
            using (StreamReader sr = new StreamReader(path))    {
                while ((str = sr.ReadLine()) != null)    {
                    //пропустить пеметки
                    fields = str.Split(';');
                    if (fields.Length <= 0)
                        continue;
                    if (firsttime){
                        dgv.ColumnCount = fields.Length;
                        for (int i = 0; i < fields.Length; i++) {
                            dgv.Columns[i].Name = String.Format("Pole{0}", i+1);
                        }
                        firsttime = false;
                    }
                    dgv.Rows.Add(fields);
                }
            }
        }
    }
}

```

```
}
```

```
---- End Of File:./cs/labs/datagrid/wnd.cs
```

Пример использования

```
---- File:./cs/labs/datagrid/program.cs
```

```
//
#define WND
using System;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Reflection;

namespace dtGrd{
    class Program    {
        [STAThread]
        static void Main(string[] input)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
            openFileDialog1.FilterIndex = 1;
            openFileDialog1.RestoreDirectory = true;
            // C,C#C, PIC<C#C#PSC#Pμ,0>, 0 <>> μ,0' >> μμ
            openFileDialog1.InitialDirectory
                = Path.GetDirectoryName(
                    Assembly.GetExecutingAssembly().Location
                );
            if (openFileDialog1.ShowDialog() == DialogResult.OK)    {
                string fnm = openFileDialog1.FileName;

#if WND
                Form y = new wnd(fnm);
#else
                Form y = new wnd2(fnm);
#endif
                Application.Run(y);
            }
        }
    }
}
```

```
---- End Of File:./cs/labs/datagrid/program.cs
```

16.1.8 Рисование немасштабированной ломаной

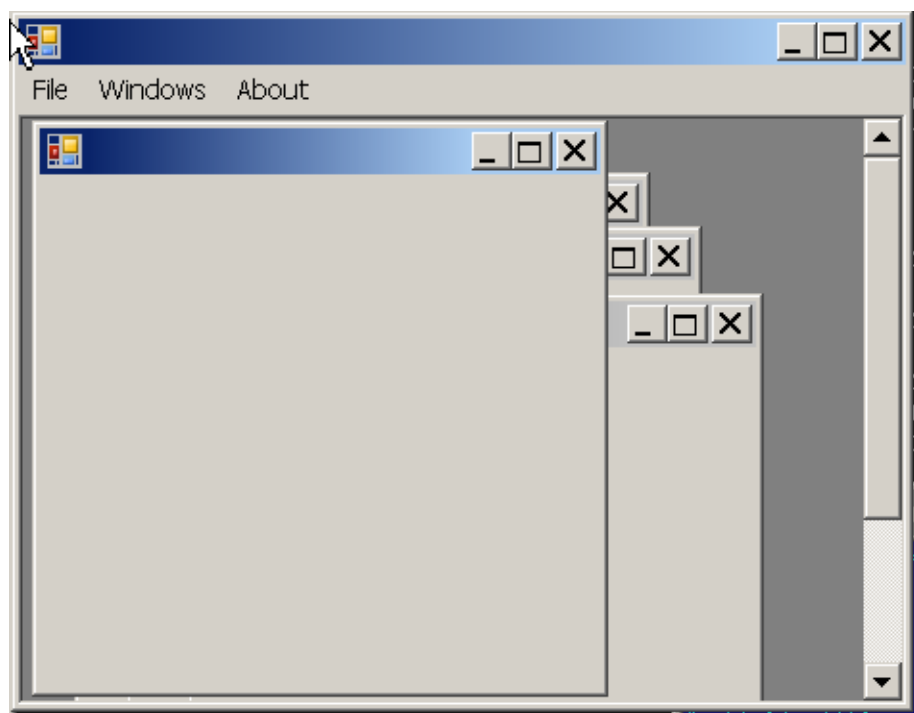
Раздел [13.2](#) содержит указания для рисования немасштабированной ломаной.

16.2 Методические указания для курса Проектирование Управляющих, Информационных и Интеллектуальных систем

- лабораторная 3.4 - конвертация текстовых файлов с записями из целых чисел в двоичные и наоборот;
- лабораторная 3.3 - обработка события FormClosing;
- лабораторная 5.2 - создание диалогового окна OkCancel;
- лабораторная 5.3 - создание диалогового окна для ввода данных;
- лабораторная 5.4 - меню в главном окне приложения.
- лабораторная 5.5 - представление табличных данных.
- лабораторная работа - экспорт управляющего элемента DataGridView в SCV -файл (нужно использовать навыки из лабораторной 3.4) , более сложная версия - экспорт в XML файл, как в лабораторной 3.5 .

Задание:

- Доработать делегат- обработчик для события Click на панели инструментов, требуется обрабатывать нажатия на кнопку Экспорт.
- Для экспорта файла использовать стандартное диалоговое окно SaveFileDialog, поиск файлов начинать с каталога из которого было запущено приложение (а не с диска c:).
- Использовать не две группы файлов, как раньше (файлы с расширением txt и с любым расширением), а четыре: txt, csv, xml, все.
- Файлы читать и писать с кодировкой страницы 1251 (узкая кодировка Windows)
- В начальной версии лабораторной - выполнять экспорт в формате CSV, в качестве разделителей использовать символ как в лабораторной 5.5 , использовать виртуальную функцию - заглушку для вывода в xml - формате. В усложненной версии лабораторной - выполнять экспорт в формате XML. Для уточнения расширения использовать метод Path.GetExtension.
- лабораторная 5.7 - открытие нескольких дочерних окон;



Многодокументный интерфейс

- лабораторная 5.6 - окно редактирования таблиц;
- лабораторная 6.3 - масштабирование ломаной.

Замечание

После выполнения лабораторной 'Создание диалогового окна для ввода данных', в дальнейших работах для целей ввода данных вместо разработанного класса `inWnd` можно использовать класс `OkCancelDlg` из проекта [19, `OkCancelDlg`]. В этом проекте вместо простого класса `fld` (поддерживает только тип строки), используемого для передачи параметров в `inWnd`, для передачи параметров в `OkCancelDlg` используется несколько различных типов: `ArgFlg` - логический тип, `ArgStr` - строки, `ArgChar` - символы, `ArgFloat` - плавающие, `ArgFloatMM`, `ArgInt`, `ArgIntMM`.

16.2.1 Конвертация текстовых файлов с записями из целых чисел в двоичные и наоборот

16.2.1.1 Законченные примеры конвертации файлов

Из текстового файла в двоичный:

```
---- File:./cs/labs/txt2bin/program.cs
```

```

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            using (BinaryWriter bw = new BinaryWriter(File.Open("1.bin", FileMode.Create)))
            {
                Program x = new Program();
                x.str2bin ( Console.In, bw);
            }
        }

        void str2bin(TextReader tr, BinaryWriter bw)
        {
            int a;
            int b;
            string r;
            string[] sep = { " ", "\t" }; // separators

            while ((r = tr.ReadLine())!=null){
                string[] flds = r.Split(sep, StringSplitOptions.RemoveEmptyEntries);
                if (flds.Length >= 2)
                {
                    a = int.Parse(flds[0]);
                    b = int.Parse(flds[1]);
                    bw.Write(a);
                    bw.Write(b);
                }
            }
        }
    }
}

---- End Of File:./cs/labs/txt2bin/program.cs

```

Из двоичного в текстовый:

```

---- File:./cs/labs/bin2txt/program.cs

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;
using System.IO;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            using (BinaryReader br = new BinaryReader(File.Open("1.bin", FileMode.Open)))
            {
                bin2str(br, Console.Out);
            }
        }
        static void bin2str(BinaryReader br, TextWriter tw)
        {
            int a;
            int b;
            string r;
            string[] sep = { " ", "\t" }; // separators

            while (br.PeekChar() != -1)
            {
                a = br.ReadInt32();
                b = br.ReadInt32();
                tw.WriteLine("{0} {1}", a, b);
            }
        }
    }
}

---- End Of File: ./cs/labs/bin2txt/program.cs

```

16.2.2 Обработка события FormClosing

Для выполнения лабораторной работы необходимо выполнить следующие действия:

- Подключить к проекту динамически подгружаемую библиотеку System.Windows.Forms.dll. При компиляции проекта из командной строки, команду

```
csc /out:a.exe /e:winexe *.cs
```

заменить на

```
csc /out:a.exe /e:winexe /r:System.Windows.Forms.dll *.cs
```

- Именованную область видимости System.Windows.Forms добавить к просмотру по умолчанию. В код добавить строчку


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

- Перед методом Main добавить атрибут статического однонитевого приложения:

```
[STAThread]
static void Main(string[] args)
```

- Создать класс для демонстрации будущего окна.

```
class wnd: Form {
    public wnd (string str) { Text = str;}
}
```

- Создать два объекта нового класса и тремя различными способами (дочернего окна, демонстрация модального окна, главного окна приложения) показать созданные окна.

```
wnd a = new wnd("первое окно");
wnd b = new wnd("второе окно");
wnd c = new wnd("третье окно");
b.Show();
a.ShowDialog();
Application.Run(c);
```

- В класс wnd добавить делегат-обработчик события FormClosing:

```
void fc (object sender, FormClosingEventArgs a ){
    Console.WriteLine("событие закрытие окна тут. причина закрытия: {0}",
        a.CloseReason
    );
}
```

- В конструкторе класса wnd делегат-обработчик добавить к событию FormClosing:

```
public wnd (string str) {
    Text = str;
    this.FormClosing += fc;
}
```

- В случае если причина закрытия была действия пользователя приложения (а не, скажем, завершение работы всей операционной системы) выдать запрос на подтверждение и, возможно, передумать выходить из приложения:

```
void fc (object sender, FormClosingEventArgs a ){
    Console.WriteLine("событие закрытие окна тут. причина закрытия: {0}",
        a.CloseReason
    );
    if (e.CloseReason == CloseReason.UserClosing)
    {
        if (MessageBox.Show("Вы уверены, что хотите выйти?"
```

```

        , "My application"
        , MessageBoxButtons.YesNo) == DialogResult.No)
    {
        e.Cancel = true;
    }
}
}

```

16.2.2.1 Законченный пример

---- File:./cs/labs/showwnd/program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ConsoleApplication4
{
    class Program
    {
        [STAThread]

        static void Main(string[] args)
        {
            WND x = new WND("sedg");
            x.ShowDialog(); //Показываем дьялоговое окно. Программа ждет закрытия окна
            Console.WriteLine("Показывали первое модальное окно");
            x.Show(); //Показали дочернее окно. Main не ждет завершения окна
            Console.WriteLine("Показывали дочернее окно");
            Console.ReadKey(); //Затормозили работу программы
            Console.WriteLine("Запускаем главное окно предложения");
            WND y = new WND("fdhd"); //Создали новое окно, что бы показать как главное окно приложе
            Application.Run(y); //Закончило работу главное окно приложение

        }

    }

    class WND:Form
    {
        public WND(string Titl)
        {
            Text = Titl;
            FormClosing += a;

        }

        void a(object sender, FormClosingEventArgs b)
        {
            Console.WriteLine("Закрываем окно{0} {1}", b.CloseReason, b.Cancel);
        }
    }
}

```

```

        if (b.CloseReason == CloseReason.UserClosing)
        {
            DialogResult RS =
                MessageBox.Show("Точно закрыть", "Вопрос", MessageBoxButtons.OKCancel);
            if (DialogResult.Cancel == RS) {
                b.Cancel = true;
            }
        }
    }
}

---- End Of File:./cs/labs/showwnd/program.cs

```

16.2.3 Создание диалогового окна OkCancel

В разделе [12.2](#) изложен порядок выполнения лабораторной. Далее можно посмотреть законченный пример.

16.2.3.1 Законченный пример

Использование окна:

```

---- File:./cs/labs/okcancel/program.cs

using System;
using System.Windows.Forms;

namespace ConsoleApplication5{
    class Program    {
        [STAThread]
        static void Main()    {
//            Console.OutputEncoding = Encoding.GetEncoding( "cp866" );
            Form w = new OkCancel("пример диалогового окна");
            DialogResult rc = w.ShowDialog();
            if (rc == DialogResult.OK)
            {
                Console.WriteLine("Вы нажали Ok");
            }
            else
                Console.WriteLine("Вы нажали Cancel");
        }
    }
}

---- End Of File:./cs/labs/okcancel/program.cs

```

Класс для окна:

---- File:./cs/labs/okcancel/okcancel.cs

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

namespace ConsoleApplication5{
    class OkCancel:Form    {
        public OkCancel(string qwert)    {
            Text = qwert;
            int p = 10;
            this.Padding = new Padding (p,p,p,p);

            this.FormBorderStyle = FormBorderStyle.FixedDialog;
            this.MinimizeBox = false;
            this.MaximizeBox = false;
            this.ControlBox = false;
            this.AutoScroll = false;
            this.AutoSize = true;
            this.Size = new Size(10, 10);

            Panel p0 = new Panel ( ); // эта панель задает высоту строки
            p0.Size = new Size (1, 32);
            p0.Dock = DockStyle.Top;
            p0.BorderStyle = BorderStyle.Fixed3D;
            p0.BackColor = Color.Green;
            Controls.Add (p0);

            Button butOk = new Button ( );
            butOk.Size = new Size( 112,32);
            butOk.Dock = DockStyle.Right;
            butOk.Text = "Ok";
            Controls.Add(butOk);

            p0 = new Panel(); // эта панель задает расстояние между кнопками
            p0.Size = new Size( 112,32);
            p0.Dock = DockStyle.Right;
            p0.BorderStyle = BorderStyle.Fixed3D;
            p0.BackColor = Color.Red;
            Controls.Add(p0);

            Button butCancel = new Button ( );
            butCancel.Size = new Size( 112,32);
            butCancel.Dock = DockStyle.Right;
            butCancel.Text = "Cancel";
            Controls.Add(butCancel);

            butOk.DialogResult = DialogResult.OK;
            butCancel.DialogResult = DialogResult.Cancel;

            this.DialogResult = DialogResult.Cancel;
            this.AcceptButton = butOk;
            this.CancelButton = butCancel;
```

```

        this.StartPosition = FormStartPosition.CenterScreen;
    }
}
}
---- End Of File:./cs/labs/okcancel/okcancel.cs

```

16.2.4 Создание диалогового окна ввода данных

В разделе 12.3 изложен порядок выполнения лабораторной. Далее можно посмотреть законченный пример.

16.2.4.1 Законченный пример

Использование окна:

```

---- File:./cs/labs/inwnd/program.cs

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;

namespace laba3
{
    class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            fld a = new fld("Введите имя пользователя", "Admin");
            fld b = new fld("Пароль", "*****");

            Form w = new inWnd("Введите имя и пароль", a, b);
            DialogResult rc = w.ShowDialog();
            if (rc == DialogResult.OK)
            {
                Console.WriteLine("Вы ввели пароль '{0}' '{1}'", a.value, b.value);
            }
            else
                Console.WriteLine("Вы отказались от ввода");
        }
    }
}
---- End Of File:./cs/labs/inwnd/program.cs

```

Класс для окна:

```

---- File:./cs/labs/inwnd/inwnd.cs

```

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;

namespace laba3 {
    class fld {
        public string name;
        public string def;
        public string value;
        public fld(string nm, string df)
        {
            name = nm;
            def = df;
            value = "";
        }
    }

    class inWnd : OkCancel
    {
        Dictionary<TextBox, fld> flds = null;
        public inWnd(string Title, fld a, fld b) : base(Title)
        {
            flds = new Dictionary<TextBox, fld>(); // создать список пар
            flds.Add(addFld (1, b), b);           /// добавили второе поле ввода
            flds.Add(addFld (0, a), a);           /// добавили первое поле ввода
            butOk.Click += _KeyDown;
        }

        public TextBox addFld                               /// метод добавляет в окно новое поле ввода
            (int n, fld par){
            int h = 32;

            Panel p = new Panel();                        /// панель содержит метку и поле ввода
            p.Name = string.Format("p{0}", n);
            p.BorderStyle = BorderStyle.FixedSingle;
            p.Size = new Size(1, h);
            p.AutoSize = true;
            p.Dock = DockStyle.Top;
            p.Padding = new Padding (0,0,0,8);           ///

            Panel p1 = new Panel();                        /// панель для высоты
            p1.BorderStyle = BorderStyle.FixedSingle;
            p1.Size = new Size(1, h);
            p1.Dock = DockStyle.Top;                      /// <-----
            p1.BackColor = Color.Blue;
            p.Controls.Add(p1);

            Label l1;
            TextBox t1;

```

```

        l1 = new Label();
        l1.Name      = string.Format("l{0}", n);
        l1.Size      = new Size(172 , h);;
        l1.Text      = par.name;  /// создать новую строку
        l1.Dock      = DockStyle.Right ;

//          Console.Error.WriteLine ("new field: '{0}'/'{1}'/'{2}'"
//          , par.name,  l1.Text, par.value);
        l1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
        p.Controls.Add(l1);

        t1 = new TextBox();
        t1.TabIndex = n;
        t1.Name      = string.Format("t{0}", n);;
        t1.Size      = new Size(162 , h);
        t1.Text      = par.def;
        t1.Dock      = DockStyle.Right ;
        p.Controls.Add(t1);
        Controls.Add(p);          /// готовое поле ввода
        return t1;
    }

    void _KeyDown(object sender, EventArgs e) {
        TextBox tb;
        fld     f;
        foreach (KeyValuePair<TextBox, fld> Item in flds)
        {
            tb = Item.Key;
            f  = Item.Value;
            f.value = tb.Text;
            tb.Text = f.def;          /// а теперь восстанавливается значение по умолчанию.
            Console.Error.WriteLine("keyDown: '{0}':'{1}' ", f.name, f.value);
        }
    }
}

}

---- End Of File: ./cs/labs/inwnd/inwnd.cs

```

Раздел [12.3](#) содержит класс OkCancel, который используется в качестве родителя окна inWnd (с изменением пространства имен).

16.2.5 Меню в главном окне приложения

В разделе [12.4](#) изложен порядок выполнения лабораторной. Далее можно посмотреть законченный пример.

16.2.5.1 Законченный пример

```

---- File: ./cs/labs/winmenu/program.cs

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;

namespace lab3
{
    class Program
    {
        [STAThread]
        static void Main(string[] args)
        {
            Application.Run(new x());
        }
    }

    class x : Form
    {
        StatusStrip statStrip;
        public x()
        {
            Menu = new MainMenu();
            MenuItem FileIt = new MenuItem("File");
            MenuItem workIt = new MenuItem("work");
            MenuItem AboutIt = new MenuItem("About");
            MenuItem ExitIt = new MenuItem("Exit");

            FileIt.MenuItems.Add(ExitIt);

            AboutIt.Click += aboutF;
            ExitIt .Click += exitF;
            workIt.Click += workF;

            Menu.MenuItems.Add(FileIt);
            Menu.MenuItems.Add(workIt);
            Menu.MenuItems.Add(AboutIt);
            //      Menu.MenuItems.Add(ExitIt);
            statStrip = new StatusStrip();
            ToolStripStatusLabel
            statLabel = new ToolStripStatusLabel();
            statStrip.Items.Add(statLabel);

            statLabel = new ToolStripStatusLabel();
            statStrip.Items.Add(statLabel);          ///
            Controls.Add(statStrip);
            statStrip.Items[0].Text = "окно готово";
            statStrip.Items[1].Text = "дочерних окон нет";

        }

        void aboutF(object sender, EventArgs a)

```



```

        {
            Console.WriteLine("about");
            statStrip.Items[1].Text = "открыли окно About";
            MessageBox.Show("This programm was made by ImiaRek, Kiev 201?.");
            statStrip.Items[1].Text = "дочерних окон нет";
        }
        void workF(object sender, EventArgs a)
        {
            Console.WriteLine("work");
            statStrip.Items[1].Text = "окно работы не готово";
        }

        void exitF(object sender, EventArgs x)
        {
            Console.WriteLine("FormClosing");
            //ClickEvent FormClosing = FormClosedEventHandler;
            MessageBox.Show("exit");
            Close();
        }
    }
}

---- End Of File:./cs/labs/winmenu/program.cs

```

16.2.6 Представление табличных данных

Методические указания см. в разделе [12.5.2](#)

16.2.6.1 Законченный пример

Родительский класс без управляющих элементов:

```

---- File:./cs/labs/datagrid/wnd.cs

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

namespace dtGrd
{
    public class wnd : Form
    {
        string fileName;
        protected
        DataGridView dgv = new DataGridView();

        public wnd(string fnm)
        {
            fileName = fnm;
        }
    }
}

```

```

        DataGridViewColumn clm = new DataGridViewColumn();
        clm.Name = "Col 1";

        dgv.ReadOnly = true;
        dgv.Dock = DockStyle.Fill;
        dgv.AllowUserToAddRows = false;
        dgv.BackgroundColor = Color.White;
        dgv.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        Load += fLoad;
        Controls.Add(dgv);
    }

    void fLoad(object sender, EventArgs a)    {
        Console.WriteLine("Load");
        ReadFile(fileName);
    }

    public void ReadFile(string path) {
        string str;
        string[] fields;
        bool firsttime = true;
        using (StreamReader sr = new StreamReader(path))    {
            while ((str = sr.ReadLine()) != null)    {
                //пропустить пустые строки
                fields = str.Split(';');
                if (fields.Length <= 0)
                    continue;
                if (firsttime){
                    dgv.ColumnCount = fields.Length;
                    for (int i = 0; i < fields.Length; i++) {
                        dgv.Columns[i].Name = String.Format("Pole{0}", i+1);
                    }
                    firsttime = false;
                }
                dgv.Rows.Add(fields);
            }
        }
    }
}

```

---- End Of File:./cs/labs/datagrid/wnd.cs

Класс - наследник со статус строкой и полосой инструментов:

---- File:./cs/labs/datagrid/wnd2.cs

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

```

```

namespace dtGrd{
    public class wnd2 : wnd    {
        ToolBar tb = new ToolBar();
        StatusStrip statStrip;

        public wnd2(string fnm):base (fnm)    {
            statStrip = new StatusStrip();
            ToolStripStatusLabel
            statLabel = new ToolStripStatusLabel();
            statStrip.Items.Add(statLabel);
            Controls.Add(statStrip);
            statStrip.Items[0].Text = "окно готово";

            tb.ButtonSize = new System.Drawing.Size((int)(200/ 3), (int)(40) );

            ToolBarButton tlbExit = new ToolBarButton("Exit");
            tlbExit.ToolTipText = "закрыть окно таблицы";
            ToolBarButton tIns = new ToolBarButton("Insert");
            tIns.ToolTipText = "добавить запись";
            ToolBarButton tEdit = new ToolBarButton("Edit");
            tEdit.ToolTipText = "редактировать запись";
            ToolBarButton tDel = new ToolBarButton("Delete");
            tDel.ToolTipText = "удалить запись";
            ToolBarButton tSave = new ToolBarButton("Save");
            tSave.ToolTipText = "сохранить изменения";
            ToolBarButton tExp = new ToolBarButton("Export");
            tExp.ToolTipText = "экспортировать таблицу";

            Padding = new Padding(2);
            tb.Buttons.AddRange(new ToolBarButton[] {
                tIns
                , tEdit
                , tDel
                , tSave
                , tExp
                , tlbExit
            }
            );

            tb.ButtonClick += new ToolBarButtonClickEventHandler(ToolBarButtonClick);

            tb.Dock = DockStyle.Top;
            Controls.Add(tb);
            Load += fLoad;
        }
        void fLoad(object sender, EventArgs a)
        {
            statStrip.Items[0].Text
            = string.Format("{0} records has been red", dgv.Rows.Count);
        }
    }
}

```

```

void ToolBarButtonClick(object sender, ToolBarButtonEventArgs e)
{
    string bNm = e.Button.Text;
    statStrip.Items[0].Text
    = string.Format("You've pressed {0} button", bNm);

    if (bNm == "Exit")
        Close();
    else if (bNm == "Open")
        ;
    else {
        MessageBox.Show("action not ready!", "Warning");
    }
}
}
}

```

---- End Of File: ./cs/labs/datagrid/wnd2.cs

16.2.7 Открытие нескольких дочерних окон

Методические указания см. в разделе [12.7](#)

Далее приводится текст полученного класса главного окна.

16.2.7.1 Класс главного окна приложения для законченного примера

---- File: ./cs/labs/mdi/program.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Reflection;
using System.IO;
using System.Drawing;
using dtGrd;

namespace mdi
{
    class Program
    {
        static public Form mWin;
        [STAThread]
        static void Main(string[] args)
        {
            mWin = new x();
        }
    }
}

```

```

        mWin.IsMdiContainer = true;    //<<<
        Application.Run(mWin);
    }
}
class x : Form    {
    StatusStrip statStrip;
    public x()    {
        Menu = new MainMenu();
        MenuItem FileIt = new MenuItem("File");
        MenuItem DataIt = new MenuItem("Data");
        MenuItem AboutIt = new MenuItem("About");
        MenuItem ExitIt = new MenuItem("Exit");

        MenuItem SupplierIt = new MenuItem("Supplier");
        MenuItem PartIt = new MenuItem("Part");
        MenuItem ProjectIt = new MenuItem("Project");
        MenuItem DeliveryIt = new MenuItem("Delivery");
        MenuItem WinIt = new MenuItem("Windows");
        WinIt.MdiList = true;    //<<<

        FileIt.MenuItems.Add(ExitIt);

        AboutIt.Click += aboutF;
        ExitIt .Click += exitF;

        SupplierIt.Click += SupplierF;
        PartIt .Click += PartF ;
        ProjectIt .Click += ProjectF ;
        DeliveryIt.Click += DeliveryF;

        DataIt.MenuItems.Add(SupplierIt);
        DataIt.MenuItems.Add(PartIt);
        DataIt.MenuItems.Add(ProjectIt);
        DataIt.MenuItems.Add(DeliveryIt);

        Menu.MenuItems.Add(FileIt);
        Menu.MenuItems.Add(DataIt);
        Menu.MenuItems.Add(WinIt);
        Menu.MenuItems.Add(AboutIt);
        statStrip = new StatusStrip();
        ToolStripStatusLabel
        statLabel = new ToolStripStatusLabel();
        statStrip.Items.Add(statLabel);

        statLabel = new ToolStripStatusLabel();
        statStrip.Items.Add(statLabel);    ///
        Controls.Add(statStrip);
        statStrip.Items[0].Text = "окно готово";
        statStrip.Items[1].Text = "дочерних окон нет";
    }
}

bool shouldIOpen (string text){
    for (int i = 0; i < MdiChildren.Count(); i++){
        if (this.MdiChildren[i].Name == text) {

```

```

        MdiChildren[i].Activate();
        return false;
    }
}
return true;
}
void SupplierF(object sender, EventArgs a) {
    Console.WriteLine("Поставщики");
    string id = "supplier";
    if (shouldIOpen(id)) {
        statStrip.Items[1].Text = "окно поставщиков не готово";
        Form z = new wnd2(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
            + "/s.csv");
        z.MdiParent = Program.mWin;          //<<<
        z.Text = "Поставщики";
        z.Name = id;
        z.Show();
    }
    statStrip.Items[1].Text = "Поставщики загружены";
}
void PartF(object sender, EventArgs a)
{
    Console.WriteLine("Детали");
    string id = "part";
    if (shouldIOpen(id)) {
        statStrip.Items[1].Text = "окно деталей не готово";
        Form z = new wnd2(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
            + "/p.csv");
        z.MdiParent = Program.mWin;
        z.Text = "Детали";
        z.Name = id;
        z.Show();
    }
    statStrip.Items[1].Text = "Детали загружены";
}
void ProjectF(object sender, EventArgs a)
{
    Console.WriteLine("Проекты");
    string id = "project";
    if (shouldIOpen(id)) {
        statStrip.Items[1].Text = "окно проектов не готово";
        Form z = new wnd2(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
            + "/j.csv");
        z.MdiParent = Program.mWin;
        z.Text = "Проекты";
        z.Name = id;
        z.Show();
    }
    statStrip.Items[1].Text = "Проекты загружены";
}
}

```

```

void DeliveryF(object sender, EventArgs a)
{
    Console.WriteLine("Поставки");
    statStrip.Items[1].Text = "окно поставок не готово";
    string id = "delivery";
    if (shouldIOpen(id)) {

        Form z = new wnd2(
            Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
            + "/spj.csv");
        z.MdiParent = Program.mWin;
        z.Text = "Поставки";
        z.Name = id;
        z.Show();
    }
    statStrip.Items[1].Text = "Поставки загружены";

}

void aboutF(object sender, EventArgs a)
{
    Console.WriteLine("about");
    statStrip.Items[1].Text = "открыли окно About";
    MessageBox.Show("This programm was made by ImiaRek, Kiev 201?.");
    statStrip.Items[1].Text = "дочерних окон нет";
}

void exitF(object sender, EventArgs x)
{
    Console.WriteLine("FormClosing");
    //ClickEvent FormClosing = FormClosedEventHandler;
    MessageBox.Show("exit");
    Close();
}

}

}

---- End Of File:./cs/labs/mdi/program.cs

```

16.2.8 Экспорт таблицы

Сначала требуется скопировать все файлы лабораторной 16.2.6 в новый каталог. Далее, выполнять следующие действия:

- В методе ReadFile класса wnd строчку

```
using (StreamReader sr = new StreamReader(path))
```

заменить на строчку

```
using (StreamReader sr = new StreamReader(path, Encoding.GetEncoding (1251)))
```

что бы явно задать кодировку файла.

- В делегате - обработчике ToolBarButtonClick класса wnd2 добавить действия для реакции при нажатии оператором нажатия на кнопку "Export":

```
else if (bNm == "Export")
    export (dgv, ',');
```

- К классу wnd2 добавить метод экспорта файлов - export:

```
public void
export (DataGridView dgvw    \\ экспортируемый дата грид
      , char sep){           \\ символ - разделитель полей
                                \\ в файле экспорта
}
```

- В метод export добавить применение стандартного окна сохранения файла SaveFileDialog. В поле диалога для фильтрации файлов Filter задать четыре вида файлов:

```
"csv files (*.csv)|*.csv|txt files (*.txt)|*.txt| xml files (*.xml)|*.xml|All files (*.*)|*.*"
```

- Добавить виртуальный метод- заглушку для вывода таблицы в xml файл:

```
public virtual
string xmlExport (string nm, DataGridView dgvw){
    string rc = string.Format ("Can't export data to {0} file", Path.GetFileName(nm));
    MessageBox.Show(rc, "Wrning! XML export it's not ready!");
    return rc;
}
```

- Добавить метод- заглушку для вывода таблицы в xml файл:

```
public virtual
string csvExport (string nm, char sep_, DataGridView dgvw){
    string rc = "error";
    return rc;
}
```

- В метод export добавить код для вызова методов - заглушек:

```
if (svFileDialog1.ShowDialog() == DialogResult.OK)
{
    string fnm = svFileDialog1.FileName;
    string ext = Path.GetExtension (fnm);
    if (ext == ".xml")
        statStrip.Items[0].Text = xmlExport (fnm, dgvw);
    else {
        string ss = csvExport (fnm, ',', dgvw);
        statStrip.Items[0].Text = ss;
    }
}
```


- Добавить в метод - заглушку csvExport цикл для прохода по записям таблицы dgvw:

```
int rNo = 0;
using (StreamWriter wr = new StreamWriter(nm, false, Encoding.GetEncoding (1251)))
    DataGridViewCellCollection row;
    string[] flds;
    for(rNo = 0; rNo < dgvw.RowCount;  rNo++){
        // обработка текущей записи таблицы тут
    }
    rc =  string.Format("{0} records has been exported to file {1}"
        , rNo, Path.GetFileName(nm) );
}
```

- Добавить в метод - заглушку csvExport цикл для прохода по полям записи таблицы dgvw:

```
row = dgvw.Rows[rNo].Cells;
flds = new string[row.Count]; // узнать кол-во полей
for(int cNo = 0; cNo < row.Count ; cNo++){ // скопировать поля
    flds[cNo] = (row[cNo]).Value.ToString();
}
string rec = string.Join(sep, flds); // построить строчку для вывода
wr.WriteLine(rec);
```

Разработка приложения лабораторной закончено. Далее, требуется запустить приложение и попробовать выполнить экспорт в файлы с расширением xml и с другими расширениями.

Замечание

Позднее виртуальные методы у объектов классов - наследников будут заменяться на другие методы, чтобы продемонстрировать механизм позднего связывания.

16.2.9 Окно редактирования таблицы

Методические указания см. в разделе [12.6](#)

16.2.9.1 Законченный пример

Далее приводится законченный пример разработанного класса wnd3:

---- File:./cs/labs/edittbl/part2/wnd3.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using System.IO;
```

```
using System.Reflection;
using ConsoleApplication5;
```

```
namespace dtGrd
{
    public class wnd3 : wnd2
    {
        public wnd3(string fnm):base (fnm)
        {
            dgv.SelectionMode
                = DataGridViewSelectionMode.FullRowSelect; // выделяем всю запись
            dgv.RowHeadersVisible = false;                // слева удаляем лишнюю колонку

            tb.ButtonClick -= base.ToolBarButtonClick;
            tb.ButtonClick += ToolBarButtonClick3;
            dgv.CellDoubleClick += dc2;                  // редактирование записи по enter
            dgv.PreviewKeyDown += _PreviewKeyDown;
            dgv.KeyDown += _KeyDown;
        }

        void _PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                Console.WriteLine("CellDoubleClick: selrow / row: {0} "
                    , dgv.CurrentRow.Index);
                doEdit();
            }
        }

        void _KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                e.SuppressKeyPress = true;
            }
        }

        protected
        void dc2(object sender, DataGridViewCellEventArgs e)
        {
            Console.WriteLine("CellDoubleClick: selrow / row: {0}/{1} "
                , dgv.CurrentRow.Index, e.RowIndex);

            if (e.RowIndex >= 0){
                doEdit();
                dgv.CurrentCell = dgv.Rows[e.RowIndex].Cells[0];
            }
        }
    }
}
```

```

}

protected
void ToolBarButtonClick3(object sender, ToolBarButtonClickEventArgs e)
{
    string bNm = e.Button.Text;
    statStrip.Items[0].Text
= string.Format("You've pressed {0} button", bNm);
    if (bNm == "Exit")
        Close();
    else if (bNm == "Insert")
        doIns();
    else if (bNm == "Edit")
        doEdit();
    else if (bNm == "Save")
        doSave();
    else {
        base.ToolBarButtonClick
        ( sender, e);
    }
}

public void doEdit(){
    string ss = "";
    if(dgv.RowCount > 0) {

        DataGridViewRow dgvR = dgv.Rows[dgv.CurrentRow.Index];
        DataGridViewCellCollection row = dgvR.Cells;/// это не строка

        List<fld> flds = new List<fld>();
        fld f;
        for (int i = 0; i< dgv.ColumnCount; i++){
            f = new fld (dgv.Columns[i].Name
                        , (row[i]).Value.ToString());
            flds.Add(f);
        }
        if(flds.Count> 0){
            Form w = new inWnd("Edit selected record", flds.ToArray());
            DialogResult rc = w.ShowDialog();
            if (rc == DialogResult.OK) {
                for (int kk = 0; kk< flds.Count; kk++){
                    dgvR.Cells[kk].Value = flds[kk].value;
                }
            }
        }
    }
    else
        ss = "nothing to dp!";
    Console.WriteLine("Edit: '{0}''", ss);
    statStrip.Items[0].Text = ss;
}

```


16.2.10 Рисование масштабированной ломаной

Раздел [13.3](#) содержит методические указания для рисования масштабированной кривой.

Предметный указатель

- *NIX, [3](#), [37](#)
- ./cs/2bin/i2.cs, [160](#)
- ./cs/2bin/p.cs, [158](#)
- ./cs/2bin/r2.cs, [160](#)
- ./cs/app.cs, [109](#)
- ./cs/args/p.cs, [47](#), [203](#)
- ./cs/chart/1.cs, [174](#)
- ./cs/chart/compile.cmd, [176](#)
- ./cs/checked/p.cs, [140](#)
- ./cs/cs.cmd, [113](#)
- ./cs/cs2.cmd, [3](#)
- ./cs/deadblock/p.cs, [179](#)
- ./cs/encoding/utf8.cs, [164](#)
- ./cs/encoding2/866to1251.cs, [163](#)
- ./cs/gpx2csv/p.cs, [179](#)
- ./cs/inpolygon/program.cs, [168](#)
- ./cs/labs/2xml/p.cs, [204](#)
- ./cs/labs/2xml/test.cmd, [58](#)
- ./cs/labs/bin2txt/program.cs, [212](#)
- ./cs/labs/datagrid/program.cs, [209](#)
- ./cs/labs/datagrid/wnd.cs, [208](#), [222](#)
- ./cs/labs/datagrid/wnd2.cs, [223](#)
- ./cs/labs/edittbl/part2/wnd3.cs, [230](#)
- ./cs/labs/inwnd/inwnd.cs, [218](#)
- ./cs/labs/inwnd/program.cs, [218](#)
- ./cs/labs/mdi/program.cs, [225](#)
- ./cs/labs/mod1lab3/p.cs, [39](#)
- ./cs/labs/mod3lab1/p.cs, [202](#)
- ./cs/labs/mod3lab1/wnd.cs, [60](#)
- ./cs/labs/okcancel/okcancel.cs, [217](#)
- ./cs/labs/okcancel/program.cs, [216](#)
- ./cs/labs/showwnd/program.cs, [215](#)
- ./cs/labs/txt2bin/program.cs, [211](#)
- ./cs/labs/winmenu/program.cs, [220](#)
- ./cs/lambda/p.cs, [162](#)
- ./cs/lambda/p.cs, [138](#)
- ./cs/list/p.cs, [125](#)
- ./cs/map/1.txt, [186](#)
- ./cs/map/p.cs, [182](#)
- ./cs/map/wget.cmd, [186](#)
- ./cs/menustrip/form1.cs, [165](#)
- ./cs/override/program.cs, [132](#)
- ./cs/resxgen/eyelst/_resgen.cmd, [86](#)
- ./cs/resxgen/ut/pics.lst, [193](#)
- ./cs/resxgen/ut/program.cs, [189](#)
- ./cs/resxgen/ut/test.cmd, [193](#)
- ./cs/run/program.cs, [176](#)
- ./cs/serialization/1.txt, [199](#)
- ./cs/serialization/1.xml, [199](#)
- ./cs/serialization/program.cs, [194](#)
- ./cs/serialization/record.cs, [198](#)
- ./cs/tennis/program.cs, [170](#)
- ./cs/threads/program.cs, [177](#)
- ./cs/timer/timer.cs, [162](#)
- ./cs/wget/wget.cs, [167](#)
- ./date/date/j.csv, [81](#)
- ./date/date/p.csv, [81](#)
- ./date/date/s.csv, [81](#)
- ./date/date/spj.csv, [82](#)
- /debug, [7](#)
- Бертран Мейер, [158](#)
- Карделли, [158](#)
- Люка Карделли, [131](#)
- МГС84, [92](#)
- Стрейчи, [131](#)
- Строгое Имя Сборки, [12](#)
- абстрактный класс, [17](#), [128](#)
- аргумент командной строки, [42](#)
- автоматически реализуемые свойства, [18](#), [137](#)
- библиотека динамической компоновки, [113](#)
- блокнот (notepad.exe), [41](#)
- делегат - обработчик, [79](#), [103](#), [214](#)
- делегат-обработчик, [214](#), [229](#)
- динамически подгружаемая библиотека, [113](#)
- экспорт, [210](#)
- форма, [10](#)
- функция - замыкание, [139](#)
- главное окно приложения, [67](#)
- главное окно приложения, [60](#)
- глобальная именованная область видимости, [16](#), [114](#)
- глобальная область видимости, [16](#), [114](#)
- именованная область видимости, [37](#)
- импорт функций из системной библиотеки, [162](#)
- интерфейс, [50](#)
- исключительное состояние, [17](#), [128](#)
- исполняемый файл, [113](#)
- класс, [158](#)

- ключ, 201
- ключевое слово out, 16, 39, 119
- ключевое слово ref, 16, 39, 119
- командная строка MS Visual Studio, 37
- командный файл, 11
- компиляции приложений в среде MONO, 37
- компиляция с ключом /debug, 7
- компилятором ресурсов, 86
- контекстное меню, 27, 104
- контур, 105
- линия экрана для рисования, 90
- локальная область видимости, 16, 114
- лямбда - выражение, 18, 138
- метод доступа, 137
- метод с переменным числом параметров (пример), 204
- наследование, 132
- не обязательный аргумент командной строки, 49
- немедленное прекращение работы приложения, 43
- область видимости класса (структуры), 16, 114
- обязательный аргумент командной строки, 49
- окно, 10
- окно ввода данных, 76, 77
- оператор вызова метода, 115
- оператор вызова метода, 123
- параметр ключа командной строки, 201
- параметрический полиморфизм, 131
- перечисление, 46
- передача фактического параметра, 5, 39
- перегрузка функций, 131
- переменная - делегат, 52, 73
- переменных - делегатов, 52
- подписать метод на событие, 91
- полиморфизм, 131
- полиморфизм (пример), 204
- полиморфизм включения, 131
- полоса инструментов, 71, 72, 77, 94
- позднее связывание, 230
- позднее связывание, 132
- приложение касса, 12, 23
- принцип подстановки Лисков, 158
- продолжительность тика, 162
- пространство имен, 37
- раннее связывание, 132
- разделитель дробной части, 193
- ресурс приложения, 25
- ресурсы приложения, 24, 85
- сборка, 113
- сборщик мусора, 38
- случайные числа, 18, 135
- состояние кассы, 23
- состояние кассы, 12
- спецификатор места вывода, 7, 38
- сравниваемые элементы, 127
- стандартный вывод, 38
- стандартный вывод ошибок, 38
- статус строки, 67
- свойство, 137
- свойство класса, 17, 123
- шаблон, 50
- шаблон, 131
- таймер, 170
- тик, 162
- тип, 158
- тип - делегат, 118
- тип - делегат, 52, 103
- точка входа в приложения, 38
- управляющий элемент (окна), 19, 60
- 866, 6
- 1251, 6
- accessor, 137
- ad hoc полиморфизм, 131
- Add, 124
- args, 47
- AppendFormat, 138
- Arg.mkVHelp, 47
- ArgFlg, 204
- ArgFloat, 204
- ArgStr, 204
- ArrayList, 124
- as, 115
- auto-implemented properties, 18, 137
- BinaryReader, 54
- BinaryReader, 122
- BinaryWriter, 54
- BinaryWriter, 122
- break, 117
- break, 16, 117
- CellDoubleClick, 80
- char, 16, 120

Chart, [47](#)
 check, [47](#)
 checked , [140](#)
 cmd , [35](#)
 Console, [56](#)
 ContextMenu, [27](#), [104](#)
 ContextMenuStrip, [27](#), [104](#)
 continue , [117](#)
 continue , [16](#), [117](#)
 Count , [124](#)
 CSV файл, [210](#)
 Current , [117](#)

 DataGridView, [69](#), [210](#)
 DateTime , [119](#)
 DialogResult, [46](#)
 diff, [8](#), [12](#), [23](#)
 Directory, [18](#), [135](#)
 DirectoryInfo, [18](#), [135](#), [136](#)
 Dispose, [49](#), [54](#)
 Dispose , [141](#)
 doxygen, [32](#)
 dynamic, [19](#), [141](#)

 Encoding, [56](#), [163](#)
 Environment.Exit(1);, [43](#)
 Exception, [16](#), [17](#), [117](#), [128](#)
 eyelst.resx, [193](#)

 File, [18](#), [54](#), [135](#)
 FileMode, [54](#)
 FileNotFoundException, [54](#)
 FileStream, [18](#), [135](#)
 FileStream , [139](#)
 fld, [211](#)
 foreach , [141](#)
 foreach , [117](#)
 Form, [46](#)
 FormatException , [121](#)

 getPoints, [88](#), [206](#)
 getRec, [44](#), [206](#)
 goto , [16](#), [117](#)
 GPX - файл, [179](#)
 GraphicsPath, [27](#), [31](#), [104](#), [155](#), [170](#)
 graphviz, [32](#)

 HashTable , [124](#)
 HResult, [46](#)

 IComparer , [127](#)
 IComparable , [141](#)
 IComparable , [127](#)
 IDisposable, [17](#), [49](#), [125](#)
 IDisposable , [141](#)
 IEnumerable , [141](#)
 InputEncoding, [56](#)
 Insert, [80](#)
 interface , [136](#)
 Interlocked, [28](#), [177](#)
 Invalidate, [26](#), [92](#)
 inWnd, [74](#), [211](#)
 is , [115](#)
 IsMdiContainer , [143](#)

 KeyNotFoundException, [57](#)

 Load, [70](#), [72](#)
 lock, [28](#)

 m, [210](#)
 Main, [38](#)
 mapping, [92](#)
 MaxValue , [121](#)
 MdiChildren , [29](#), [143](#)
 MdiList , [29](#), [143](#)
 MdiParent, [29](#), [143](#)
 Message, [46](#)
 MessageBox, [46](#)
 MinValue , [121](#)
 Monitor, [28](#), [177](#)
 MoveNext , [117](#)

 named pipe , [141](#)
 NumberDecimalSeparator, [93](#)

 object, [19](#), [141](#)
 OkCancelDlg, [211](#)
 Open, [54](#)
 OpenFileDialog, [22](#), [69](#)
 OverflowException , [121](#)

 Paint, [92](#)
 Parse , [121](#)
 Parse , [120](#)
 piclst.resources , [86](#)
 piclst.resx, [86](#)
 PreviewKeyDown, [80](#)

 Queue , [124](#)

 Random , [135](#)

Read , 16, 121
ReadLine, 16, 121
ReadToEnd, 16, 121
Regex , 139
Region, 31, 155, 170
Region , 27, 104
resgen.exe, 25, 86
ResXResourceWriter, 86, 189
return , 16, 117
rLine, 92

SaveFileDialog, 210, 229
sLine, 89, 90, 92
Source, 46
Stack , 124
StackTrace, 46
STAThreadAttribute, 46
static , 118
StatusStrip, 67, 71, 72
Stream, 54, 57, 106
StreamReader, 18, 135
StreamReader , 16, 121
StreamWriter , 16, 121
StringBuilder, 18, 137
System.Collections , 124
System.Collections.Generic , 124
System.Windows.Forms.dll, 46
System.Xml, 11

TargetSite, 17, 46, 128
TextReader, 55
TextReader , 16, 121
TextWriter, 16, 55, 121
Thread, 28
thread , 141
throw , 16, 117
timeBeginPeriod, 162
timeEndPeriod, 162
TimeSpan , 119
ToolBar, 71, 72, 77
ToolBarButton, 72
ToolStripStatusLabel, 67
ToolTipText, 72
ToString, 16, 121
TryParse , 121
TryParse , 120

unchecked , 140
using, 54
using , 17, 125

UTF-16, 6, 42
UTF-16 , 119, 120
UTF-8, 6, 11, 164

var, 19, 141

wget, 186
winmm.dll, 162
wnd, 78
Write , 16, 121
WriteLine, 16, 121

X.rec2bin, 158
X.rec2bin , 131
X.str2rec, 158
X.str2rec , 131
X.txt2bin , 131
XML файл, 189, 210

yield , 16, 117

Список литературы

- [1] Cardelli, Abadi. A theory of objects, 1996, Springer. – https://drive.google.com/open?id=1mShdbLP3LnooSSfk80sh_SAtIc54Jczu. 160
- [2] Math Parser. Math Parser.NET. <https://www.codeproject.com/Articles/274093/Math-Parser-NET>. 14
- [3] MicroSoft. Руководство по программированию на C#. – <https://docs.microsoft.com/ru-ru/dotnet/csharp/>. 7, 9, 44
- [4] Robert C.Martin. The Liskov Substitution Principle, 2003. C++ Report, March 1996, <http://www.objectmentor.com/resources/articles/lsp.pdf>. 161
- [5] Бертран Мейер. Основы объектно-ориентированного программирования. Абстрактные типы данных (АТД). <https://b-ok.cc/dl/2951261/89645a>. 161
- [6] Лавринович В.Ю. Накладання треків з gps-координатами на растрові карти відкритих тайлових сервісів / В.Ю. Лаврінович. // Тези XVI міжнародної науково-практичної конференції молодих учених і студентів «ПОЛІТ.Сучасні проблеми науки. Інформаційно-діагностичні системи». - Київ: НАУ - 2016 - 262 с. 36
- [7] В.Ю.Лавринович. Програмна реалізація машини Тьюринга/ Л.М. Соснев, В.Ю. Лавринович// Тези хві міжнародної науково-практичної конференції молодих учених та студентів «ПОЛІТ. Сучасні проблеми науки. Інформаційно-діагностичні системи». – К.:НАУ, 2016, С.159, <http://er.nau.edu.ua:8080/handle/NAU/36796>. 14, 31
- [8] Вахромеева Л.А. Картография: Учебник для вузов. – М: Недра, 1981. - 224 с. 14
- [9] Г.Майерс. Надежность программного обеспечения/. М.: Мир, 1980, - 359 с. <https://www.twirpx.com/file/2351752/>. 62
- [10] К.М. Грінченко. Перерахунок координат з системи СК-42 в wgs84 і навпаки/ К.М. Грінченко. //Тези XVI міжнародної науково-практичної конференції молодих учених та студентів «ПОЛІТ. Сучасні проблеми науки. Інформаційно-діагностичні системи» - НАУ, Київ.— 2016. 36
- [11] ДСТУ 3008:2015. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ. Структура та правила оформлювання / уклали: В. Земцева; Ю. Поліщук, Р. Санченко, Л. Шрамко; А. Ямчук. – Київ, ДП. 38
- [12] ДСТУ ГОСТ 7.1:2006. Бібліографічний запис, бібліографічний опис. Загальні вимоги та правила складання : метод. рекомендації з впровадження / уклали: Галевич О. К., Штогрин І. М. – Львів, 2008. – 20 с. 39
- [13] К. Дж. Дейт. Введение в системы баз данных = An Introduction to Database System, 7th Edition. . — М.: Вильямс, 2001. — С. 1072. — ISBN 5-8459-0138-3. 26, 27, 31, 84

- [14] Т.І. Ковальчук. Відображення даних відеореєстратора на растрових картах/Т.І. Ковальчук, Р.О. Остапчук// xviii Міжнародна науково-практична конференція молодих учених і студентів «Політ-2018». – Тезиси докладів. К.:НАУ, 2018. 36
- [15] Культин Н. Б. Microsoft Visual C# в задачах и примерах. – СПб.: БХВ-Петербург, 2009. – 320 с. 7
- [16] Пискунов А.Г. Doxygen и Graphviz: документирование проектов на C# / А.Г. Пискунов, А.С.Горбань // Журн. Алгоритм.– №4(10), – 2006. Интернет ресурс: <http://www.realcoding.net/dn/docs/dgIntro.pdf>. 35, 36, 105
- [17] Пискунов А.Г. Graphviz и Sed: построение схем иерархии наследования / А.Г. Пискунов, С.М. Петренко // Журн. Алгоритм.– №3(9). – 2006. Интернет ресурс: <http://www.softcraft.ru/design/graphvizsed/mkhrrchy.pdf>. 35
- [18] Пискунов А.Г. SQLite и введение в разработку клиент - серверных приложений на языке C# / 2019. Интернет ресурс: <http://agp1.adr.com.ua/docs/db.Labs.pdf>. 7
- [19] Пискунов А.Г. Библиотека обработки аргументов командной строки. / А.Г.Пискунов, Д.И.Гись. // LXXIV-а наукова конференція професорсько-викладацького складу, аспірантів, студентів та співробітників відокремлених структурних підрозділів університету. – К.: НТУ, 2018. – с.421, <http://agp1.adr.com.ua/args.html>. 12, 35, 50, 69, 192, 196, 206, 209, 214
- [20] Пискунов А.Г. Журналирование многосредовых приложений. Пояснительная записка / А.Г.Пискунов, Р.В.Скаковский. Интернет ресурс: <http://agp1.adr.com.ua/Logger.html>. 35, 112, 192, 196
- [21] Пискунов А.Г. Использование TCL/TK: База данных для хранения курсов валют с нуля. Пояснительная записка / 2009. Интернет ресурс: <http://agp1.adr.com.ua/docs/tclUsage.pdf>. 36
- [22] Пискунов А.Г. Майерс, Г.: Композиционное проектирование приложения / 2009. Интернет ресурс: <https://docplayer.ru/48434435-G-mayers-kompozicionnoe-proektirovanie-prilozheniya.html>. 62
- [23] Пискунов А.Г. Об одном примере нарушения принципа подстановки Лисков /Пискунов А.Г., Петренко С.М.//CITForum.Ru, 2010. Интернет ресурс: <http://citforum.ru/programming/digest/lspv/>. 37
- [24] Пискунов А.Г. Об отличиях между понятиями типа и класса / Вестник Киевского университета. Компьютерные науки. – 2015. – № 3. Интернет ресурс: http://irbis-nbuv.gov.ua/cgi-bin/irbis-nbuv/cgiirbis_64.exe?I21DBN=LINK&P21DBN=UJRN&Z21ID=&S21REF=10&S21CNR=20&S21STN=1&S21FMT=ASP_meta&C21COM=S&2_S21P03=FILE=&2_S21STR=VKNU_fiz_mat_2015_3_22. 37, 161

- [25] Пискунов А.Г. Опыт эксплуатации одной системы журналирования приложений / А.Г.Пискунов, Р.В.Скаковский. // Проблемы інфокомунікацій: Матеріали другої всеукраїнської науково-технічної конференції. – Полтава : ПолтНТУ; Київ : НТУ; Харків: НТУ «ХПІ»; Полтава : ВКСС ВІТІ, 2018. – С. 74, <https://drive.google.com/open?id=10eE0nSzhkZ1z8xR5q0qcSg4jQ2kd5Ecj>. 35, 136
- [26] Пискунов А.Г. Пояснительная записка. Эллипсоиды и геометрические вычисления / А.Г.Пискунов, К.М.Гринченко, И.А.Юрчук. // Кафедра Прикладной Математики. – Киев: ИИДС - 2017 - 248 с. 35, 36
- [27] Пискунов А.Г. Проектирование и декомпозиция двунаправленных потоков данных интерактивных систем / 2009 . Интернет ресурс: <http://er.nau.edu.ua:8080/bitstream/NAU/39629/1/dataFlow.pdf>. 35, 62
- [28] Пискунов А.Г. Разработка консольных и оконных приложений на языке C# / 2019. Интернет ресурс: <http://agp1.adr.com.ua/docs/smp.Labs.pdf>. 7
- [29] Пискунов А.Г. Разработка консольных приложений на языке Си (Часть 1) / 2018. Интернет ресурс: <http://agp1.adr.com.ua/docs/amp.Labs.pdf>. 7, 41
- [30] Пискунов А.Г. Разработка консольных приложений с элементами C++ (Часть 2) / 2018. Интернет ресурс: <http://agp1.adr.com.ua/docs/ppz.Labs.pdf>. 7, 41
- [31] Пискунов А.Г. Система ограничения доступа Вежа. Пояснительная записка / 2000. Интернет ресурс: <http://agp1.adr.com.ua/software/mlc.pdf>. 27, 37
- [32] Пискунов А.Г. Типы, множества и классы / 2010. Интернет ресурс: <http://er.nau.edu.ua:8080/handle/NAU/38528>. 161
- [33] Пискунов А.Г. Экспресс - методика билатерального тестирования индивидуально - типологических свойств высшей нервной деятельности человека / Пискунов А.Г., Седаков И.С., Трошихин В.В.// 1996. Интернет ресурс: <http://agp1.adr.com.ua/mpr.html>. 37
- [34] Пискунов А.Г. (Пер. с англ.). Классы и типы в языках, основанных на классах / 2010. Интернет ресурс: <https://drive.google.com/open?id=1M9-geni3uvVJt1C41eWmK1D0q3T70-8m>. 160
- [35] Пискунов А.Г. (Пер. с англ.). Неполное Руководство по SQLite для пользователей Windows / Grant Allen, Mike Owens: The Definitive Guide to SQLite // 2017. Интернет ресурс: <https://drive.google.com/file/d/10XPFuUPYgzxu05cwhdnqsse3p1Qq0-vs/view?usp=sharing>. 164
- [36] Рихтер Дж. Программирование на платформе Microsoft .NET Framework / Пер. с англ. . – 2-е изд., испр. – М.: Издательско – торговый дом “Русская редакция”, 2003. – 512 с. 7
- [37] Стив Кост. Некоммерческий картографический проект OSM. – <http://www.openstreetmap.org>. 36

- [38] Н.С. Сусідик. Автоматизований збір висот точок місцевості/Н.С. Сусідик // xviii Міжнародна науково-практична конференція молодих учених і студентів «Політ-2018». – Тезиси докладів. К.:НАУ, 2018. 36
- [39] Троелсен Э. С# и платформа. NET. Библиотека программиста / Пер. с англ. . – СПб.: Питер, 2004. – 796 с. 7
- [40] Шалыто А.А. Синхронное программирование / А.А.Шалыто, Д.Г.Шопырин. // Информационно - управляющие системы. – С-Петербург: 2004, №3, – с.35-42, <http://www.softcraft.ru/auto/switch/syncprog/syncprog.pdf>. 62
- [41] Шилдт Г. Полный справочник по С#/ Пер. с англ. . – М.: Издательский дом "Вильямс", 2004. – 752 с. 7