

Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Кузбасский государственный технический университет  
имени Т.Ф. Горбачева»

Институт профессионального образования  
Кафедра информатики и информационных систем

Кирилл Владимирович Ерошечин

## **РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ**

Методические материалы к практическим занятиям,  
лабораторным и самостоятельным работам

Рекомендовано цикловой методической комиссией по  
специальности СПО 09.02.07 Информационные системы  
и программирования в качестве электронного издания  
для использования в образовательном процессе

Кемерово 2024

Рецензенты: Кулиничев К.А. – преподаватель кафедры информатики и информационных систем ИПО ФГБОУ ВО «Кузбасский государственный технический университет имени Т.Ф. Горбачева»

**Ерошевич, К.В. Разработка кода информационных систем:** методические материалы к практическим занятиям, лабораторным и самостоятельным работам для студентов специальности СПО 09.02.07 Информационные системы и программирование очной формы обучения / сост. К. В. Ерошевич; Кузбасский государственный технический университет имени Т.Ф. Горбачева». – Кемерово, 2024. – Текст: электронный.

Методические материалы для дисциплины «Разработка кода информационных систем» содержат перечень самостоятельных, практических и лабораторных работ, контрольные вопросы к ним.

© Кузбасский государственный  
технический университет  
имени Т.Ф. Горбачева, 2024  
© Ерошевич К. В.,  
составление, 2024

## ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ .....	4
ЛАБОРАТОРНАЯ РАБОТА 1. РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ .....	5
ЛАБОРАТОРНАЯ РАБОТА 2. ПЕРЕНАПРАВЛЕНИЕ ПОТОКОВ ВВОДА-ВЫВОДА .....	17
ЛАБОРАТОРНАЯ РАБОТА 3. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ В ПРОГРАММЕ .....	29
ЛАБОРАТОРНАЯ РАБОТА 4. ОДНОМЕРНЫЕ МАССИВЫ.....	41
ЛАБОРАТОРНАЯ РАБОТА 5. МНОГОМЕРНЫЕ МАССИВЫ.....	53
ЛАБОРАТОРНАЯ РАБОТА 6. КЛАССЫ. ОБЪЕКТНОЕ МОДЕЛИРОВАНИЕ .....	63
ЛАБОРАТОРНАЯ РАБОТА 7. КОНСТРУКТОР КЛАССА. ПЕРЕГРУЗКА МЕТОДОВ КЛАССА .....	77
ЛАБОРАТОРНАЯ РАБОТА 8. ПРОЕКТИРОВАНИЕ ИЕРАРХИИ КЛАССОВ.....	84
ЛАБОРАТОРНАЯ РАБОТА 9. ПОЛИМОРФИЗМ НА ОСНОВЕ ИНТЕРФЕЙСОВ.....	94
ЛАБОРАТОРНАЯ РАБОТА 10. ОСНОВЫ РАБОТЫ С ФАЙЛАМИ .....	104
ЛАБОРАТОРНАЯ РАБОТА 11. ОБРАБОТКА ДАННЫХ В ФАЙЛАХ.....	129
ЛАБОРАТОРНАЯ РАБОТА 12. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ.....	151
ЛАБОРАТОРНАЯ РАБОТА 13. ОСНОВЫ РАБОТЫ С XML-ДОКУМЕНТАМИ НА ПЛАТФОРМЕ .NET FRAMEWORK .....	168
ЛАБОРАТОРНАЯ РАБОТА 14. СОЗДАНИЕ ЗАПРОСОВ К ДОКУМЕНТУ XML ПРИ ПОМОЩИ ЯЗЫКА XPATH .....	205

ЛАБОРАТОРНАЯ РАБОТА 15. ОСНОВЫ СОЗДАНИЯ ЗАПРОСОВ К КОЛЛЕКЦИЯМ ОБЪЕКТОВ С ПОМОЩЬЮ LINQ.....	239
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1. МЕТОДЫ СОРТИРОВКИ.....	268
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2. МЕТОДЫ ПОИСКА .....	281
СПИСОК ЛИТЕРАТУРЫ.....	292



## ПРЕДИСЛОВИЕ

Целью освоения дисциплины «Разработка кода информационных систем» является приобретение обучающимися знаний в области проведения тестирования программного обеспечения (ПО) в процессе его создания.

Основными задачами изучения дисциплины «Разработка кода информационных систем», являются:

1. Изучение способов моделирования программных продуктов.
2. Написание кода программных средств.
3. Отладка программных средств.

В соответствии с учебным планом изучение дисциплины «Разработка кода информационных систем» предусматривает проведение лекционных, практических занятий, лабораторных работ и самостоятельной работы обучающимися.

При подготовке к практическим занятиям обучающиеся самостоятельно изучают основную и дополнительную литературу, готовят конспекты по темам, предложенным преподавателем.

На практических занятиях преподаватель осуществляет контроль подготовки качества знаний обучающегося, используя: опрос, обсуждение вопросов по темам изучаемой дисциплины, письменный опрос при текущем контроле и предоставление отчетов по практическим занятиям.

# ЛАБОРАТОРНАЯ РАБОТА 1. РАЗРАБОТКА КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

## ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* научиться работать с переменными и константами простых типов в С#.

Задачи лабораторной работы:

- научиться объявлять переменные простых типов в языке С#;
- научиться объявлять константы простых типов в языке С#;
- научиться выполнять простейшие действия с переменными и константами.

## ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

### Структура консольного приложения

Рассмотрим структуру консольного приложения на языке С#, созданного с использованием средств MS Visual Studio (рис. 1.1).

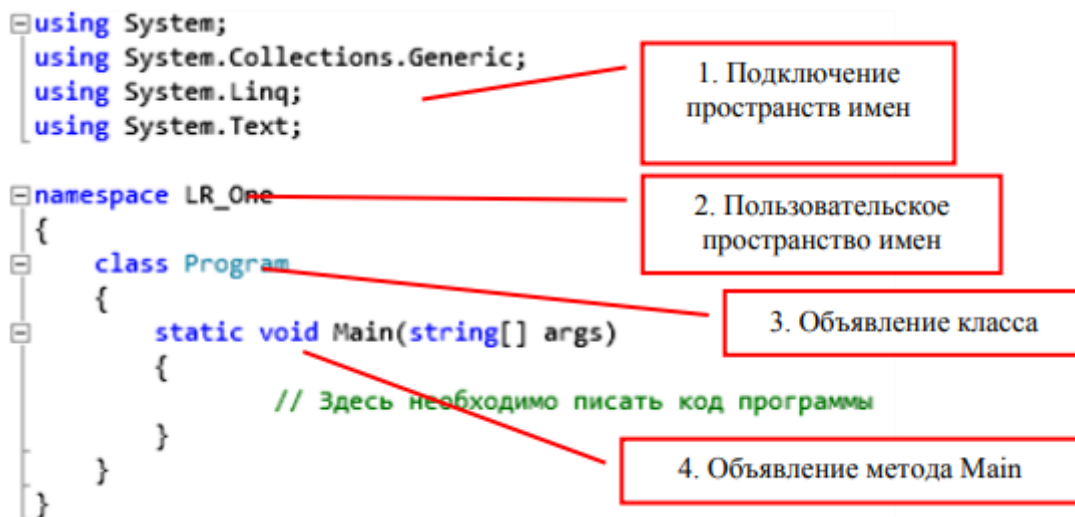


Рисунок 1.1 – Исходный код консольного приложения

В интерактивном редакторе кода среды разработки код может быть свернут/развернут с использованием кнопок +/-, введенных в код. На рис. 1.1 в области 1 показано подключение про-

пространств имен (библиотек, содержащих стандартные инструменты) с использованием зарезервированного слова `using`.

Программист сам создает свое пространство имен с именем `LR_One` (2), в котором объявляется класс с именем `Program`.

В классе `Program` объявлен один метод – функция `Main` (параметры функции не рассматриваем).

Функция `Main` имеет особенное значение в программировании на языках `C`, `C++` и `C#`.

Функцию `Main` называют «точкой входа», то есть началом выполнения программы. Далее мы рассмотрим приложения, содержащие множество функций. Операционная система знает, с какой именно функции начать выполнение программы – с функции `Main`. Очевидно, что имя этой функции менять нельзя. Это должен быть статический метод класса (или структуры), возвращающий либо значение типа `int`, либо `void`. Хотя нередко модификатор `public` указывается явно, поскольку по определению этот метод должен быть вызван извне программы, на самом деле неважно, какой уровень доступа вы назначите методу точки входа. Он запустится, даже если вы пометите его как `private`.

Когда компилируется консольное или Windows-приложение `C#`, по умолчанию компилятор ищет в точности один метод `Main` () с описанной выше сигнатурой в любом классе и делает его точкой входа программы. Если существует более одного метода `Main` (), компилятор возвращает сообщение об ошибке.

Обратите внимание на вложенность конструкций на рис. 1.1: в пространство имен `LR_One` вложен класс `Program`, в класс `Program` вложена функция `Main`. В свою очередь, в функции `Main` содержатся инструкции на языке `C#`-код, который начнет выполняться при старте программы.

В `C#`, как и в других `C`-подобных языках, большинство операторов завершаются точкой с запятой (;) и могут продолжаться в нескольких строках без необходимости указания знака переноса. Операторы могут быть объединены в блоки с помощью фигурных скобок (`{ }`). Однострочные комментарии начинаются с двойного слеша (`//`), а многострочные – сослеша со звездочкой (`/*`) и заканчиваются противоположной комбинацией (`*/`). В этих аспектах язык `C#` идентичен `C++` и `Java`.

Следует также помнить о том, что язык C# чувствителен к регистру символов. Это значит, что переменные с именами myVar и MyVar являются разными.

Причина присутствия оператора using в файле Program.cs связана с использованием библиотечных классов.

Весь код C# должен содержаться внутри класса. Объявление класса состоит из ключевого слова class, за которым следует имя класса и пара фигурных скобок. Весь код, ассоциированный с этим классом, размещается между этими скобками.

### Типы значений C#

Язык C# поддерживает 8 predefined целочисленных типов (таблица 1.1).

Таблица 1.1

Целочисленные типы C#

Имя типа	Тип CTS	Описание	Диапазон (минимум : максимум)
sbyte	System.SByte	8-битное целое со знаком	-128 : 127
short	System.Int16	16-битное целое со знаком	-32 768 : 32 767
int	System.Int32	32-битное целое со знаком	-2 147 483 648 : 2 147 483 647
long	System.Int64	64-битное целое со знаком	$-2^{63} : 2^{63} - 1$
byte	System.Byte	8-битное целое без знака	0 : 255
ushort	System.UInt16	16-битное целое без знака	0 : 65 535
uint	System.UInt32	32-битное целое без знака	$0 : 2^{32} - 1$
ulong	System.UInt64	64-битное целое без знака	$0 : 2^{64} - 1$

Язык C# также поддерживает и типы с плавающей точкой (таблица 1.2).

Таблица 1.2

## Типы с плавающей точкой C#

Имя типа	Тип CTS	Описание	Кол-во знаков
float	System.Single	32-битное с плавающей точкой одинарной точности	7
double	System.Double	64-битное с плавающей точкой двойной точности	15/16

В таблице 1.3 представлен десятичный тип C#. Данный тип реализован для финансовых операций.

Таблица 1.3

## Десятичный тип C#

Имя типа	Тип CTS	Описание	Кол-во знаков
decimal	System.Decimal	128-битное с плавающей точкой в десятичной нотации с высокой точностью	28

Как и во многих языках программирования существует булевский тип (таблица 1.4).

Таблица 1.4

## Булевский тип

Имя типа	Тип CTS	Значения
bool	System.Boolean	true или false

Литералы типа char записываются как одиночные, заключенные в одинарные кавычки символы: 'F', 'w', 'ц', 'Я' и т.д.

В переменных типа char можно хранить и специальные символы в виде управляющих последовательностей (таблица 1.6).

Таблица 1.6

Представление символов в виде управляющих последовательностей.

Управляющая последовательность	Символ
\'	Одиночная кавычка
\"	Двойная кавычка
\\	Обратный слэш
\0	Пусто
\a	Предупреждение (звуковой сигнал)
\b	Забой
\f	Подача формы
\n	Новая строка
\r	Возврат каретки
\t	Символ табуляции
\v	Вертикальная табуляция

Если отдельные символы объединены в строку, то необходимо использовать тип `string`, который отображается на тип CTS – `System.String`.

### Объявление и инициализация переменных в C#

Синтаксис объявления переменных в C# выглядит следующим образом: *ТипДанных ИдентификаторПеременной;*

Например:

```
int a;
```

Этот код объявляет переменную типа `int` с именем `a`. Компилятор не позволит использовать эту переменную до тех пор, пока она не будет инициализирована (т.е. пока ей не будет присвоено значение).

Для инициализации переменной `a` необходимо написать следующий код:

```
a = 123;
```

Переменную можно инициализировать во время объявления:

```
int b = 7;
```

Или

```
string str = "Hello, World!!!";
```

Синтаксис С# позволяет объявить несколько переменных (и инициализировать их) одного типа в одной синтаксической конструкции.

Например:

```
float b, i, myPerem, U_t = 0.3F, z_11 = 23.56F;
```

В данном примере объявляется 5 переменных типа float, некоторые из них инициализируются в процессе объявления.

### **Объявление и инициализация констант в С#**

*Константа* – это переменная, значение которой не меняется за время выполнения программы. Для объявления константы необходимо воспользоваться ключевым словом const. Пример:

```
const char simv = 'A';  
const double pi = 3.14;
```

Очевидно, что при таком объявлении, поменять значения simv и pi в дальнейшем будет нельзя.

## **МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

### **Создание приложения для вывода персональной информации**

1. Создайте консольное приложение, для этого выполните следующие действия:

- а. Выберите команду главного меню File→ New→ Project...
- б. В открывшемся диалоговом окне (рис. 1.2) выберите необходимые настройки для создаваемого проекта: язык Visual C#; фреймворк: .NET Framework 4 и выше; шаблон: Console Application.

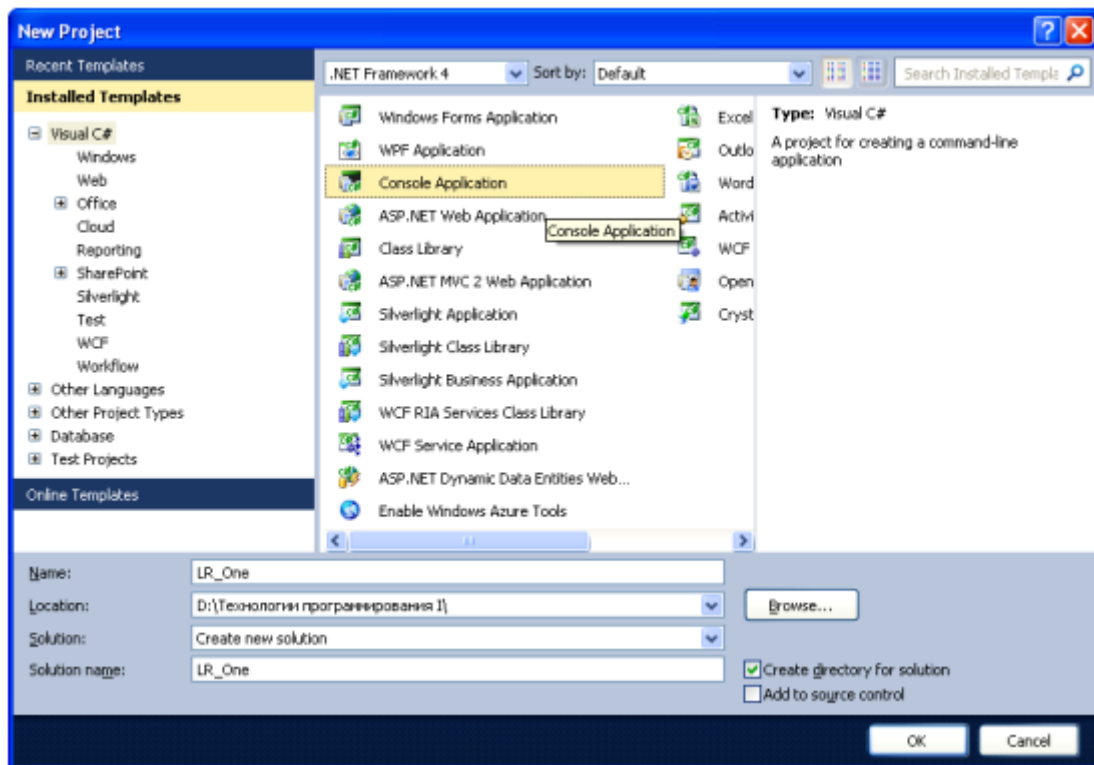


Рисунок 1.2 – Создание нового проекта консольного приложения

с. В текстовом поле Name введите имя проекта (например LR\_One).

d. В текстовом поле Location выберите место сохранения нового проекта.

е. Установите флажок-переключатель «Create directory for solution».

f. Нажмите кнопку «ОК».

2. После выполнения пункта 1 в среде разработки откроется новый созданный проект (рис. 1.3).



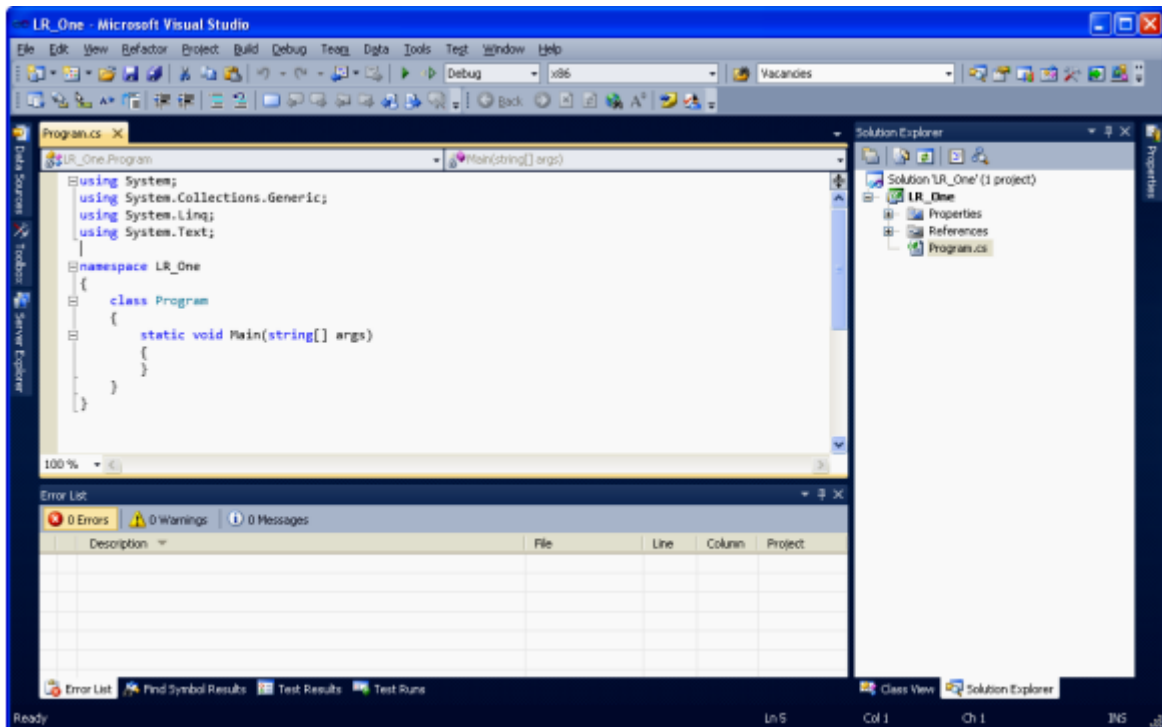


Рисунок 1.3 – Новый проект, загруженный в среду разработки: вкладка «Solution Explorer» отображает состав проекта (нас интересует только файл Program.cs); в левой части окна (сверху) открыт файл Program.cs в редакторе кода

3. На данном этапе необходимо ознакомиться со структурой исходного файла консольного приложения (рис. 1.4).


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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            // Здесь необходимо писать код программы
        }
    }
}
```

Рисунок 1.4 – Исходный файл консольного приложения

4. Весь код программы необходимо писать внутри функции Main.

5. Для построения сборки (исполняемого exe-файла) выполните команду главного меню Build → Build Solution (или использовать горячую клавишу F6 ). После этого сборка создана, но приложение не будет запущено автоматически

6. Для создания сборки и последующего запуска программы можно воспользоваться командой Debug → Start Debugging главного меню среды разработки или нажать кнопку панели инструментов  Start Debugging (F5) . Можно также использовать горячую клавишу F5.

7. Запустите приложение на выполнение одним из методов, указанных в пункте 6. Окно консольного приложения появится и исчезнет. Это означает, что приложение выполнило все команды, написанные программистом, и завершило свою работу.

8. Для удержания окна на экране измените исходный файл в соответствии с рисунком 1.5. В функции Main добавлен вызов только одной команды Console.ReadKey ( ) – эта функция останавливает выполнение программы и ждет, когда пользователь нажмет любую клавишу

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ReadKey();
        }
    }
}
```

Рисунок 1.5 – Исходный файл консольного приложения для предотвращения закрытия окна консольного приложения

9. Запустите измененное приложение, убедитесь, что окно удерживается на экране.

10. Добавьте несколько строк кода в исходный файл (рис. 1.6).

```

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №1");
            Console.WriteLine("");
            Console.WriteLine("Выполнил: Иванов Иван иванович");
            Console.WriteLine("Группа: ИСТБ-121");
            Console.WriteLine("Наименование ЛР: Структура консольного приложения");
            Console.WriteLine("");
            Console.WriteLine("для завершения работы программы нажмите любую клавишу...");

            Console.ReadKey();
        }
    }
}

```

Рисунок 1.6 – Исходный файл консольного приложения для вывода информации на экран.

11. Внимательно изучите исходный код примера на рис. 1.6. Запустите приложение и убедитесь, что отсутствуют ошибки и информация выводится.

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

### Задание 1

Измените приложение, созданное в ходе выполнения данной лабораторной работы таким образом, чтобы программа выводила на экран следующую информацию (каждый студент должен использовать персональную информацию о себе):

- Название и номер лабораторной работы;
- ФИО студента;
- Группа студента и шифр специальности;
- Дата рождения студента;
- Населенный пункт постоянного места жительства студента;
- Любимый предмет в школе;
- Краткое описание увлечений, хобби, интересов.

### Задание 2

Создайте второй проект (или добавьте функционал после вывода информации о себе). Объявите требуемые переменные,

присвойте им начальные значения (определите самостоятельно, значения какого типа могут принимать переменные), выведите на экран с использованием форматной строки значения переменных и результат вычисления выражения в соответствие с вариантом (таблица 1.7).

Таблица 1.7

## Варианты задания

Вариант	Выражение для вычисления
1	$F = a\_1 + b - a \cdot (x + y5)$
2	$Se = w111 + bt - x + y \cdot w$
3	$R\_x = a \cdot b + b/t - x + f \cdot i\_2$
4	$c = gh + b \cdot q3 - x + y/w$
5	$H = \frac{g \cdot h}{d17} + b/h1 - \frac{x + y}{4}$
6	$Z = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$
7	$U = \frac{35}{a} \cdot z + z \cdot a - \frac{a + E \cdot t}{4}$
8	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
9	$Se = w111 + bt - x + y \cdot w$
10	$R\_x = a \cdot b + b/t - x + f \cdot i\_2$
11	$g = w \cdot h + b \cdot q3 - q3 + y/w$
12	$ee = \frac{g \cdot h}{d17} + d17/h - \frac{g + d17 + h}{4}$
13	$Zze = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$
14	$t = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
15	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
16	$Zxy = a\_5 + b - a\_5 \cdot (b + y)$
17	$ch = \frac{6}{f1} \cdot z + z \cdot f1 - \frac{f1 + Et}{6}$

Продолжение таблицы 1.7

18	$H = \frac{g \cdot h}{d17} + b / h1 - \frac{x + y}{4}$
19	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
20	$R\_x = 15 + a \cdot b + b / t - x + f \cdot i\_2$
21	$X = \frac{g \cdot h}{d17} + d17 / h - \frac{g + d17 + h}{4}$
22	$t = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
23	$ch = rx \cdot z + z \cdot f1 - \frac{f1 + Et}{rx}$
24	$R = 2 \cdot x + 2 \cdot y - 4 \cdot x \cdot y + z$
25	$U = rx \cdot z + z \cdot Y - \frac{Y + z}{rx}$
26	$Se = w111 + bt - x + y \cdot w$

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая функция имеет особенное значение при выполнении программы на языках C, C++, C#?
2. Что такое «точка входа» в программе?
3. Как вы понимаете термины «пространства имен», «класс», «метод», «функция»? Напишите определение каждому термину.
4. Что такое переменная? Как объявляется переменная?
5. Как объявляется константа? Чем константа отличается от переменной?
6. Перечислите целочисленные типы C#.
7. Перечислите отличия типов char и string.
8. Какие из перечисленных идентификаторов нельзя использовать в качестве имен пользовательских переменных?

**\_1\_01, b100, int, double\_1, \_b200, MyVar, create-var, 4perem, \_5elem, zo0, wodoo, UserCount, system\_call, string, System.Double.**

## ЛАБОРАТОРНАЯ РАБОТА 2. ПЕРЕНАПРАВЛЕНИЕ ПОТОКОВ ВВОДА-ВЫВОДА

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* научиться перенаправлять стандартные потоки ввода-вывода.

Задачи лабораторной работы:

- научиться сохранять состояние потоков ввода-вывода;
- научиться объявлять новые потоки, ассоциированные с файлами;
- научиться решать задачи с вводом-выводом в файлы;
- научиться восстанавливать состояние потоков.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Для выполнения данной лабораторной работы потребуется теоретический материал, изученный в предыдущей лабораторной работе, а также дополнительные сведения о механизмах перенаправления стандартных потоков ввода-вывода.

Работа с файлами будет рассмотрена в данном курсе позже, но можно реализовать чтение и запись в файлы без глубокого знания потоков и других возможностей пространства имен System.IO.

Рассмотрим простое приложение (рисунок 2.1):

```

1  using System;
2
3  namespace Test
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              int a, b, c;
10             float res = 0;
11             a = Convert.ToInt32(Console.ReadLine());
12             b = Convert.ToInt32(Console.ReadLine());
13             c = Convert.ToInt32(Console.ReadLine());

```

```

14
15     res = a * b * c;
16
17     Console.WriteLine(res);
18
19     Console.ReadKey();
20
21 }
22
23

```

Рисунок 2.1 – Пример программы, использующей ввод-вывод

В представленной программе выполняются простые действия: объявляются переменные, считываются с клавиатуры и конвертируются значения, производится вычисление и производится вывод в консоль. На рисунке 2.2 показан результат работы программы



```

5
6
210

```

Рисунок 2.2 – Работа программы, использующей ввод-вывод

Данная программа может быть изменена путем перенаправления стандартных потоков ввода (по умолчанию данный поток связан с клавиатурой) и вывода (по умолчанию связан с экраном, консолью). На рисунке 2.3 показан механизм перенаправления потоков ввода-вывода.

```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             TextWriter save_out = Console.Out;
11             TextReader save_in = Console.In;
12             var new_out = new StreamWriter(@"output.txt");
13             var new_in = new StreamReader(@"input.txt");
14             Console.SetOut(new_out);
15             Console.SetIn(new_in);

```



```

12      var new_out = new StreamWriter(@"output.txt");
13      var new_in = new StreamReader(@"input.txt");
14      Console.SetOut(new_out);
15      Console.SetIn(new_in);
16
17      int a, b, c;
18      float res = 0;
19      a = Convert.ToInt32(Console.ReadLine());
20      b = Convert.ToInt32(Console.ReadLine());
21      c = Convert.ToInt32(Console.ReadLine());
22
23      res = a * b * c;
24
25      Console.WriteLine(res);
26
27      Console.SetOut(save_out); new_out.Close();
28      Console.SetIn(save_in); new_in.Close();
29  }
30 }
31

```

Рисунок 2.3 – Листинг программы, использующей перенаправленный ввод-вывод

Программа, представленная на рисунке 2.3 выглядит аналогично рассмотренной ранее, но добавлены строки: 2, 1-15, 27, 28. То есть сам алгоритм не изменился, дополнительные строки просто перенаправляют ввод-вывод.

В строке 2 подключается пространство имен System.IO, которое содержит необходимые типы для работы с файлами

В строках 10, 11 сохраняются стандартные потоки System.In и System.Out в переменные save\_in и save\_out.

В строке 12 создается объект new\_out типа StreamWriter, который ассоциируется с файлом output.txt. В строке 13 создается объект new\_in типа StreamReader, который ассоциируется с файлом input.txt.

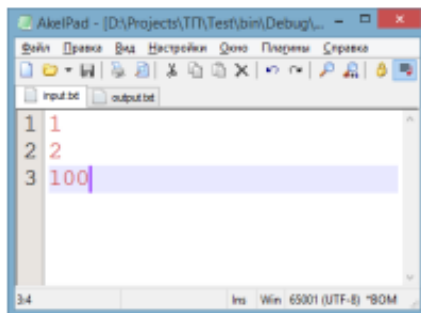
В строках 14 и 15 в качестве новых источников стандартных потоков устанавливаются объекты new\_in и new\_out. Производится данное действие с использованием методов Console.SetIn() и Console.SetOut().

После таких манипуляций любые чтения из консоли на самом деле производятся из файла input.txt, а любой вывод в консоль производится в файл output.txt.

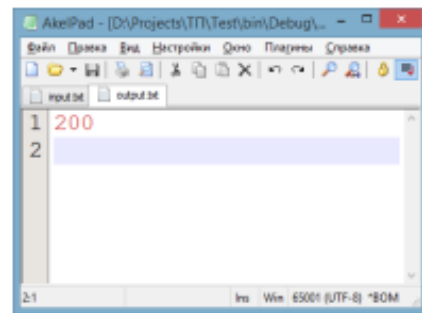


Важно понимать, что если потоки были перенаправлены, то их необходимо вернуть к начальному состоянию после завершения работы программы. Поэтому, в строке 27 функцией `Console.SetOut` устанавливается изначальное значение выходного потока, которое было сохранено в переменной `save_out`. В этой же строке производится закрытие файла `output.txt` путем вызова `new_out.Close()`. В строке 28 аналогичные действия производятся для входного потока.

В результате работы программы будет сформирован файл `output.txt` на основе входного файла `input.txt` (рисунок 2.4).



а)



б)

Рисунок 2.4 – Файлы, используемые программой в качестве источников ввода-вывода: а) входной файл; б) выходной файл

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Требуется вычислить значения двух выражений **geo** и **my\_tmp**:

$$my\_tmp = a\_71 \times \sqrt{\frac{u2 - test}{x}} \times \frac{zoo}{a71 - zoo},$$

$$geo = \sqrt{\frac{x}{u2}} \times (a\_71)^2 \times test,$$

на основании значений переменных **a\_71**, **\_u2**, **test**, **zoo**, **x**

*Входные данные:* вещественные числа **a1**, **a2**, **a3**, **a4**, **a5**, каждое не меньше 0, но не превосходит  $10^5$ .

*Выходные данные:* вещественные числа **s** и **k**. Результаты необходимо округлить до сотых. Если вычислить значение выражения невозможно, то вывести слово **ERROR**. Результат округлить до тысячных.

Таблица 2.1

Пример входных и выходных данных

Пример input.txt	Пример output.txt
5 10 20 11 100000	ERROR 50000,000
500,2384 20,0024 14 11 10,1997	8,628 2501697,151

Решение на языке C#. Перед непосредственным программированием решения задачи необходимо проанализировать условие. На основе анализа можно сделать следующие выводы:

- Переменные `x` и `_u2` не могут быть равными 0.
- Значение `_u2` - test должно быть неотрицательным.
- Значение `a_71` - zoo должно быть отличным от 0.

Решение предложенной задачи представлено на рисунке 2.5

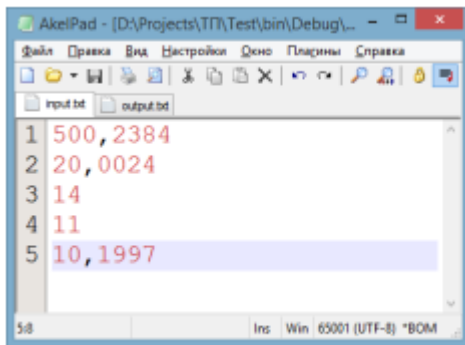
```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             TextWriter save_out = Console.Out;
11             TextReader save_in = Console.In;
12             var new_out = new StreamWriter(@"output.txt");
13             var new_in = new StreamReader(@"input.txt");
14             Console.SetOut(new_out);
15             Console.SetIn(new_in);
16
17             double a_71, _u2, test, zoo, x;
18             double my_tmp, geo;
19             a_71 = Convert.ToDouble(Console.ReadLine());
20             _u2 = Convert.ToDouble(Console.ReadLine());
21             test = Convert.ToDouble(Console.ReadLine());
22             zoo = Convert.ToDouble(Console.ReadLine());
23             x = Convert.ToDouble(Console.ReadLine());
24
25             if ((x == 0) || (a_71 - zoo == 0) || (_u2 - test < 0))
26                 Console.WriteLine("ERROR");
27             else
28             {
29                 my_tmp = a_71 * Math.Sqrt((_u2 - test) / x) * zoo / (a_71 - zoo);
30                 Console.WriteLine(String.Format("{0:0.000}", my_tmp));
31
32
33                 if (_u2 == 0)
34                     Console.WriteLine("ERROR");
35                 else
36                 {
37                     geo = Math.Sqrt(x / _u2) * a_71 * a_71 * test;
38                     Console.WriteLine(String.Format("{0:0.000}", geo));
39                 }
40
41                 Console.SetOut(save_out); new_out.Close();
42                 Console.SetIn(save_in); new_in.Close();
43             }
44         }
45     }

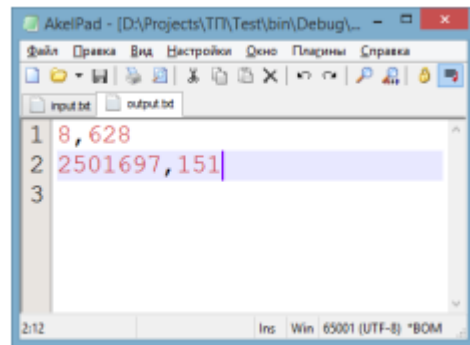
```

Рисунок 2.5 – Решение задачи о вычислении выражения с перенаправлением потоков ввода-вывода

В результате работы программы будет сформирован выходной файл, содержащий результаты вычислений (рисунок 2.6).



а)



б)

Рисунок 2.6 – Файлы, используемые программой в качестве источников ввода-вывода: а) входной файл; б) выходной файл

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Разработайте приложение на языке C# для решения задачи в соответствии с индивидуальным вариантом.

Таблица 2.2

Индивидуальные задания к лабораторной работе 2.

Вариант	Задание
1, 14	<p>Требуется вычислить площадь треугольника по трем сторонам <b>a</b>, <b>b</b>, <b>c</b> (формула Герона). Заданный треугольник существует.</p> <p><i>Входные данные:</i> Три вещественных числа <b>a</b>, <b>b</b>, <b>c</b>, каждое из которых больше 1, но не превосходит 1000.</p> <p><i>Выходные данные:</i> Одно вещественное число – площадь треугольника со сторонами <b>a</b>, <b>b</b>, <b>c</b>. Результат необходимо округлить до тысячных.</p>
2, 15	<p>Требуется вычислить длину окружности радиуса <b>r</b> и площадь образованного ей круга. Число <math>\pi</math> принять равным 3,14.</p> <p><i>Входные данные:</i> Одно вещественное число <b>r</b>, <math>0 &lt; r &lt; 105</math>. <i>Выходные данные:</i> Два вещественных числа: <b>L</b> – длина окружности; <b>S</b> – площадь круга. Результат необходимо округлить до тысячных</p>

## Продолжение таблицы 2.2

3, 16	<p>Требуется вычислить значение выражения <math>s</math>:</p> $s = a1 \cdot \sqrt{a2 - a3} \cdot \frac{a4}{a5},$ <p>на основании значений переменных <b>a1, a2, a3, a4, a5</b>. Входные данные: Вещественные числа <b>a1, a2, a3, a4, a5</b>, каждое не меньше 0, но не превосходит <math>10^5</math>. Выходные данные: Одно вещественное число <math>s</math>. Результат необходимо округлить до тысячных. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
4, 17	<p>Требуется вычислить значения двух выражений <math>s</math> и <math>k</math>:</p> $s = \frac{a1 \times \sqrt{a2 - a5}}{a3}, k = \sqrt{\frac{a3}{a1}} \times (a2)^2,$ <p>на основании значений переменных <b>a1, a2, a3, a4, a5</b>.  <i>Входные данные:</i> вещественные числа <b>a1, a2, a3, a4, a5</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <math>s</math> и <math>k</math>.  Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
5, 18	<p>Требуется вычислить значения двух выражений <math>s</math> и <math>k</math>:</p> $s = \frac{\sqrt{a2 - a1 + a3}}{a3} + \frac{a4}{100}, k = \sqrt{\frac{a3 + a4}{a1 - a3}} \times (a2 - a5)^2,$ <p>на основании значений переменных <math>a1, a2, a3, a4, a5</math>.  <i>Входные данные:</i>  вещественные числа <b>a1, a2, a3, a4, a5</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <math>s</math> и <math>k</math>.  Результаты необходимо округлить до тысячных. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
6, 19	<p>Требуется вычислить значения двух выражений <math>s</math> и <math>k</math>:</p>

	$s = \frac{\sqrt{a2-a1}}{a3-a5} + \frac{a1}{a3}, k = \sqrt{\frac{a3+a4}{3.14-a3}} \times \frac{1}{(a2-a5)^2},$ <p>на основании значений переменных <b>a1, a2, a3, a4, a5</b>.</p> <p><i>Входные данные:</i> вещественные числа <b>a1, a2, a3, a4, a5</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.</p> <p><i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.</p> <p>Результаты необходимо округлить до десятитысячных. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
7, 20	<p>Требуется вычислить значения двух выражений <b>f</b> и <b>g</b>:</p> $f = \frac{1}{\sqrt{apo} - \sqrt{ew}}, \quad g = \sqrt{\frac{boo + \sqrt{ddd - boo}}{css}},$ <p>на основании значений переменных <b>apo, boo, css, ddd, ew</b>. <i>Входные данные:</i> вещественные числа <b>apo, boo, css, ddd, ew</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.</p> <p><i>Выходные данные:</i> Вещественные числа <b>f</b> и <b>g</b>.</p> <p>Результаты необходимо округлить до тысячных. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
8, 21	<p>Требуется вычислить значения двух выражений <b>f</b> и <b>g</b>:</p> $f = \frac{1}{100} - \frac{1}{apo} - \frac{1}{boo^2}, \quad g = \frac{1}{css^2} + \frac{\sqrt{ew}}{ddd^3},$ <p>на основании значений переменных <b>apo, boo, css, ddd, ew</b>. <i>Входные данные:</i> вещественные числа <b>apo, boo, css, ddd, ew</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.</p> <p><i>Выходные данные:</i> Вещественные числа <b>f</b> и <b>g</b>.</p> <p>Результаты необходимо округлить до тысячных. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>

## Продолжение таблицы 2.2

9, 22	<p>Требуется вычислить значения двух выражений <b>s</b> и <b>k</b>:</p> $s = \frac{\sqrt{a-e}}{\sqrt{b}}, \quad k = \sqrt{\frac{c}{ b-d }},$ <p>на основании значений переменных <b>a, b, c, d, e</b>.  <i>Входные данные:</i> вещественные числа <b>a, b, c, d, e</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.  Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
10, 23	<p>Требуется вычислить значения двух выражений <b>s</b> и <b>k</b>:</p> $s = \frac{\sqrt{a-e}}{\sqrt{b-a+100}}, \quad k = \sqrt{\frac{c-e}{ b-2d }},$ <p>на основании значений переменных <b>a, b, c, d, e</b>.  <i>Входные данные:</i> вещественные числа <b>a, b, c, d, e</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.  Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
11, 24	<p>Требуется вычислить значения двух выражений <b>s</b> и <b>k</b>:</p> $s = \frac{\sqrt{b-e+a}}{\sqrt{d-1}}, \quad k = \sqrt{\frac{c-11b}{b-e+a}},$ <p>на основании значений переменных <b>a, b, c, d, e</b>.  <i>Входные данные:</i> вещественные числа <b>a, b, c, d, e</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.  Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>

## Продолжение таблицы 2.2

12, 25	<p>Требуется вычислить значения двух выражений <b>s</b> и <b>k</b>:</p> $s = \frac{a}{\sqrt{b-c}} + \frac{b}{\sqrt{15c-d}}, \quad k = \frac{a}{b} + \frac{c}{\sqrt{3d-e}},$ <p>на основании значений переменных <b>a, b, c, d, e</b>.  <i>Входные данные:</i> вещественные числа <b>a, b, c, d, e</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.          Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>
13, 26	<p>Требуется вычислить значения двух выражений <b>s</b> и <b>k</b>:</p> $s = \frac{1}{\sqrt{b-a}} + \frac{2}{\sqrt{c-b}} + \frac{3}{\sqrt{d-c}}, \quad k = \sqrt{a - \sqrt{b - \sqrt{c}}},$ <p>на основании значений переменных <b>a, b, c, d, e</b>.  <i>Входные данные:</i> вещественные числа <b>a, b, c, d, e</b>, каждое не меньше 0, но не превосходит <math>10^5</math>.  <i>Выходные данные:</i> Вещественные числа <b>s</b> и <b>k</b>.          Результаты необходимо округлить до сотых. Если вычислить выражение невозможно, программа выводит <b>ERROR</b>.</p>



### **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Укажите различие между `Console.Write()` и `Console.WriteLine()`.
2. Опишите метод `Console.ReadKey()`. Укажите параметры, возвращаемое значение, применение.
3. Укажите стандартные потоки ввода-вывода в консольном приложении .NET Framework.
4. Укажите различия между методами `Console.ReadLine()` и `Console.Read()`.
5. Программисту необходимо произвести конвертацию значения типа `System.String` в значение типа `float`. Напишите метод, который необходимо использовать.

## ЛАБОРАТОРНАЯ РАБОТА 3. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ В ПРОГРАММЕ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* изучить операторы, позволяющие организовывать непоследовательное выполнение программного кода.

Задачи лабораторной работы:

- научиться применять условный оператор `if .. else;`
- научиться применять операторы цикла `for`, `while`, `do ... while;`
- научиться применять операторы выбора `switch`.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

#### Условный оператор

В предыдущих лабораторных работах были рассмотрены вопросы программирования на языке C# с использованием только последовательного выполнения операторов программы.

В языке C# (как и в большинстве языков) можно вводить условия, циклы, ветвления.

Условный оператор в C# позволяет организовать условие типа «если ..., то ..., иначе ...».

Синтаксис условного оператора:

*if (условие )*

*оператор*

*[else if (условие)*

*оператор*

*...*

*else*

*оператор]*

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки). Условный оператор содержит только одно обязательное зарезервированное слово – `if`. Все остальные конструкции не яв-

ляются обязательными (в синтаксическом описании на это указывают скобки [ ]).

Пример использования условного оператора:

```
int a = 100, b = 13, c = 23, d = 0, e = 0;

if (a == 100) // Если a равно 100
{
    // Увеличиваем b на 1000
    b += 1000;
}
else if (a < 100) // иначе если a меньше 100
{
    // изменяем значение d
    d = b + c;
    // и изменяем значение e
    e = a + b;
}
else // иначе (остальные варианты не предусмотренные первыми двумя условиями)
    // edtkbxbdfv a на 1
    a++;

// продолжается последовательное выполнение программы
Console.WriteLine("{0}", a);
```

В представленном примере сначала проверяется, является ли значение переменной *a* равным 100.

1. Если «Да», то выполняется единственный оператор (увеличение *b* на 1000). На этом выполнение всего условного оператора завершено и программа переходит к выполнению операторов, следующих за ним, то есть к выводу значения *a*.

2. Если первое условие не выполнилось, проверяется второе – является ли *a* меньше 100 (конструкция *else if*). Если это утверждение истинно, то выполняется составной оператор, состоящий из двух – изменение значений переменных *d* и *e*. Затем осуществляется переход к выводу переменной *a*.

3. Если не выполнилось ни одно условие с оператором *if*, то выполняется оператор, указанный после зарезервированного слова *else*. В данном случае это несоставной оператор (*a* увеличивается на 1), поэтому фигурные скобки можно не писать

Во всей этой конструкции только оператор *if* является обязательным. Конструкций *else if* может быть любое количество, а *if* и *else* – только по одному.

Еще один пример условного оператора:

```

int a = 100;

if (a >= 100)
{
    Console.WriteLine("Значение переменной a больше или равно 100");
}

```

В данном случае вывод в консоль выполнится, только если *a* больше либо равно 100. Иначе условный оператор ничего не выведет. Фигурные скобки в данном примере можно опустить.

Обратите внимание, что для проверки на равенство используется оператор `==`, а не `=`. Знак «равно» используется в С# для присваивания значений. Следует понимать, что условное выражение, стоящее в конструкции `if` должно возвращать булево значение.

Например, следующий условный оператор всегда будет выполняться:

```

if (true)
    Console.WriteLine("Всегда выводится");
else
    Console.WriteLine("Никогда не выводится");

```

## Оператор выбора

Синтаксис оператора:

```

switch (перем)
{
    case константа1 :
        оператор_1;
        break;
    case константа2 :
        оператор_2;
        break;
    ...
    default :
        оператор_n;
        break;
}

```

В данном операторе осуществляется выбор оператора (который может быть составным) в зависимости от значения переменной *перем*. Если *перем* равна значению *константа1*, то выполнится *оператор\_1*, если *константа2* – *оператор\_2*, и т.д. Если ни одно значение, указанное в каком-либо операторе *case*, не совпало со значением *перем*, то выполнится оператор, указанный в секции *default*. Внутри каждой секции *case* следует указать оператор **break**, который предотвращает проверку других условий после выполнения данного оператора. Следует обратить внимание, что в секциях *case* следует использовать только константы (переменные не допускаются).

Пример использование оператора *switch ... case*:

```
// Организация ответа на действие пользователя:
string text = "1 - Вывод группы \n" +
    "2 - Вывод фамилии преподавателя \n" +
    "3 - Вывод названия предмета \n" +
    "4 - Вывод приветствия \n";

Console.Write(text + "Введите команду > ");
int result = Convert.ToInt32(Console.ReadLine());

switch (result)
{
    case 1: { Console.WriteLine("ИСТБ - 101"); break; }
    case 2: Console.WriteLine("Николаев Евгений Иванович"); break;
    case 3: { Console.WriteLine("Технологии программирования"); break; }
    case 4: Console.WriteLine("Привет !"); break;
    default: Console.WriteLine("Недопустимый вариант !!!"); break;
}
```

Изучите представленный пример самостоятельно.

## Цикл **for**

Циклы позволяют выполнять определенную последовательность операторов до тех пор, пока выполняется некоторое условие. Каждый «круг» выполнения этого блока называется итерацией.

*for* (*инициализатор*; *условие*; *итератор*)  
*оператор*

*инициализатор* – выражение, выполняемое перед первой итерацией цикла.

*условие* – выражение булевого типа, которое проверяется перед каждой итерацией. Если оно истинно то итерация выполняется, иначе – цикл завершается.

*итератор* – выражение, вычисляемое после каждой итерации.

Пример применения оператора цикла for:

```
/*
 * Вычислить сумму чисел от 1 до 1000
 */

// Объявляем переменную, которая хранит значение суммы чисел
System.Int32 Sum = 0;
// Объявляем счетчик цикла
int i;

for (i = 1; i <= 1000; i++)
    Sum += i;

Console.WriteLine("Итого: {0}", Sum);
```

Изучите код самостоятельно (создайте такую программу в VS). Еще один пример для самостоятельного изучения:

```
/*
 * вычислить сумму всех чисел, делящихся на 3 без остатка
 * и находящихся в диапазоне от 1 до 1000000
 */

long Sum = 0;
int i;

for (i = 1; i <= 1000000; i++)
    if (i % 3 == 0)
        Sum += i;
```

Пример, отображающий возможность применения итератора:

```

/*
 * вычислить сумму всех чисел, делящихся на 3 без остатка
 * и находящихся в диапазоне от 1 до 1000000
 */

long Sum = 0;
int i;

for (i = 0; i <= 1000000; i += 3)
    Sum += i;

```

### Цикл while

Цикл while похож на for тем, что является конструкцией с предварительной проверкой условия продолжения цикла. Но синтаксис цикла while более лаконичен:

```

while (условие)
    оператор

```

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки).

Следует понимать, что while используется для заранее неизвестного количества повторных выполнений операторов.

Пример выполнения цикла:

```

/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = false;
int chislo;

while (!Stop)
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        Stop = true;
}

```

Оператор **break** уже встречался в конструкции case оператора switch для выхода из case. Но break часто применяется внутри циклов for, while и do ... while.

Данный оператор прерывает выполнение цикла и передает управление оператору, следующему за циклом.

Следует обратить внимание, что цикл является бесконечным (логическая переменная *Stop* всегда является *true*), но выполнение цикла все равно прервется, если пользователь введет 0.

Оператор *continue* также применяется внутри цикла – он останавливает выполнение текущей итерации и немедленно переходит к выполнению следующей итерации.

### Цикл **do ... while**

Цикл *do ... while* полностью повторяет функциональные возможности *while*, но предполагает проверку условия окончания цикла после выполнения тела цикла. То есть блок операторов тела цикла всегда выполнится хотя бы один раз.

Синтаксис оператора:

```
do
{
    оператор (операторы)
}
while (условие);
```

Пример использования цикла:

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */
bool Stop = false;
int chislo;

do
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
    {
        Stop = true;
    }
} while (!Stop);
```



## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Учебная задача

Пример. Требуется написать программу для вычисления значения выражения:

$$Z = \frac{1 \cdot 3 \cdot X^2}{2!} - \frac{3 \cdot 5 \cdot Y^4}{4!} + \frac{5 \cdot 7 \cdot X^6}{6!} - \frac{7 \cdot 9 \cdot Y^8}{8!} + \dots \quad (3.1)$$

Следует реализовать программу таким образом, чтобы все необходимые для вычисления значения выражения исходные значения считывались из входного файла, а результат записывался в выходной файл. Следует учитывать, что необходимо реализовать все три типа цикла при решении задачи, то есть во входном файле необходимо предусмотреть признак, указывающий с использованием какого типа цикла необходимо вычислять выражение.

Решение. Проанализировав выражение (3.1) можно сделать вывод о том, какие именно значения понадобятся, чтобы вычислить выражение для  $Z$ : нужны значения  $X$ ,  $Y$ ,  $N$  и  $t$ . Значение  $N$  – это количество слагаемых, которые будут складываться;  $t$  – это тип используемого цикла (например, 0 – цикл for, 1 – цикл while, 2 – цикл do...while). Вид входного файла представлен на рисунке 3.1.

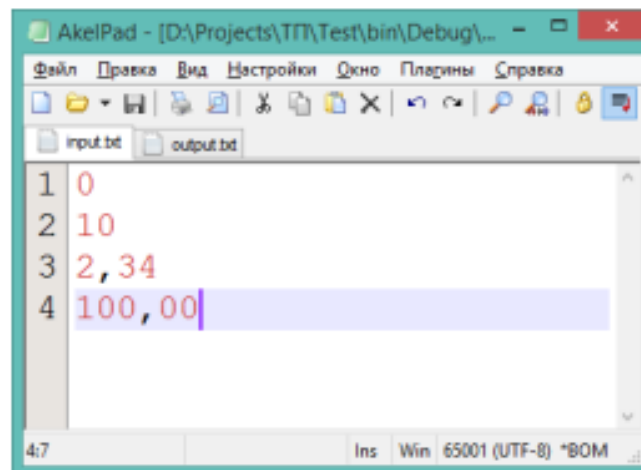


Рисунок 3.1 – Входной файл для решения задачи

Первая строка содержит значение для  $t$ , вторая –  $N$ , третья и четвертая – для  $X$  и  $Y$  соответственно.

В листинге на рисунке 3.2 представлено решение задачи.

```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             TextWriter save_out = Console.Out;
11             TextReader save_in = Console.In;
12             var new_out = new StreamWriter(@"output.txt");
13             var new_in = new StreamReader(@"input.txt");
14             Console.SetOut(new_out);
15             Console.SetIn(new_in);
16
17             int t = 0, N = 1;
18             double X = 0, Y = 0, Z = 0;
19             t = Convert.ToInt32(Console.ReadLine());
20             N = Convert.ToInt32(Console.ReadLine());
21             X = Convert.ToDouble(Console.ReadLine());
22             Y = Convert.ToDouble(Console.ReadLine());
23
24             int i = 1, step = 1;
25             double znam = 1, chisl;
26

```

```

27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

```

if (t == 0)
{
    for (i = 1; i <= N; i++)
    {
        step = i * 2;
        znam *= (step - 1) * step;
        if (i % 2 == 0)
            chisl = -Math.Pow(Y, step);
        else
            chisl = Math.Pow(X, step);
        Z += (step - 1) * (step + 1) * chisl / znam;
    }
}

if (t == 1)
{
    i = 1;
    while (i <= N)
    {
        step = i * 2;
        znam *= (step - 1) * step;
        if (i % 2 == 0)
            chisl = -Math.Pow(Y, step);
        else
            chisl = Math.Pow(X, step);
        Z += (step - 1) * (step + 1) * chisl / znam;
        i++;
    }
}

if (t == 2)
{
    i = 1;
    do
    {
        step = i * 2;
        znam *= (step - 1) * step;
        if (i % 2 == 0)
            chisl = -Math.Pow(Y, step);
        else
            chisl = Math.Pow(X, step);
        Z += (step - 1) * (step + 1) * chisl / znam;
        i++;
    } while (i <= N);
}

```

```

71     }
72
73     Console.WriteLine(String.Format("{0:0.0000000}", Z));
74
75
76     Console.SetOut(save_out); new_out.Close();
77     Console.SetIn(save_in); new_in.Close();
78 }
79
80 }

```

Рисунок 3.2 – Листинг решения учебной задачи

Следует обратить внимание, что независимо от выбранного цикла выражение вычисляется и в файл output.txt записывается одно и то же значение.

### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если требуется, оператор выбора. В каждом варианте все входные значения считываются из файла, а результат записывается в файл.

Таблица 3.1

#### Индивидуальные задания

Вариант	Выражение для вычисления
1, 14	$j = -\frac{\sin^3(Y)}{1 \cdot 3} + \frac{\sin^5(X^2)}{3 \cdot 5} - \frac{\sin^7(Y^3)}{5 \cdot 7} + \frac{\sin^9(X^4)}{7 \cdot 9} - \dots$
2, 15	$Z = \frac{1}{1 \cdot 3 \cdot 5} - \frac{X}{2 \cdot 4 \cdot 6} + \frac{Y^2}{3 \cdot 5 \cdot 7} - \frac{X^3}{4 \cdot 6 \cdot 8} + \frac{Y^4}{5 \cdot 7 \cdot 9} - \dots$
3, 16	$Z = \frac{X^2}{1 \cdot 3} - \frac{Y^4}{3 \cdot 5} + \frac{X^6}{5 \cdot 7} - \frac{Y^8}{7 \cdot 9} + \dots$
4, 17	$Z = \frac{Y^2 \cdot X}{2} - \frac{Y^4 \cdot X^3}{4} + \frac{Y^6 \cdot X^5}{6} - \frac{Y^8 \cdot X^7}{8} + \dots$
5, 18	$A = -\frac{\sin(X) \cdot \lg(Y)}{1!} + \frac{\ln(X^3) \cdot \cos(Y^2)}{3!} - \frac{\sin(X^5) \cdot \lg(Y^3)}{5!} + \frac{\ln(X^7) \cdot \cos(Y^4)}{7!} - \dots$
6, 19	$s = -\frac{Y \cdot X^2}{1 \cdot 3} + \frac{X \cdot Y^3}{2 \cdot 4} - \frac{Y \cdot X^4}{3 \cdot 5} + \frac{X \cdot Y^5}{4 \cdot 6} - \dots$

Продолжение таблицы 3.1

7, 20	$Z = 1 - \frac{\sin(X^2)}{2} + \frac{\cos(Y^3)}{3} - \frac{\sin(X^4)}{4} + \frac{\cos(Y^5)}{5} - \dots$
8, 21	$Z = \frac{1}{1 \cdot 3} - \frac{X}{2 \cdot 4} + \frac{X^2}{3 \cdot 5} - \frac{X^3}{4 \cdot 6} + \dots$
9, 22	$Z = \frac{1}{1!} - \frac{X}{2!} + \frac{Y^2}{3!} - \frac{X^3}{4!} + \frac{Y^4}{5!} - \dots$
10, 23	$Z = \frac{\cos(X) +  Y }{1!} - \frac{\cos^2(Y) +  X }{2!} + \frac{\cos^3(X) +  Y }{3!} - \frac{\cos^4(Y) +  X }{4!} + \dots$
11, 24	$Z = \frac{1}{1 \cdot 3} - \frac{X^2 \cdot Y}{2 \cdot 4} + \frac{X^4 \cdot Y^2}{3 \cdot 5} - \frac{X^6 \cdot Y^3}{4 \cdot 6} + \dots$
12, 25	$Z = -\frac{\sin^3(Y)}{1 \cdot 3} + \frac{\cos^5(X^3)}{3 \cdot 5} - \frac{\sin^7(Y^5)}{5 \cdot 7} + \frac{\cos^9(X^7)}{7 \cdot 9} - \dots$
13, 26	$Z = \frac{X^2 \cdot 1}{3!} - \frac{Y^4 \cdot 2}{5!} + \frac{X^6 \cdot 3}{7!} - \frac{Y^8 \cdot 4}{9!} + \dots$

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Опишите, каким образом оформляется комментарий в языке C#.
2. Какое ключевое (зарезервированное) слово в условном операторе является обязательным?
3. Дан фрагмент кода:

```
double x = 2, y = 3, z = 4, res;
Boolean yes = true;
if (!yes)
{
    res = Math.Pow(x, y) * z;
}
else
{
    res = Math.Pow(z, x) * y;
}
```

Чему равно значение переменной **res** после выполнения данного фрагмента?

## ЛАБОРАТОРНАЯ РАБОТА 4. ОДНОМЕРНЫЕ МАССИВЫ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* изучить типы и принципы работы с одномерными массивами.

Задачи лабораторной работы:

- научиться инициализировать одномерные массивы;
- научиться реализовывать простейшие алгоритмы на одномерных массивах;
- научиться вычислять простейшие агрегированные показатели на одномерных массивах.

### ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Для работы с набором значений одного типа используются массивы.

*Массив* – это структура данных, содержащая множество элементов одного и того же типа.

Массив определяется типом элементов, содержащимся в нем, за которым следуют квадратные скобки и имя переменной.

Примеры объявления массивов:

```
// Объявление массива целочисленных элементов
int[] mas_int;

// Объявление массива элементов типа double
double[] mas_d;

// Объявление массива элементов типа string
string[] MassivStr;
```

При объявлении массива не указывается его размер. Для выделения памяти для элементов массива необходимо использовать следующий синтаксис (для массивов, объявленных в предыдущем примере):

```
// Спецификация размера массивов
mas_d = new double[5];
mas_int = new Int32[100];
MassivStr = new string[100];
```

После данной инициализации можно обращаться к элементам массивов.

После такой инициализации нельзя изменять размер массива.

Объявить и инициализировать массив можно в одной строке:

```
float[] keof = new float[1000];
```

При инициализации можно также присвоить значения элементам массива (все три способа присвоения элементов массива эквивалентны):

```
int[] m1 = new int[5] {1, 23, 2, 100, 7};

int[] m2 = new int[] {23, 12, 100, 101, 0};

int[] m3 = {0, 1, 2, 56, 90};
```

Там, где программист не указал явно размер массива, компилятор определит его самостоятельно. Очевидно, что задать таким образом большое количество элементов неудобно. Для доступа к элементам массивов используется индексатор.

После того как массив объявлен и инициализирован, можно производить доступ к элементам массива для чтения и записи через индексатор.

Индексатор – это целочисленный номер элемента массива. Индексатор начинается с 0 (для первого элемента массива) и заканчивается числом равным количеству элементов в массиве минус 1.

Например, определим массив и попробуем обратиться к его элементам:

```
// Объявление и инициализация массива
int[] mas = new int[4];

// Установка значений элементов массива
mas[0] = 1000; // для первого элемента
mas[1] = 45; // для второго элемента

// Считывание значений
mas[2] = mas[0]; // считываем значение первого и присваиваем его третьему элементу
int val = mas[1];
int a = mas[3]; // возникнет ОШИБКА! Т.к. четвертый элемент не инициализирован
int oo = mas[4]; // возникнет ОШИБКА! Т.к. пятого элемента в массиве нет
```

Часто (когда размер массива большой) используют циклы для обращения к элементам массива:



```

/*
 * В цикле последовательно запрашиваются у пользователя
 * значения для инициализации элементов массива
 */
for (int i = 0; i < massiv.Length; i++)
{
    Console.Write("Введите значение {0}-го элемента > ", i+1);
    massiv[i] = Convert.ToInt32(Console.ReadLine());
}

```

Для больших массивов есть смысл сгенерировать значения элементов массива случайным образом (используя генератор случайных чисел):

```

int[] Pro = new int[1000];

// Объявление переменной класса Random
Random rnd = new Random(1);

int i = 0;

while (i < Pro.Length)
{
    // Вызов метода Next класса Random
    Pro[i] = rnd.Next();
    i++;
}

// Вывод элементов массива в обратном порядке
for (i = Pro.Length-1; i >= 0; i--)
    Console.WriteLine("Элемент {0} равен {1}", i, Pro[i]);

```

Внимательно изучите представленный код.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Учебная задача

Условие. Дана последовательность  $N$  целых чисел. Вычислить среднее арифметическое этих чисел. Найти среди исходных  $N$  чисел все числа, которые не меньше среднего арифметического. Для выполнения задачи необходимо написать программу-генератор исходных массивов.

Решение. Перед непосредственно кодированием задачи необходимо проанализировать условие и разбить задачу на подзадачи. Особое внимание нужно уделить написанию программы-



генератора исходных данных, то есть сначала необходимо написать программу, которая позволит автоматически создавать файлы, которые будут содержать исходные данные для решения учебной задачи. Код программы-генератора массивов представлен на рисунке 4.1.

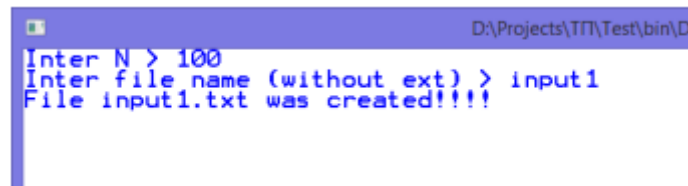
```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             int N;
11             String FileName;
12
13             Console.Write("Enter N > ");
14             N = Convert.ToInt32(Console.ReadLine());
15             Console.Write("Enter file name (without ext) > ");
16             FileName = Console.ReadLine();
17             FileName += ".txt";
18
19             TextWriter save_out = Console.Out;
20             var new_out = new StreamWriter(FileName);
21             Console.SetOut(new_out);
22
23             Console.WriteLine(N);
24
25             Random r = new Random(DateTime.Now.Millisecond);
26             int x = 0;
27             for (int i = 0; i < N; i++)
28             {
29                 x = r.Next(1000);
30                 Console.Write(x + " ");
31             }
32
33             Console.SetOut(save_out); new_out.Close();
34             Console.WriteLine("File " + FileName + " was created!!!!");
35             Console.ReadKey();
36         }
37     }
38 }

```

Рисунок 4.1 – Листинг программы для генерации исходных файлов

Вывод данной программы в консоли представлен на рисунке 4.2.



```

D:\Projects\TIT\Test\bin\De
Inter N > 100
Inter file name (without ext) > input1
File input1.txt was created!!!!

```

Рисунок 4.2 – Вывод в консоль программы для генерации исходных файлов

С использованием данного генератора можно сформировать следующие файлы (рисунок 4.3), содержащие массивы данных для обработки и решения задач по тематике одномерных массивов.

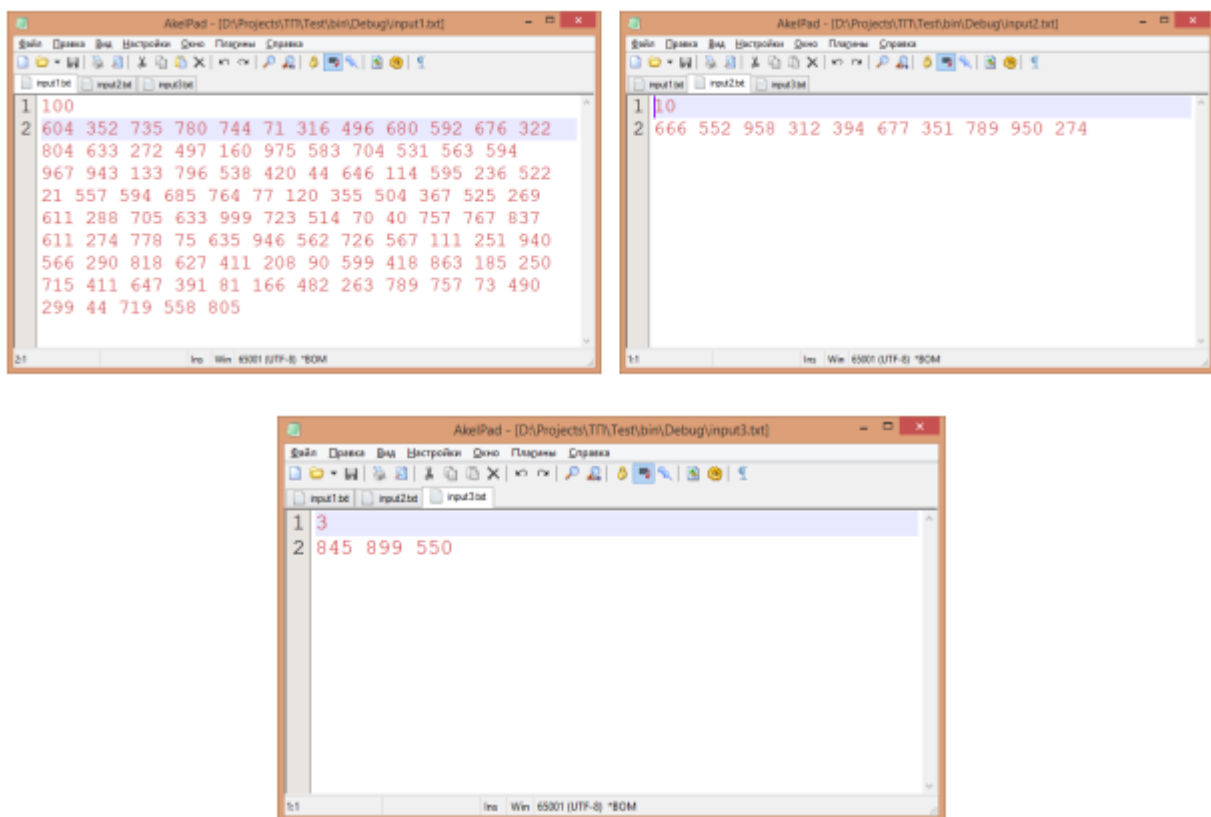


Рисунок 4.3 – Файлы, сгенерированные программой-генератором

Понятно, что для решения учебной задачи можно сформировать исходный файл вручную, но для больших массивов это будет сделать проблематично.

Теперь напишем программу, которая на основании данных, содержащихся во входном файле, вычисляет среднее арифметическое всех чисел и выводит все исходные числа, которые не меньше среднего арифметического (рисунок 4.4).

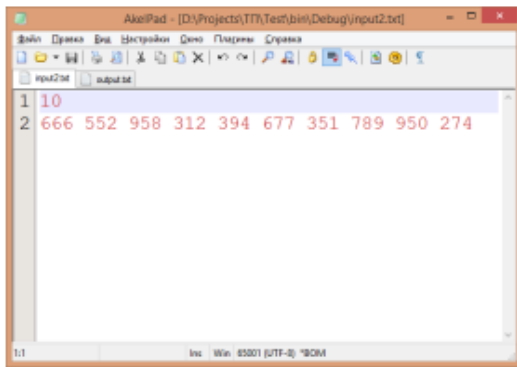
```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             TextWriter save_out = Console.Out;
11             TextReader save_in = Console.In;
12             var new_out = new StreamWriter(@"output.txt");
13             var new_in = new StreamReader(@"input2.txt");
14             Console.SetOut(new_out);
15             Console.SetIn(new_in);
16
17             // Чтение данных из файла и инициализация массива
18             int N = Convert.ToInt32(Console.ReadLine());
19             String str_all = Console.ReadLine();
20             string[] str_elem = str_all.Split(' ');
21
22             int[] mas = new int[N];
23             for (int i = 0; i < N; i++)
24             {
25                 mas[i] = Convert.ToInt32(str_elem[i]);
26             }
27
28             // Вычисление среднего арифметического
29             int s = 0;
30             float sa = 0;
31             for (int i = 0; i < N; i++)
32             {
33                 s += mas[i];
34             }
35             sa = 1.0f * s / N;
36
37             // Вывод среднего арифметического и требуемых значений
38             Console.WriteLine(string.Format("{0:0.000000}", sa));
39             for (int i = 0; i < N; i++)
40             {
41                 if (mas[i] >= sa)
42                     Console.Write(mas[i] + " ");
43             }
44
45             Console.SetOut(save_out); new_out.Close();
46             Console.SetIn(save_in); new_in.Close();
47         }
48     }
49 }

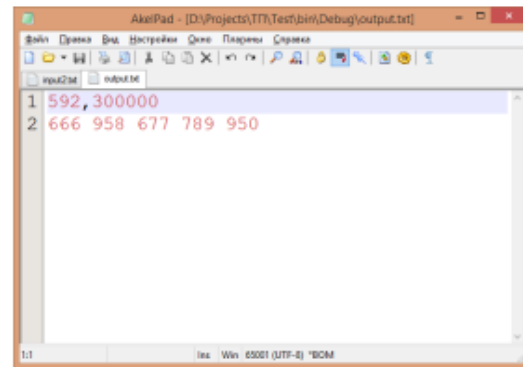
```

Рисунок 4.4 – Вычисление среднего арифметического и вывод требуемых чисел

В результате работы программы для файла input2.txt будет сформирован следующий output.txt (рисунок 4.5):



а)



б)

Рисунок 4.5 – Файлы учебной задачи: а) входной; б) выходной

### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Для инициализации исходной матрицы необходимо использовать программу-генератор, разработанную в ходе выполнения учебной задачи. Целесообразно использовать ограничение на генерируемые числа, например, ограничить значения случайно генерируемых чисел величиной 100, 10 и т.д., чтобы вывод программы был более читабельным. Алгоритм решения должен работать для массивов любой величины, поэтому необходимо протестировать программу на различных входных данных:

Таблица 4.1

#### Индивидуальные задания

Вариант	Задание
1, 14	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от 0 до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. количество чисел, больших среднего арифметического исходных чисел;</li> <li>2. сумма всех чисел, меньших среднего арифметического;</li> <li>3. максимальное число.</li> </ol>

Продолжение таблицы 4.1

2, 15	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от 0 до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. произведение чисел, меньших 50 (если таких чисел нет, то выводится 0);</li> <li>2. сумма всех чисел, больших среднего арифметического;</li> <li>3. минимальное число</li> </ol>
3, 16	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от 0 до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. произведение чисел, меньших 50 (если таких чисел нет, то выводится 0);</li> <li>2. все числа, большие среднего арифметического через пробел</li> </ol>
4, 17	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от 0 до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. максимальный элемент;</li> <li>2. среднее арифметическое всех элементов;</li> <li>3. модифицированный массив, в котором все элементы исходного, меньшие среднего арифметического, заменены на 0.</li> </ol>

Продолжение таблицы 4.1

5, 18	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. среднее арифметическое всех элементов;</li> <li>2. максимальный элемент;</li> <li>3. модифицированный массив, в котором все элементы не превышающие по модулю 1000 заменены на 0.</li> </ol>
6, 19	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> целых чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. максимальный элемент среди всех кратных 5;</li> <li>2. минимальный элемент среди всех кратных 3;</li> <li>3. сумму элементов, которые делятся на 10.</li> </ol>
7, 20	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> целых чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. сумму элементов, кратных 7;</li> <li>2. произведение элементов, кратных 5;</li> <li>3. список элементов кратных 10.</li> </ol>

Продолжение таблицы 4.1

8, 21	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. максимальный элемент, не превышающий 1000;</li> <li>2. минимальный по модулю элемент;</li> <li>3. список элементов, больших 5000 и меньших 7000</li> </ol>
9, 22	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. максимальный отрицательный элемент (если таких нет, то вывести No);</li> <li>2. минимальный положительный элемент (если таких нет, то вывести No);</li> <li>3. список элементов, превышающих по модулю 10000 (если таких нет, то вывести No).</li> </ol>
10, 23	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> вещественных чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. максимальный элемент;</li> <li>2. список элементов, отстающих от максимального не более чем на 1000 (если таких нет, то вывести No);</li> <li>3. модифицированный массив, в котором все элементы, большие по модулю 5000, заменены на «+», а меньшие на «-».</li> </ol>

Продолжение таблицы 4.1

11, 24	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> целых чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. количество элементов, кратных 10;</li> <li>2. произведение элементов, кратных 100;</li> <li>3. модифицированный массив, в котором все элементы кратные 5 заменены на 5, остальные – на <math>x</math>.</li> </ol>
12, 25	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> целых чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. количество элементов, больших 1000;</li> <li>2. минимальный элемент, кратный 7;</li> <li>3. список элементов, лежащих между 0 и 1000.</li> </ol>
13, 26	<p>Во входном файле содержится две строки: первая содержит одно целое число <math>N</math> (количество чисел во второй строке), вторая строка содержит <math>N</math> целых чисел (каждое от <math>-10^5</math> до <math>10^5</math>).</p> <p>В результате работы программы должен быть сформирован выходной файл, который содержит следующие значения:</p> <ol style="list-style-type: none"> <li>1. среднее арифметическое отрицательных чисел;</li> <li>2. максимальный элемент;</li> <li>3. список элементов, кратных 50.</li> </ol>



## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение массива в языке C#.
2. Укажите строки, в которых верно выполнено объявление массива:

```
int[] m;

System.String[] names = { "Piro", "Nadi", "Deny"};

float[,] points;

System.Int64[] big;

char[] c = { 'z', 'y', 'x' };

int mas[];

float[3] points = { 1.0, 2.0, 3.0 };

double coord;

string[] str = new string[10];
```

3. Что будет выведено в консоль в результате работы программы?

```
static void Main(string[] args)
{
    Random r = new Random();
    int[] z = new int[150];
    for (int i = 0; i < z.Length; i++)
    {
        z[i] = r.Next();
    }
    int res = 0;
    for (int i = 0; i < z.Length; i++)
    {
        res += z[i];
    }

    Console.Write(res);
    Console.ReadKey();
}
```

## ЛАБОРАТОРНАЯ РАБОТА 5. МНОГОМЕРНЫЕ МАССИВЫ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* изучить типы и принципы работы с многомерными массивами.

Задачи лабораторной работы:

- научиться инициализировать двумерные массивы;
- научиться реализовывать простейшие алгоритмы на многомерных массивах;
- научиться вычислять простейшие агрегированные показатели на многомерных массивах.

### ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

Обычные массивы (одномерные) индексируются одним числом. Многомерные массивы индексируются несколькими индексами. Очевидно, что двумерный массив можно отождествлять с матрицей, трехмерный – с кубом.

Принципы объявления, инициализации и обращения к элементам остаются теми же, что и для одномерных массивов:

```
// Объявляем и инициализируем двумерный массив
// - матрицу целых размером 3x3
int[,] matrix = new int[3, 3];

// Присваиваем значение диагональным элементам
matrix[0, 0] = 1;
matrix[1, 1] = 120;
matrix[2, 2] = 8;

// Объявляем матрицу с одновременным присвоением
// значений элементов
int[,] mm = {
    {0, 23, 100},
    {6, 12, 23},
    {0, 0, 5}
};

// Объявляем трехмерный массив
double[, ,] org = new double[100, 100, 50];

// Присваиваем значение 34 элементу с индексами (0, 0, 0)
org[0, 0, 0] = 34;
```

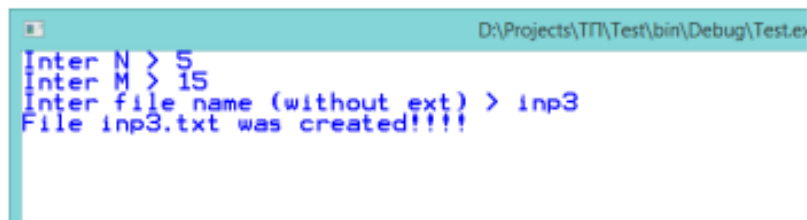
Как и в случае одномерных массивов, для работы с многомерными применяются циклы.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Условие. Дана исходная матрица размером  $M \times N$  из целых чисел, принадлежащих отрезку  $[-10^3, 10^3]$ . В выходной файл необходимо вывести следующие значения:

- исходную матрицу;
- максимальный элемент для каждой строки;
- среднее арифметическое матрицы и модифицированную матрицу, в которой все элементы, большие среднего арифметического элементов, заменены на +, остальные на -.

Решение. По аналогии с одномерными массивами перед выполнением основного задания необходимо написать программу, которая будет генерировать входные файлы для задачи. Данная программа-генератор должна генерировать матрицы различного размера и типа (данный пункт реализуйте самостоятельно по аналогии с рассмотренной реализацией генератора). На рисунке 5.1 представлено окно программы-генератора, а на рисунке 5.2 представлены примеры входных файлов, полученных при использовании разработанного генератора.



```
D:\Projects\TTP\Test\bin\Debug\Tester
Enter N > 5
Enter M > 15
Enter file name (without ext) > inp3
File inp3.txt was created!!!!
```

Рисунок 5.1 – Возможный интерфейс программы-генератора

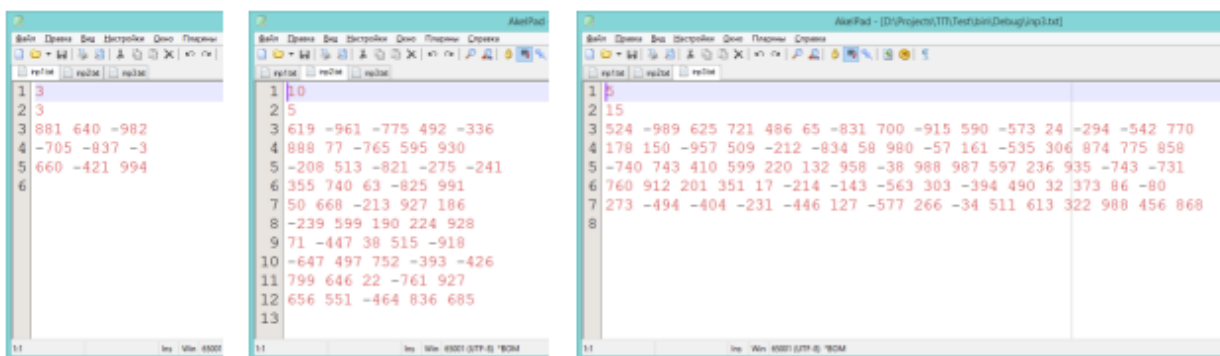


Рисунок 5.2 – Исходные файлы, полученные автоматически с использованием программы-генератора

После разработки программы-генератора можно решить учебную задачу и протестировать ее на различных наборах данных. Исходный код показан на рисунке 5.3.

```
1  using System;|
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             TextWriter save_out = Console.Out;
11             TextReader save_in = Console.In;
12             var new_out = new StreamWriter(@"output.txt");
13             var new_in = new StreamReader(@"inp2.txt");
14             Console.SetOut(new_out);
15             Console.SetIn(new_in);
16
17             // Чтение данных из файла и инициализация массива
18             // и сразу выводим исходную
19             int N = Convert.ToInt32(Console.ReadLine());
20             int M = Convert.ToInt32(Console.ReadLine());
21
22             Console.WriteLine("*** Input matrix ***");
23         }
24     }
25 }
```

```

24 int[,] mas = new int[N, M];
25 for (int i = 0; i < N; i++)
26 {
27     String str_all = Console.ReadLine();
28     string[] str_elem = str_all.Split(' ');
29     for (int j = 0; j < M; j++)
30     {
31         mas[i, j] = Convert.ToInt32(str_elem[j]);
32         Console.Write(mas[i, j] + " ");
33     }
34     Console.WriteLine();
35 }
36
37 // Ищем и выводим максимальные элементы для строк
38 // Находим среднее арифметическое
39 int max = -1001;
40 float sa = 0;
41 Console.WriteLine("*** Max row elements ***");
42 for (int i = 0; i < N; i++)
43 {
44     max = -1001;
45     for (int j = 0; j < M; j++)
46     {
47         if (mas[i, j] > max)
48             max = mas[i, j];
49         sa += mas[i, j];
50     }
51     Console.WriteLine(max);
52 }
53
54 sa = sa / (M*N);
56 // Вывод модифицированной матрицы
57 Console.WriteLine("*** Modified matrix ***");
58 Console.WriteLine(string.Format("Avg = {0:0.000}", sa));
59 for (int i = 0; i < N; i++)
60 {
61     for (int j = 0; j < M; j++)
62     {
63         if (mas[i, j] > sa)
64             Console.Write("+ ");
65         else
66             Console.Write("- ");
67     }
68     Console.WriteLine();
69 }
70
71 Console.SetOut(save_out); new_out.Close();
72 Console.SetIn(save_in); new_in.Close();
73 }
74 }
75 }

```

Рисунок 5.3 – Исходный код решения учебной задачи

Входной и выходной файл показаны на рисунке 5.4.

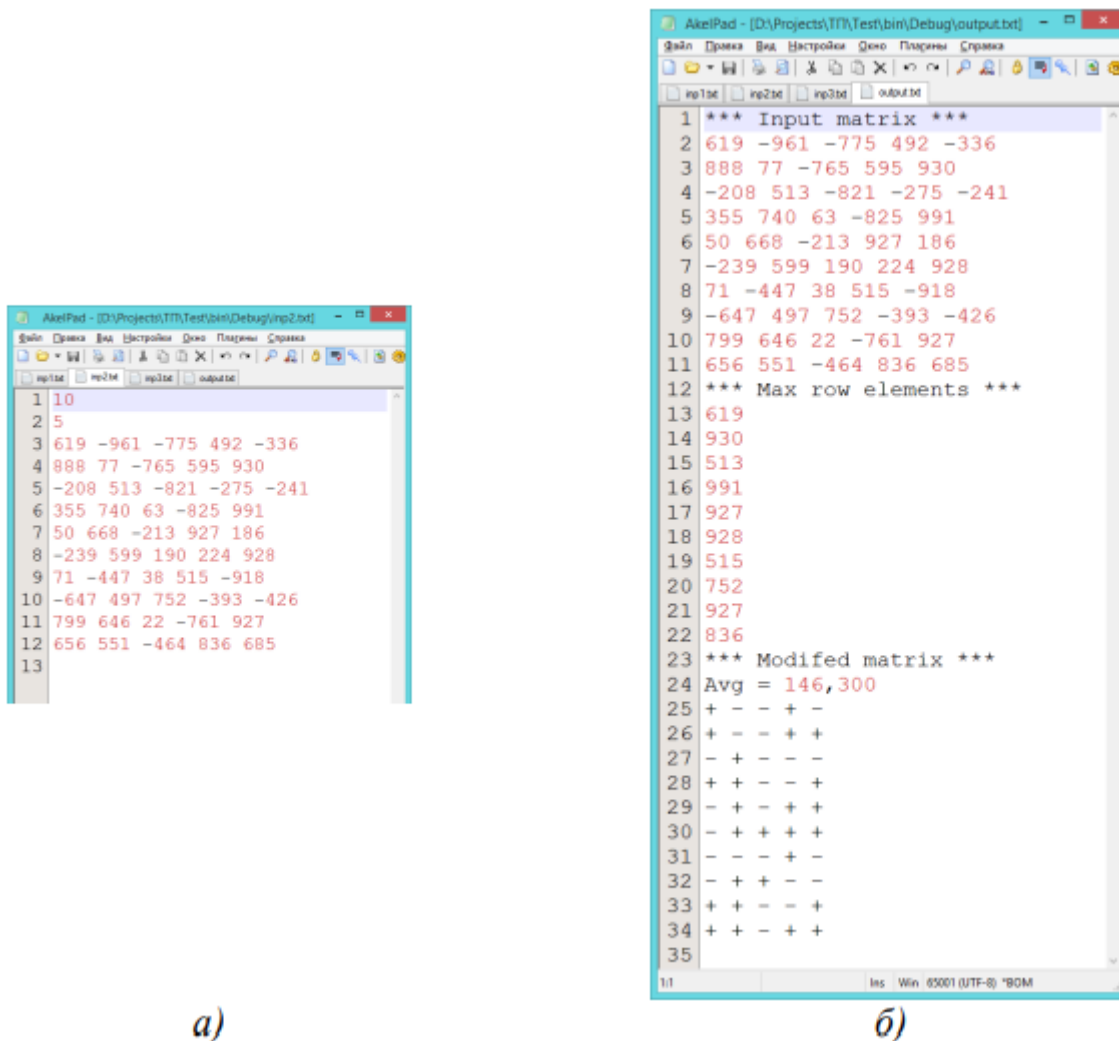


Рисунок 5.4 – Файлы, используемые программой в качестве источников ввода/вывода: а) входной файл; б) выходной файл

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Для инициализации исходной матрицы необходимо использовать генератор случайных чисел. Следует использовать ограничение на генерируемые числа, например, ограничить значения случайно генерируемых чисел величиной 100, 10 и т.д. (ограничение определено в условии), чтобы вывод программы был более читабельным. Алгоритм решения должен работать для массивов любой величины, поэтому размер исходного массива считывается из файла.

Таблица 5.1

## Индивидуальные задания

Вариант	Задание
1, 14	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• среднее арифметическое для каждой строки;</li> <li>• модифицированную матрицу, в которой все четные элементы заменены на 2, а нечетные на 1.</li> </ul>
2, 15	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• вывести минимальный элемент для каждой строки;</li> <li>• среднее арифметическое (sp) положительных элементов и модифицированную матрицу, в которой все элементы, меньшие sp заменены на x, а остальные – на y.</li> </ul>
3, 16	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• сумму элементов по каждой строке;</li> <li>• среднее арифметическое элементов по каждому столбцу.</li> </ul>

## Продолжение таблицы 5.1

4, 17	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• вывести среднее арифметическое четных элементов для каждой строки;</li> <li>• модифицированную матрицу, в которой все элементы, меньшие чем среднее арифметическое соответствующей строки заменены на <math>q</math>, а оставшиеся на <math>z</math>.</li> </ul>
5, 18	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• максимальный элемент для каждого столбца;</li> <li>• модифицированную матрицу, в которой все элементы, меньшие по модулю 500 заменены на <math>m</math>, а оставшиеся на <math>g</math>.</li> </ul>
6, 19	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• среднее арифметическое всех отрицательных элементов;</li> <li>• модифицированную матрицу, в которой все элементы, меньшие чем среднее арифметическое отрицательных элементов заменены на <math>-</math>, а оставшиеся на <math>+</math></li> </ul>



## Продолжение таблицы 5.1

7, 20	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• вывести среднее арифметическое нечетных элементов для каждого столбца;</li> <li>• модифицированную матрицу, в которой все элементы, кратные 3 заменены на 3, кратные 7 – на 7, а оставшиеся на 0.</li> </ul>
8, 21	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• среднее арифметическое для каждого столбца;</li> <li>• модифицированную матрицу, в которой все элементы, большие соответствующего значения, вычисленного в задании 2) заменены на L, остальные – на m.</li> </ul>
9, 22	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• сумму элементов для каждой строки;</li> <li>• модифицированную матрицу, в которой все элементы, кратные 5 заменены на 5, остальные – на x</li> </ul>

## Продолжение таблицы 5.1

10, 23	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• среднее арифметическое элементов для каждого столбца;</li> <li>• модифицированную матрицу, в которой все элементы, кратные 3 заменены на 3, кратные 5 – на 5, остальные – на 0</li> </ul>
11, 24	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• сумму элементов для каждой строки;</li> <li>• модифицированную матрицу, в которой элементы большие чем соответствующая сумма строки заменены на +, а меньшие – на -, остальные элементы – без изменений.</li> </ul>
12, 25	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• среднее арифметическое всех отрицательных элементов;</li> <li>• модифицированную матрицу, в которой все элементы, меньшие чем среднее арифметическое отрицательных элементов заменены на -, а оставшиеся на +.</li> </ul>

## Продолжение таблицы 5.1

13, 26	<p>Дана исходная матрица <math>M \times N</math> целых чисел, принадлежащих отрезку <math>[-10^3, 10^3]</math>.</p> <p>В выходной файл необходимо вывести следующие значения:</p> <ul style="list-style-type: none"> <li>• исходную матрицу;</li> <li>• сумму элементов для каждой строки;</li> <li>• модифицированную матрицу, в которой все элементы кратные 3 заменены на III, кратные 5 – на V, кратные 10 – на X, кратные 100 – на C</li> </ul>
--------	--

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Приведите пример объявления массива. Приведите пример инициализации элементов массива.
2. Приведите фрагмент кода, демонстрирующий использование генератора случайных чисел.
3. В чем преимущества и недостатки непосредственного задания значений элементов массива в `{ }` при его объявлении? В чем недостатки такого подхода?
4. Что будет выведено в консоль в результате работы приложения?

```
static void Main(string[] args)
{
    int[,] dots = { { 0, 0, 0 }, { 1, 1, 1 }, { 2, 2, 2 } };

    int zz = 10;
    for (int i = 0; i < 3; i++)
    {
        zz *= dots[0, i];
    }

    Console.Write(zz);
    Console.ReadKey();
}
```

## **ЛАБОРАТОРНАЯ РАБОТА 6. КЛАССЫ. ОБЪЕКТНОЕ МОДЕЛИРОВАНИЕ**

### **ЦЕЛЬ И СОДЕРЖАНИЕ**

Цель лабораторной работы: изучить структуру и принципы объявления классов, освоить технологию создания экземпляров классов (объектов).

Задачи лабораторной работы:

- научиться объявлять классы;
- научиться создавать объекты классов;
- научиться работать с полями данных и методами классов.

### **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

#### **Классы и структуры**

Класс – это тип данных, объединяющий данные и методы их обработки. Класс – это пользовательский шаблон, в соответствии с которым можно создавать объекты. То есть класс – это правило, по которому будет строиться объект. Сам класс не содержит данных.

Объект класса (экземпляр класса) – переменная типа класс. Объект содержит данные и методы, манипулирующие этими данными. Класс определяет, какие данные содержит объект и каким образом он ими манипулирует.

Все что справедливо для классов можно распространить и на структуры. Отличие состоит в методе хранения объектов данных типов в оперативной памяти: структуры – это типы по значению, они размещаются в стеке; классы – это ссылочные типы, объекты классов размещаются в куче. Структуры не поддерживают наследование.

Структуры применяются для представления небольших объемов данных. Объявление структур происходит с использованием ключевого слова `struct`, объявление классов – с помощью ключевого слова `class`.

Пример объявления класса:

```
// Объявление класса
public class MyFirstClass
{
    // Данные-члены класса

    // Доступные на уровне экземпляра
    public int a;
    public float b__;
    public string fio;

    // Доступные только на уровне класса
    private bool IsOK;
    private double precision;
}
```

При создании как классов, так и структур, используется ключевое слово `new`, например:

```
MyFirstClass obj = new MyFirstClass();
```

### Структура класса

Данные и функции, объявленные внутри класса, называются членами класса (class members). Доступность членов класса может быть описана как `public`, `private`, `protected`, `internal` или `internal protected`.

#### Данные-члены

Данные-члены – это те структуры внутри класса, которые содержат данные класса – поля, константы события.

Поля – это любые переменные, ассоциированные с классом. После создания экземпляра класса к полям можно обращаться с использованием синтаксиса , например: `ИмяПоля ИмяОбъекта`.

```
MyFirstClass obj = new MyFirstClass();
```

```
obj.a = 5;
int cc = obj.a;
```

```
obj.fio = "Novak E.I.";
```

Аналогичным образом с классом ассоциируются константы. События будут рассмотрены в следующих лабораторных работах

#### Функции-члены

Функции-члены – это члены, которые обеспечивают некоторую функциональность для манипулирования данными классов. Они делятся на следующие виды: методы, свойства, конструкторы, финализаторы, операции и индексаторы.

Методы (method) – это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора `static`.

Свойства (property) – это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса. В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на `Set` и `Get`, не понадобится. Поскольку не существует какого-то отдельного синтаксиса для свойств, который отличал бы их от нормальных функций, создается иллюзия объектов как реальных сущностей, предоставляемых клиентскому коду.

Конструкторы (constructor) – это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

Финализаторы (finalizer) похожи на конструкторы, но вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды (~). Предсказать точно, когда будет вызван финализатор, невозможно.

Операции (operator) – это простейшие действия вроде `+` или `-`. Когда вы складываете два целых числа, то, строго говоря, применяете операцию `+` к целым. Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операций).

Индексаторы (indexer) позволяют индексировать объекты таким же способом, как массив или коллекцию.

В данной лабораторной работе рассматриваются только методы класса – это функции, ассоциированные с определенным классом.

В C# объявление метода класса состоит из спецификатора доступности, возвращаемого значения, имени метода, списка формальных параметров и тела метода:

```
[модификатор] тип_возврата имя_метода ([список_параметров])
{
    // тело метода
}
```

Например, добавим методы для объявленного ранее класса `MyFirstClass`:

```
// Объявление класса
public class MyFirstClass
{
    // Данные-члены класса

    // Доступные на уровне экземпляра
    public int a;
    public float b__;
    public string fio;

    // Доступные только на уровне класса
    private bool IsOK;
    private double precision;

    // Метод для инициализации некоторых полей класса
    public void InitClassMembers(int pA, float pB__, string pFio)
    {
        a = pA;
        b__ = pB__;
        fio = pFio;
    }

    // Метод, возвращающий значение вычисленное на основе полей класса
    public int GetAbsA()
    {
        return Math.Abs(a);
    }
}
```

Синтаксис вызова методов аналогичен синтаксису обращения к данным-членам:

```
MyFirstClass obj = new MyFirstClass();

obj.InitClassMembers(10, 0.8F, "Новиков П.Е.");

int abs_a = obj.GetAbsA();
```

В данном примере метод `InitClassMembers` не возвращает никаких данных, но требует передачи ему фактических параметров. В свою очередь, метод `GetAbsA` возвращает значение типа `int` и не предполагает никаких параметров.

В общем случае параметры могут передаваться методу либо по значению, либо по ссылке. Когда переменная передается по ссылке, вызываемый метод получает саму переменную, поэтому любые изменения, которым она подвергнется внутри метода, останутся в силе после его завершения. Но если переменная передается по значению, вызываемый метод получает копию этой переменной, а это значит, что все изменения в ней по завершении метода будут утеряны. Для сложных типов данных передача по ссылке более эффективна из-за большого объема данных, который приходится копировать при передаче по значению.

Если не указано обратное, то в С# все параметры передаются по значению. Тем не менее, можно принудительно передавать значения по ссылке, для чего используется ключевое слово `ref`. Если параметр передается в метод, и входной аргумент этого метода снабжен префиксом `ref`, то любые изменения этой переменной, которые сделает метод, отразятся на исходном объекте.

В С-подобных языках функции часто возвращают более одного значения. Это обеспечивается применением выходных параметров, за счет присваивания значений переменным, переданным в метод по ссылке. Часто первоначальное значение таких переменных не важно. Эти значения перезаписываются в функции, которая может даже не обращать внимания на то, что в них хранилось первоначально.

Было бы удобно использовать то же соглашение в С#. Однако в С# требуется, чтобы переменные были инициализированы каким-то начальным значением перед тем, как к ним будет выполнено обращение. Хотя можно инициализировать входные переменные какими-то бессмысленными значениями до передачи их в функцию, которая наполнит их осмысленными значениями, этот прием выглядит в лучшем случае излишним, а в худшем — сбивающим с толку. Тем не менее, существует способ обойти требование компилятора С# относительно начальной инициализации переменных.

Это достигается ключевым словом `out`. Когда входной аргумент снабжен префиксом `out`, этому методу можно передать неинициализированную переменную. Переменная передается по ссылке, поэтому любые изменения, выполненные методом в переменной, сохраняются после того, как он вернет управление.



Ключевое слово `out` также должно указываться при вызове метода – так же, как при его определении

#### *Частичные классы.*

Ключевое слово `partial` (частичный) позволяет определить класс, структуру или интерфейс, распределенный по нескольким файлам. Но ситуации, когда множеству разработчиков требуется доступ к одному и тому же классу, или же в ситуации, когда некоторый генератор кода генерирует часть класса, такое разделение класса на несколько файлов может оказаться полезным. Ключевое слово `partial` просто помещается перед классом, структурой или интерфейсом

#### *Статические классы.*

Статический класс функционально представляет собой то же самое, что и класс с приватным статическим конструктором. Создать экземпляр такого класса невозможно. Если указать ключевое слово `static` в объявлении класса, компилятор будет гарантировать, что к этому классу никогда не будут добавлены нестатические члены.

#### *Класс Object.*

Классы .NET изначально унаследованы от `System.Object`. Фактически, если при определении нового класса базовый класс не указан, компилятор автоматически предполагает, что он наследуется от `Object`.

Практическое значение этого в том, что помимо методов и свойств, которые программист определяет самостоятельно, также появляется доступ к множеству общедоступных и защищенных методов-членов, которые определены в классе `Object`. Эти методы присутствуют во всех определяемых классах.

## **МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

### **Учебная задача**

Условие. Разработать класс для представления объекта «Прямоугольный параллелепипед». Реализуйте все необходимые поля данных (закрытые) и методы позволяющие:

- считывать из файла состояние полей данных объекта;
- вычислять объем прямоугольного параллелепипеда;

- вычислять площадь поверхности прямоугольного параллелепипеда;
- выводить полную информацию об объекте в выходной файл.

Решение. Решение данной задачи состоит из следующих этапов:

- проектирование класса;
- объявление класса (кодирование);
- демонстрация использования класса (создание объекта и вызов методов).

Проектирование класса можно производить с использованием любого средства проектирования (UML-редактор, онлайн средства построения диаграмм (<https://www.draw.io>), обычный графический редактор). На рисунке 6.1 показана диаграмма классов для спроектированного класса Par.

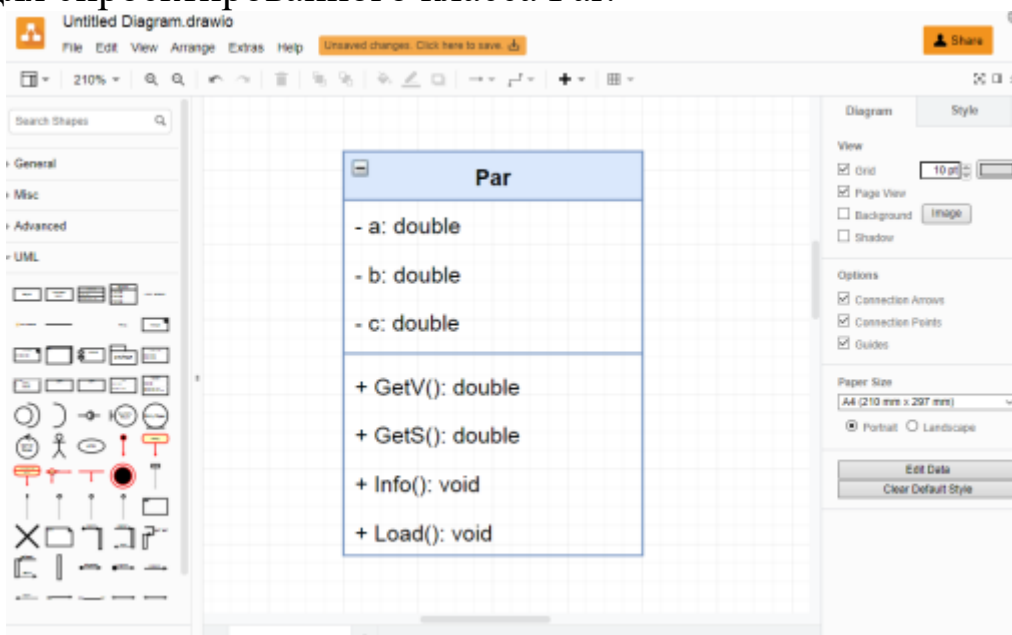


Рисунок 6.1 – Класс Par в редакторе диаграмм классов

Проект класса учитывает все данные и методы, которые можно выявить, проанализировав задание:

- в классе определены три закрытые переменные a, b, c, описывающие стороны прямоугольного параллелограмма;
- в классе должны быть следующие открытые методы: GetV() – возвращает объем прямоугольного параллелограмма; GetS() – возвращает площадь поверхности прямоугольного парал-

лелограмма; Info() – выводит в стандартный поток информацию о параллелограмме; Load() – производит загрузку данных из входного потока для инициализации закрытых полей.

Переходим к кодированию. Добавляем новый класс Par в проект Visual Studio, наполняем его полями данных и методами (пока пустыми, позже реализуем). Листинг данного класса с методами-заглушками представлен на рисунке 6.2.

```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Par
7      {
8          private double a, b, c;
9
10         public double GetV() { throw new NotImplementedException(); }
11         public double GetS() { throw new NotImplementedException(); }
12         public void Info() { throw new NotImplementedException(); }
13         public void Load() { throw new NotImplementedException(); }
14     }
15 }

```

Рисунок 6.2 – «Набросок» класса Par в Visual Studio

Представленный набросок класса содержит пустые методы, но в проектах на любом языке нельзя оставлять методы-заглушки пустыми, или возвращающими нулевые значения. В большом объеме кода можно забыть реализовать пустой метод и возникнет трудноуловимая ошибка. Поэтому в теле метода-заглушки вставляют вызов исключения **throw new NotImplementedException()**. При попытке вызова такого метода программа сгенерирует исключение, и программа завершится аварийно.

После этого можно наполнить класс содержанием. На рисунке 6.3 показан листинг класса, содержащий определение всех методов.

```

3 namespace Test
4 {
5     class Par
6     {
7         private double a, b, c;

8         ссылка: 1
9         public double GetV() { return a * b * c; }

10
11        ссылка: 1
12        public double GetS() { return 2 * (a*b + b*c + a*c); }

13        ссылка: 1
14        public void Info() {
15            String str =
16                "*****\n" +
17                "*\n" +
18                "*\n" +
19                "*\n" +
20                "*\n" +
21                "*****\n";
22
23            Console.WriteLine(str);
24            Console.WriteLine(string.Format(
25                "A = {0:0.00}, B = {0:0.00}, C = {0:0.00}", a, b, c));
26            Console.WriteLine(string.Format("V = {0:0.00}", GetV()));
27            Console.WriteLine(string.Format("S = {0:0.00}", GetS()));
28        }
29
30        ссылка: 0
31        public void Load() {
32            a = Convert.ToDouble(Console.ReadLine());
33            b = Convert.ToDouble(Console.ReadLine());
34            c = Convert.ToDouble(Console.ReadLine());
35        }
36    }
37 }

```

Рисунок 6.3 – Определение класса Par

Теперь можно продемонстрировать использование класса Par. В данной работе необходимо создать две версии исполняемого файла: Release и Debug. Версия Release будет работать с файлами par\_input.txt и output.txt в качестве источников ввода-вывода. Версия Debug будет использовать в качестве источников стандартных потоков клавиатуру и консоль (что установлено по умолчанию). На рисунке 6.4 представлен листинг, демонстрирующий работу с классом Par.

```

1  using System;
2  using System.IO;
3
4  namespace Test
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             #if !DEBUG
11                 TextWriter save_out = Console.Out;
12                 TextReader save_in = Console.In;
13                 var new_out = new StreamWriter(@"par_output.txt");
14                 var new_in = new StreamReader(@"par_input.txt");
15                 Console.SetOut(new_out);
16                 Console.SetIn(new_in);
17             #endif
18
19             Par p = new Par();
20             p.Load();
21             p.Info();
22
23             #if !DEBUG
24                 Console.SetOut(save_out); new_out.Close();
25                 Console.SetIn(save_in); new_in.Close();
26             #else
27                 Console.ReadKey();
28             #endif
29         }
30     }
31 }

```

Рисунок 6.4 – Использование класса Par

Следует обратить внимание на использование директивы препроцессора `#if`, `#else`, `#endif`. Данные директивы реализуют компиляцию кода, только если определен соответствующий идентификатор (в данном случае `DEBUG`). В данном случае используется конструкция:

```
#if !DEBUG
```

```
...
```

```
#endif
```

Это значит, что если «не `DEBUG`» (т.е. `RELEASE`), то код заключенный внутри условного блока будет компилироваться, в противном случае – нет.

Переключить режим сборки можно в настройках сборки конфигурации (рисунок 6.5).

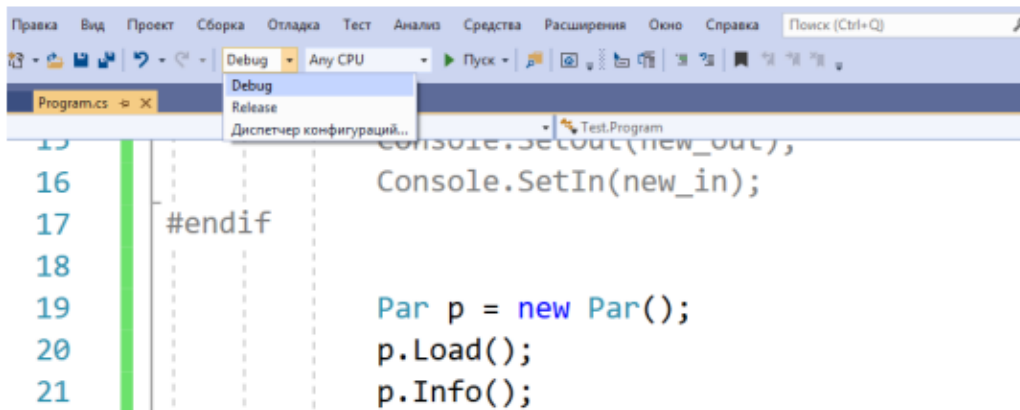


Рисунок 6.5 – Настройки конфигурации

Visual Studio создает две сборки: одна находится в папке Release, вторая – в Debug, соответственно одна программа Release – для заказчика будет работать с файлами, вторая Debug – для отладки работает с консолью.

На рисунке 6.6 показана программа Debug, на рисунке 6.7 представлены входной и выходной файл программы, собранной как Release.

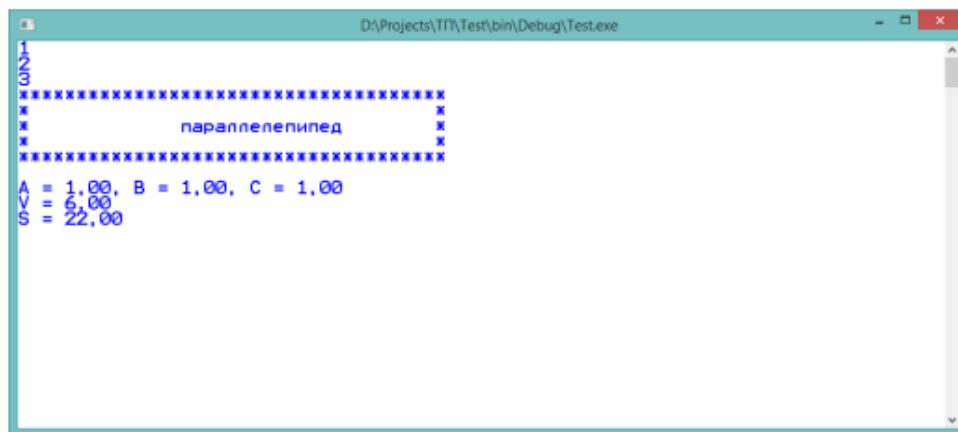


Рисунок 6.6 – Выполнение debug-приложения

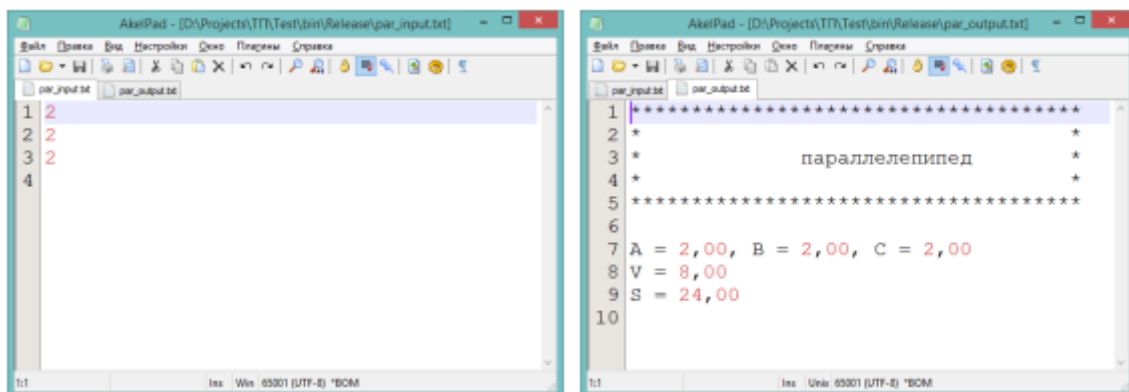


Рисунок 6.7 – Файлы release-приложения

Таким образом, в версии Debug можно реализовать довольно сложный диалог с пользователем, отладчиком, тестировщиком, а в версии Release программа будет обрабатывать данные из файлов в фоновом режиме.

### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса. Реализуйте две версии: debug и release, продемонстрируйте разное поведение программы для различных конфигураций сборки.

Таблица 6.1

Индивидуальные задания

Вариант	Задание
1, 14	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
2, 15	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
3, 16	Класс «График $y=x$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
4, 17	Класс «Матрица $M \times N$ ». Реализовать инициализацию элементов матрицы случайными числами, вывод матрицы, нахождение максимального и минимального элементов, а также вывод информации об объекте.
5, 18	Класс «Прямоугольный треугольник». Реализовать ввод и вывод полей данных, вычисление гипотенузы, площади и периметра, а также вывод информации об объекте.
6, 19	Класс «График $y=3x+5$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.

Продолжение таблицы 6.1

7, 20	Класс «График $y=x-10$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
8, 21	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
9, 22	Класс «График $y=x$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
10, 23	Класс «Правильная треугольная пирамида». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
11, 24	Класс «Усеченный конус». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
12, 25	Класс «Четырехугольная пирамида». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
13, 26	Класс «Транспорт» с полями: скорость, объем бака, расход топлива на 1 километр. Реализовать ввод и вывод полей данных, вычисление времени в пути по переданному расстоянию, количества затрачиваемого топлива по переданному расстоянию, хватит или нет топлива на заданное расстояние, а также вывод информации об объекте.



## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Приведите определение понятия «класс».
2. Что такое структура? Чем структура отличается от класса?
3. Чем отличаются определения классов?

```
class P {
    int a, b, c;
}
```

```
class P {
    private int a, b, c;
}
```

4. Опишите ошибки компиляции (если есть) в представленном фрагменте:

Ссылка: 2

```
class Person {
    private int a, b, c;
}
```

Ссылка: 0

```
static void Main(string[] args)
{
    Person p = new Person();
    p.a = 100;
    p.b = 200;
    p.c = 300;
}
```

5. Как называется переменная типа класс?

## **ЛАБОРАТОРНАЯ РАБОТА 7. КОНСТРУКТОР КЛАССА. ПЕРЕГРУЗКА МЕТОДОВ КЛАССА**

### **ЦЕЛЬ И СОДЕРЖАНИЕ**

*Цель лабораторной работы:* понять принципы работы конструктора.

Задачи лабораторной работы:

- научиться объявлять конструктор класса;
- научиться создавать перегруженные конструкторы.

### **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

Конструктор – это метод класса, который не возвращает значения и имеет то же самое имя, что и класс. Если конструктор класса не определен программистом явно, то компилятор создаст конструктор по умолчанию.

Конструкторы подчиняются тем же правилам перегрузки, что и все методы.

В C# поддерживается перегрузка методов – то есть может существовать несколько версий одного метода, но с разными сигнатурами (методы отличаются количеством и / или типом параметров). Чтобы перегрузить метод, просто объявляются методы с одинаковыми именами, но разными сигнатурами.

### **МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

#### **Учебная задача**

Условие. Доработать класс, спроектированный в предыдущей лабораторной работе, следующим образом:

- обеспечить перегрузку методов класса (2-3 перегруженных метода);
- обеспечить перегрузку конструктора (2-3 перегруженных конструктора).

Решение. Для выполнения данного задания добавим к классу `Par` (рисунок 6.3) следующие методы (рисунок 7.1):

```

1  using System;
2
3  namespace Test
4  {
5      Ссылка: 7
6      class Par
7      {
8          private double a, b, c;
9
10         Ссылка: 0
11         private Par() { }
12
13         Ссылка: 2
14         public Par(double pA, double pB, double pC)
15         {
16             a = pA; b = pB; c = pC;
17         }
18
19         Ссылка: 0
20         private void Load()
21         {
22             a = Convert.ToDouble(Console.ReadLine());
23             b = Convert.ToDouble(Console.ReadLine());
24             c = Convert.ToDouble(Console.ReadLine());
25         }
26
27         public static Par CreateParFromFile()
28         {
29             double aa = Convert.ToDouble(Console.ReadLine());
30             double bb = Convert.ToDouble(Console.ReadLine());
31             double cc = Convert.ToDouble(Console.ReadLine());
32             return new Par(aa, bb, cc);
33         }
34
35         Ссылка: 1
36         public double GetV() { return a * b * c; }
37
38         Ссылка: 1
39         public double GetS() { return 2 * (a * b + b * c + a * c); }
40
41         Ссылка: 3
42         public void Info()
43         {
44             String str =
45                 "\n*****\n" +
46                 "*****\n" +
47                 "параллелепипед\n" +
48                 "*****\n";
49
50             Console.WriteLine(str);
51             Console.WriteLine(string.Format(
52                 "A = {0:0.00}, B = {0:0.00}, C = {0:0.00}", a, b, c));
53             Console.WriteLine(string.Format("V = {0:0.00}", GetV()));
54             Console.WriteLine(string.Format("S = {0:0.00}", GetS()));
55         }
56
57         Ссылка: 0
58         public void Info(ConsoleColor fg, ConsoleColor bgc)
59         {
60             Console.ForegroundColor = fg;
61             Console.BackgroundColor = bgc;
62             Console.Clear();
63             Info();
64         }
65     }
66 }

```

Рисунок 7.1 – Доработка класса Par

На рисунке 7.2 демонстрируется использование модифицированного класса:

```

3
4 namespace Test
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             #if !DEBUG
11                 TextWriter save_out = Console.Out;
12                 TextReader save_in = Console.In;
13                 var new_out = new StreamWriter(@"par_output.txt");
14                 var new_in = new StreamReader(@"par_input.txt");
15                 Console.SetOut(new_out);
16                 Console.SetIn(new_in);
17             #endif
18             Par p1, p2;
19             #if DEBUG
20                 p2 = new Par(5.7, 11, 18);
21                 p2.Info(ConsoleColor.Yellow, ConsoleColor.Blue);
22             #endif
23             #if !DEBUG
24                 p1 = Par.CreateParFromFile();
25                 p1.Info();
26                 p2 = new Par(5.6, 5.11, 10.5);
27                 p2.Info();
28             #endif
29             #if !DEBUG
30                 Console.SetOut(save_out); new_out.Close();
31                 Console.SetIn(save_in); new_in.Close();
32             #endif
33             #if DEBUG
34                 Console.ReadKey();
35             #endif
36         }
37     }
38 }
39
40

```

Рисунок 7.2 – Использование класса Par с перегруженными методами

На рисунке 7.3 представлен результат работы debug-программы.

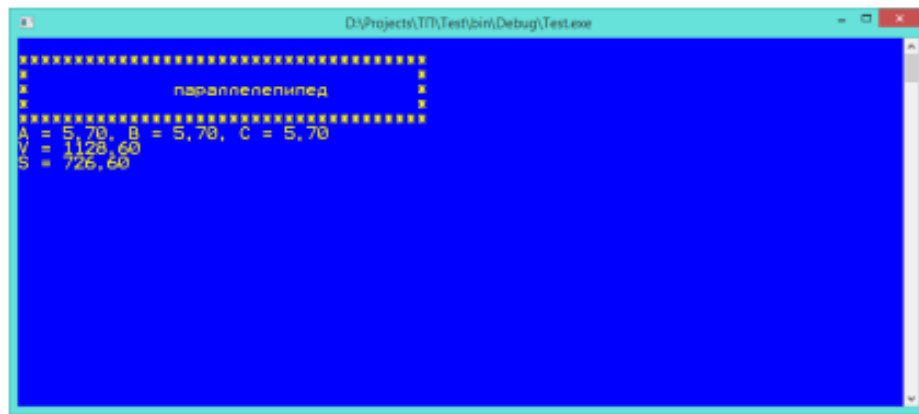


Рисунок 7.3 – Выполнение debug-приложения

На рисунке 7.4 показан входной и выходной файл с которыми работает release-приложение.

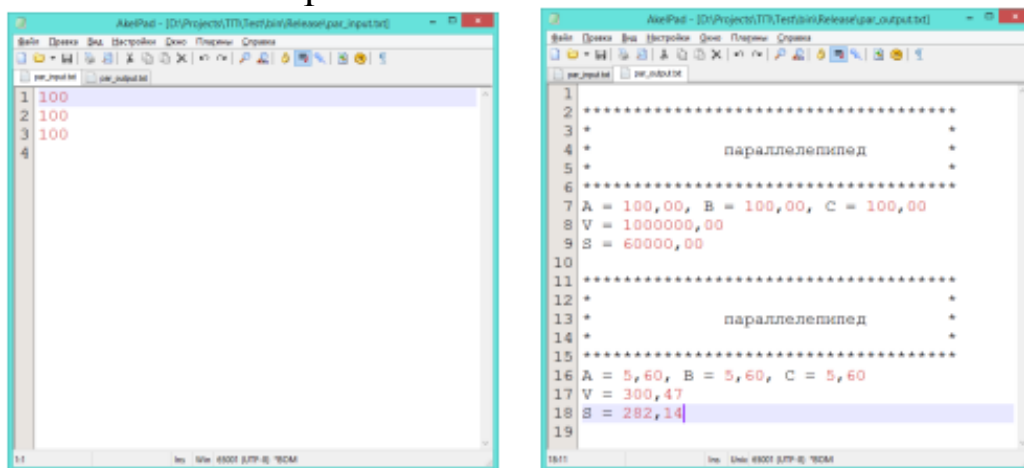


Рисунок 7.4 – Файлы release-приложения

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Условие. Спроектировать и реализовать класс в соответствии с вариантом, с учетом следующих требований:

1. обеспечить перегрузку методов класса (2-3 перегруженных метода);
2. обеспечить перегрузку конструктора (2-3 перегруженных конструктора);
3. различное поведение приложения в debug и release сборках.

Таблица 7.1

## Индивидуальные задания

Вариант	Задание
1, 14	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
2, 15	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
3, 16	Класс «График $y=x$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
4, 17	Класс «Матрица $M \times N$ ». Реализовать инициализацию элементов матрицы случайными числами, вывод матрицы, нахождение максимального и минимального элементов, а также вывод информации об объекте.
5, 18	Класс «Прямоугольный треугольник». Реализовать ввод и вывод полей данных, вычисление гипотенузы, площади и периметра, а также вывод информации об объекте.
6, 19	Класс «График $y=3x+5$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
7, 20	Класс «График $y=x-10$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
8, 21	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.

Продолжение таблицы 7.1

9, 22	Класс «График $y=x$ ». Реализовать ввод и вывод полей данных, вычисление интеграла функции от $a$ до $b$ (вводятся пользователем), длины отрезка функции от $(a, y(a))$ до $(b, y(b))$ , а также вывод информации об объекте.
10, 23	Класс «Правильная треугольная пирамида». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
11, 24	Класс «Усеченный конус». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
12, 25	Класс «Четырехугольная пирамида». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
13, 26	Класс «Транспорт» с полями: скорость, объем бака, расход топлива на 1 километр. Реализовать ввод и вывод полей данных, вычисление времени в пути по переданному расстоянию, количества затрачиваемого топлива по переданному расстоянию, хватит или нет топлива на заданное расстояние, а также вывод информации об объекте.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое статический класс?
2. Что такое статический метод?
3. Что такое конструктор класса?
4. Что такое перегрузка методов?
5. Может ли один конструктор класса вызывать другой конструктор? Прежде чем отвечать попробуйте реализовать такой вызов в своем разработанном классе.
6. Сколько перегрузок может иметь метод класса?
7. Для каких целей следует перегружать конструктор класса?
8. Что такое сигнатура метода?

9. Какие ошибки присутствуют в объявлении класса?

```
public class Vector
{
    float x;
    float y;
    float z;

    Vector(float pX, float pY, float pZ)
    {
        x = pX; y = pY; z = pZ;
    }

    public Vector()
        : this(0, 0, 0)
    {
    }
}
```



## ЛАБОРАТОРНАЯ РАБОТА 8. ПРОЕКТИРОВАНИЕ ИЕРАРХИИ КЛАССОВ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* изучить механизм организации наследования классов.

Задачи лабораторной работы:

- научиться объявлять производные классы;
- научиться создавать иерархии классов;
- научиться использовать механизм полиморфизма.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

#### Наследование реализации

Наследование реализации (implementation inheritance) означает, что тип происходит от базового типа, получая от него все поля-члены и функции-члены.

Синтаксис наследования реализации:

```
class ПроизводныйКласс: БазовыйКласс
{
    // Данные – члены и функции – члены
}
```

Если при определении класса не указан базовый класс, то С# предполагает, что базовым классом является System.Object.

#### Создание иерархии классов

При наследовании реализации производный класс наследует реализацию каждой функции базового типа, если только в его определении не указано, что реализация функции должна быть переопределена.

Определим следующую иерархию классов (рис. 6.1) и продемонстрируем, как наследуется реализация и как переопределяются свойства и методы.



Рисунок 6.1 – Иерархия классов

Создадим код, описывающий данную иерархию. Определим базовый класс:

```
class Человек
{
    public Человек(string Ф, string И, string О, int Возр)
    {
        Фамилия = Ф; Имя = И; Отчество = О; Возраст = Возр;
    }

    public Человек()
    {
        Фамилия = "нет данных"; Имя = ""; Отчество = "";
        Возраст = 18;
    }

    public string Фамилия, Имя, Отчество;
    private DateTime ДатаРождения;

    public virtual string ФИО
    {
        get { return Фамилия + " " + Имя + " " + Отчество; }
    }

    public int Возраст
    {
        get { return DateTime.Now.Year - ДатаРождения.Year; }
        set
        {
            int ГодРождения = DateTime.Now.Year - value;
            ДатаРождения = Convert.ToDateTime(ГодРождения.ToString()+".01.01");
        }
    }
}
```

Обратите внимание на наличие двух конструкторов, механизм хранения возраста человека и ключевое слово `virtual` у свойства «ФИО». Ключевое слово `virtual` указывает, что данное свойство будет переопределено в производном классе.

Определим производный класс «Учитель»:

```

class Учитель: Человек
{
    public Учитель()
        :base()
    {
        УченоеЗвание = УченыеЗвания.Без_Звания;
        УченаяСтепень = УченыеСтепени.Без_Степени;
    }

    public Учитель(string Ф, string И, string О, int Возр, УченыеЗвания УЗ, УченыеСтепени УС)
        :base(Ф, И, О, Возр)
    {
        УченоеЗвание = УЗ;
        УченаяСтепень = УС;
    }

    public УченыеЗвания УченоеЗвание;
    public УченыеСтепени УченаяСтепень;

    public override string ФИО
    {
        get
        {
            return УченаяСтепень.ToString() + ", " + УченоеЗвание.ToString() + ", " + base.ФИО;
        }
    }
}

```

Следует обратить внимание на использование ключевого слова `override` у свойства «ФИО», которое указывает на то, что данное свойство имеет новую реализацию, отличающуюся от реализации базового класса.

Определим второй производный класс «Студент»:

```

class Студент: Человек
{
    public Студент(string Ф, string И, string О, int Возр, Специальности Спец)
        :base(Ф, И, О, Возр)
    {
        Специальность = Спец;
    }

    public Специальности Специальность;

    public override string ФИО
    {
        get
        {
            return base.ФИО + ", " + Специальность.ToString();
        }
    }
}

```

В обоих производных классах следует обратить внимание на реализацию конструкторов производных классов, использование ключевого слова `base` в коде свойства «ФИО» и при объявлении конструкторов.

Также интерес представляют типы данных, используемые для членов данных: «Специальности», «УченыеЗвания», «УченыеСтепени». Вот определения данных типов-перечислений:

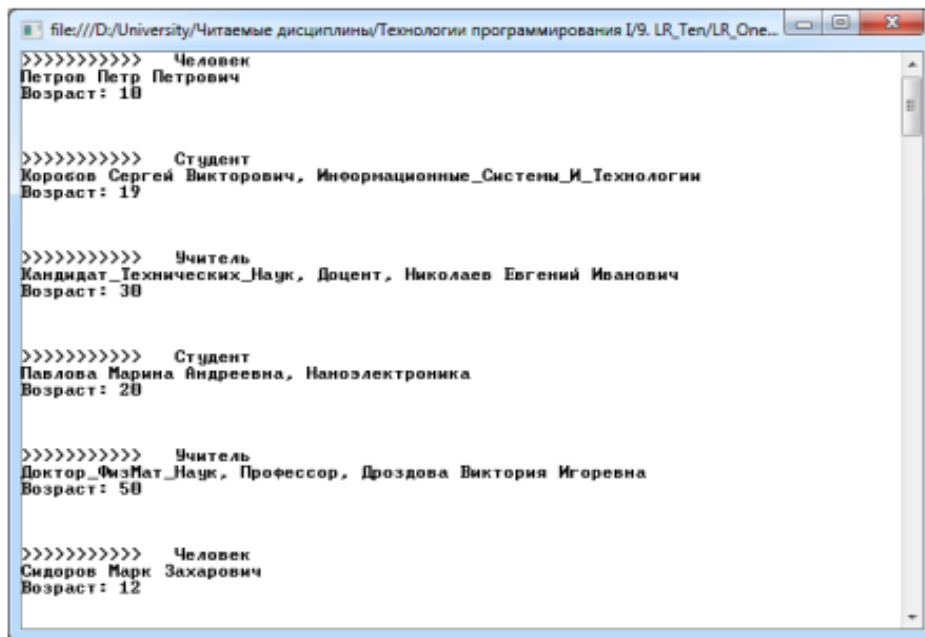
```
public enum УченыеЗвания
{
    Доцент,
    Профессор,
    Академик,
    Без_Звания
}

public enum УченыеСтепени
{
    Кандидат_Технических_Наук,
    Кандидат_ФизМат_Наук,
    Кандидат_Педагогических_Наук,
    Доктор_ФизМат_Наук,
    Без_Степени
}

public enum Специальности
{
    Информационные_Системы_И_Технологии,
    Безопасность_Информационных_Систем,
    Технология_Защиты_Информации,
    Психология,
    Нанозлектроника
}
```

Наконец, продемонстрируем использование объявленной иерархии классов. В программе объявим массив объектов типа «Человек». Следует понимать, что при объявлении такого массива, его элементам можно присваивать объекты любого производного класса, причем каждый объект будет вести себя по-своему (за счет переопределения свойств и методов).





Полная диаграмма типов в полученном приложении показана на рис. 6.2.

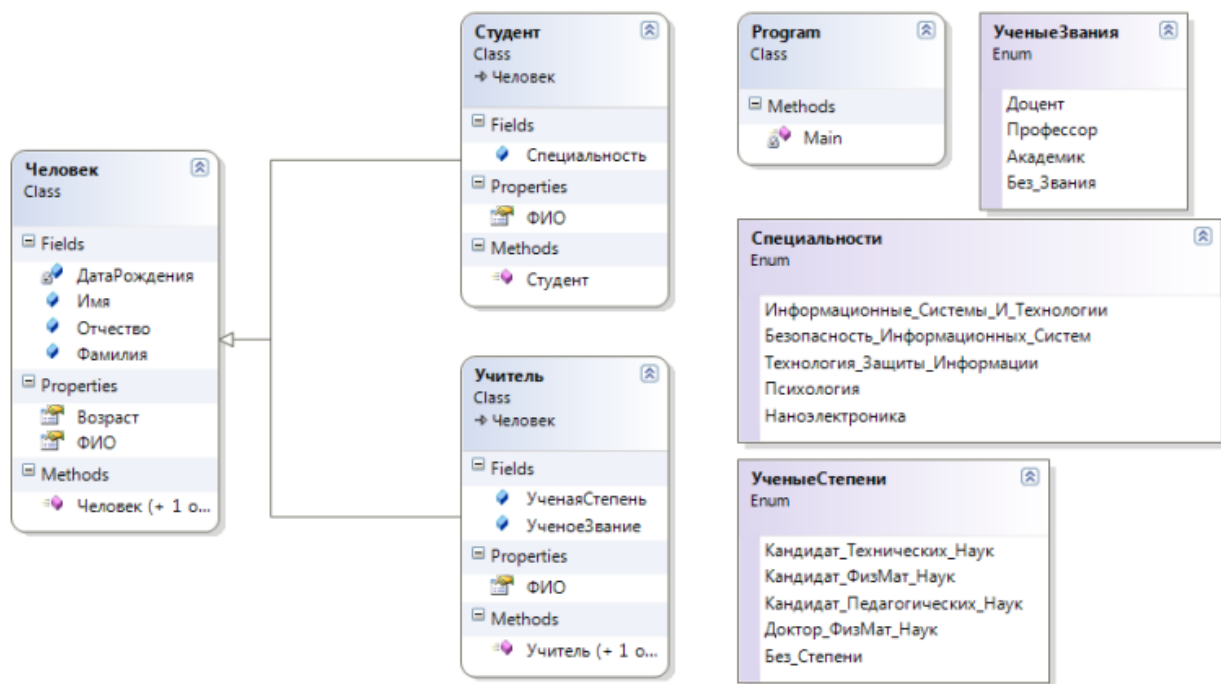


Рисунок 6.2 – Диаграмма типов приложения (создана средствами VS)

Структуры всегда наследуются от System.ValueType. Они могут также наследовать любое количество интерфейсов. Классы всегда наследуются от одного класса по вашему выбору. Они также могут наследовать любое количество интерфейсов.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Создайте консольное приложение.
2. Изучите пример создания иерархии классов, представленный в разделе «Теоретическое обоснование» данной лабораторной работы.
3. Постройте свою иерархию классов в соответствии с индивидуальным заданием. В результате выполнения лабораторной работы должны быть реализованы следующие механизмы:
  - использование типа-перечисления (хотя бы одного);
  - использование переопределенного свойства (хотя бы одного);
  - использование переопределенного метода (хотя бы одного);
  - использование вызова базового конструктора;
  - использование вызова любого базового метода (отличного от конструктора).
4. Продемонстрируйте использование классов, созданной иерархии (легче всего это сделать с использованием массивов). При защите работы укажите признаки присутствия полиморфного поведения в программе (реализация полиморфизма).

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса, продемонстрируйте полиморфное поведение объектов.

Таблица 8.1

Индивидуальные задания

Вариант	Задание
1, 14	<pre> graph BT     ГрузовойАвто --&gt; Автомобиль     ЛегковойАвто --&gt; Автомобиль     Автомобиль --&gt; ТехСредство     Вертолёт --&gt; ТехСредство           </pre>

Продолжение таблицы 8.1

2, 15	<pre> graph BT     Сервер --&gt; КомпТехника     Ноутбук --&gt; КомпТехника     ПК --&gt; КомпТехника           </pre>
3, 16	<pre> graph BT     Стрелковое --&gt; Вооружение     Танк --&gt; Вооружение     Сомолет --&gt; Вооружение           </pre>
4, 17	<pre> graph BT     Вертолет --&gt; Авиасредство     Истребитель --&gt; Авиасредство     Авиасредство --&gt; Машины     Грузовик --&gt; Машины           </pre>
5, 18	<pre> graph BT     СвободныеОС --&gt; ОС     ПлатныеОС --&gt; ОС     ОС --&gt; ПО     Прикладное --&gt; ПО           </pre>
6, 19	<pre> graph BT     Планшет --&gt; Мобильный     Ноутбук --&gt; Мобильный     Мобильный --&gt; ЭВМ     Стационарный --&gt; ЭВМ           </pre>
7, 20	<pre> graph BT     ГрузовойАвто --&gt; Автомобиль     ЛегковойАвто --&gt; Автомобиль     Автомобиль --&gt; ТехСредство     Вертолёт --&gt; ТехСредство           </pre>



Продолжение таблицы 8.1

8, 21	<pre> graph BT     Вертолет --&gt; Авиасредство     Истребитель --&gt; Авиасредство     Авиасредство --&gt; Машины     Грузовик --&gt; Машины </pre>
9, 22	<pre> graph BT     Трактор --&gt; Тяжелое     Комбайн --&gt; Тяжелое     Тяжелое --&gt; Машиностроение     Среднее --&gt; Машиностроение </pre>
10, 23	<pre> graph BT     ПрямПараллелепипед --&gt; Многоугольник     Треугольник --&gt; Многоугольник     Многоугольник --&gt; Геометрия     Эллипс --&gt; Геометрия </pre>
11, 24	<pre> graph BT     СвободныеОС --&gt; ОС     ПлатныеОС --&gt; ОС     ОС --&gt; ПО     Прикладное --&gt; ПО </pre>
12, 25	<pre> graph BT     Стол --&gt; Мебель     МягкМебель --&gt; Мебель     Шкаф --&gt; Мебель </pre>
13, 26	<pre> graph BT     Накопительная --&gt; БанкКарта     Платежная --&gt; БанкКарта     Зарплатная --&gt; БанкКарта </pre>

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое наследование реализации? Как описать синтаксически наследование реализации?
2. Для чего используется ключевое слово `base`?
3. Можно ли переопределить метод класса? Свойства класса? Данные класса?
4. Как переопределить метод в производном классе?
5. Для чего используется ключевое слово `virtual`?
6. Для чего используется ключевое слово `override`?
7. Как поменять цвет фона в консольном приложении?
8. Укажите ошибки (если есть) в представленном фрагменте:

```
public class Parent
{
    int a;
}

class Child : Parent
{
    int b;

    void E()
    {
        b = a;
    }
}
```

## ЛАБОРАТОРНАЯ РАБОТА 9. ПОЛИМОРФИЗМ НА ОСНОВЕ ИНТЕРФЕЙСОВ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* изучить принципы работы с типами интерфейсов в C#.

Задачи лабораторной работы:

- научиться объявлять интерфейсы в C#;
- научиться создавать классы, реализующие интерфейсы.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Кроме наследования реализации (implementation inheritance) в языке C# реализован механизм наследования интерфейсов (interface inheritance). Необходимо понимать, что не все объектно-ориентированные языки поддерживают интерфейсы.

Если класс наследует интерфейс, класс как бы берет на себя обязательства реализовать некоторый функционал.

Синтаксис наследования интерфейса не отличается от синтаксиса наследования реализации:

```
class КлассРеализующийИнтерфейс: Интерфейс
{
    // Данные – члены и функции – члены
}
```

Отличие от механизма наследования реализации состоит в том, что:

- класс обязательно должен реализовать методы и свойства, продекларированные в интерфейсе (никакой реализации в интерфейсе не существует);
- класс может реализовывать (наследовать) несколько интерфейсов, тогда как базовый класс может быть только один.

Для объявления типа интерфейса используется ключевое слово `interface`:

```

public interface ICalculate
{
    void Plus(int pPlus);
    void Minus(int pMinus);
}

public interface IVisual
{
    string Name { get; set; }
    void DrawObject();
}

```

В приведенном примере объявлены два интерфейса: ICalculate и IVisual.

Интерфейс ICalculate включает два метода с одним параметром. Из названий видно, что один метод что-то увеличивает, второй – что-то уменьшает на величину параметра.

Второй интерфейс IVisual содержит метод DrawObject без параметров, который призван нарисовать объект, а также свойство Name, которое доступно как для чтения, так и для записи.

Следует понимать, что интерфейсы не предполагают конкретную реализацию методов, а свойства интерфейсов не хранят данных. Интерфейс должен быть реализован классом, который и определит конкретное наполнение методов и свойств интерфейса.

Для использования объявленных интерфейсов создадим два класса Human (человек) и Car (автомобиль).

```

public class Human
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;
}

public class Car
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;
}

```

В каждом классе определен конструктор, который инициализирует закрытые данные-члены класса.

Требуется для каждого класса реализовать вывод в консоль, а также возможности увеличения скорости автомобиля и возможности изменения возраста у человека.

Этот общий функционал можно реализовать с использованием следующих способов:

- Добавить все необходимые функции в каждый класс независимо. Этот метод приводит к «разбуханию кода», сложной управляемости кода.
- Реализовать один класс, например, Base (базовый), в котором определить общие методы и свойства (Plus, Minus, DrawObject). Затем использовать класс Base в качестве базового для классов Human и Car, причем в каждом классе переопределить методы базового класса с использованием ключевого слова `override`. Этот метод — наследование реализации. Получается иерархия классов, которые по смыслу и наполнению сильно отличаются. Хотя в данном подходе классы и код более упорядочены, все равно возникает избыточность за счет многократного переопределения методов.

- Определение интерфейсов и реализация возможностей интерфейсов данными классами.

Именно метод 3 будет использован в данной лабораторной работе.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Учебная задача

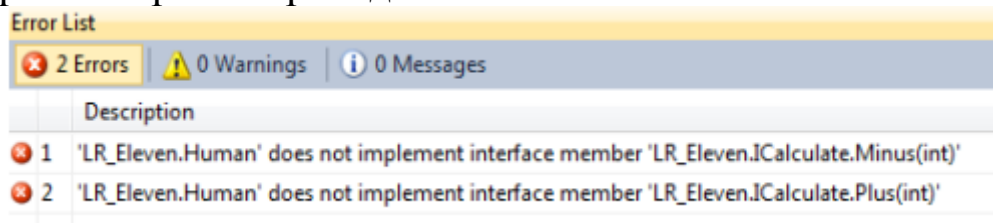
1. Создайте консольное приложение.
2. Определите в приложении классы Human и Car, а также интерфейсы ICalculate и IVisual, представленные в разделе «Теоретическое обоснование» данной лабораторной работы.

3. Реализуем механизм наследования интерфейса ICalculate классом Human. Для этого выполним следующие действия:

- 3.1. В определении класса укажем, что класс Human наследует интерфейс ICalculate.

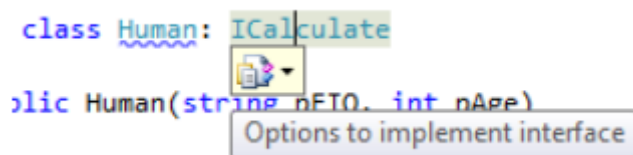
```
public class Human: ICalculate
```

- 3.2. Обратите внимание, что после этого попытка перекомпилировать проект приведет к ошибкам:

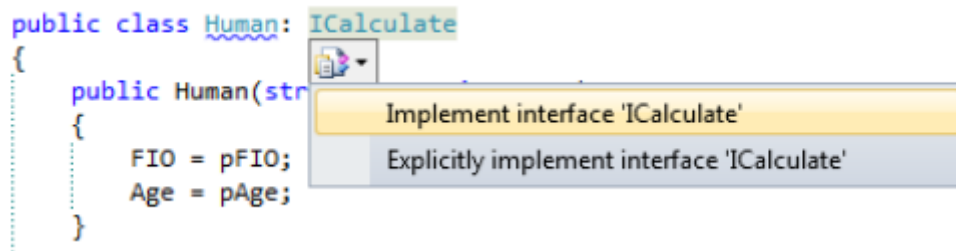


То есть компилятор сообщает, что интерфейс наследуется классом, но методы, заявленные в интерфейсе, классом не реализованы. Реализуем их.

- 3.3. Наведите курсор мыши на интерактивное подчеркивание и появится выпадающий список:



Раскройте его и выберите команду «Implement interface» (реализовать интерфейс).



3.4. В классе Human появятся два новых метода, как и было объявлено в интерфейсе ICalculate. Определение класса примет вид:

```
public class Human: ICalculate
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        throw new NotImplementedException();
    }

    public void Minus(int pMinus)
    {
        throw new NotImplementedException();
    }
}
```

Обратите внимание, что в каждый сгенерированный метод среда разработки добавила код вызова исключения, то есть проект скомпилируется без ошибок, но в процессе выполнения, при попытке использовать методы Plus или Minus программа завершится с ошибками. Это сделано для того, чтобы программист не забыл реализовать данные методы интерфейсов.

В процессе выполнения пп. 3.1 – 3.3 были использованы возможности Visual Studio по автоматизации реализации интерфейса. Очевидно, что интерфейс можно было реализовать, самостоятельно написав данный код.

- использование вызова базового конструктора;
- использование вызова любого базового метода (отличного от конструктора).

4. Определим окончательную реализацию для методов Plus и Minus:

```
public void Plus(int pPlus)
{
    Age += pPlus;
}

public void Minus(int pMinus)
{
    Age -= pMinus;
}
```

5. Аналогичным образом реализуем наследование интерфейса IVisual для класса Human. Окончательно для класса Human получим:

```
public class Human: ICalculate, IVisual
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        Age += pPlus;
    }

    public void Minus(int pMinus)
    {
        Age -= pMinus;
    }

    public string Name
    {
        get
        {
            return FIO + " : " + Age.ToString();
        }
        set
        {
            FIO = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
        (
            "  o  \n" +
            "  --- \n" +
            "  |  \n" +
            "  / \ \ \n" +
            "  / \ \ \n"
        );
        Console.WriteLine(Name);
    }
}
```



6. Реализуем интерфейсы ICalculate и IVisual для класса Car:

```
public class Car: IVisual, ICalculate
{
    public Car(string pManufacturer, string pModel, int
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;

    public void Plus(int pPlus)
    {
        Velocity += pPlus;
    }

    public void Minus(int pMinus)
    {
        Velocity += pMinus;
    }

    public string Name
    {
        get
        {
            return Manufacturer + " - " + Model +
                " : " + Velocity.ToString() + "km/h";
        }
        set
        {
            Model = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
        (
            "-----\n" +
            "____/      \\\n" +
            "|              |\n" +
            "----(@)-----(@)--- \n"
        );
        Console.WriteLine(Name);
    }
}
```

7. Когда все классы и интерфейсы определены и реализованы можно их использовать. Продемонстрируем использование типов данных созданием соответствующих объектов в функции main.

```

static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Clear();

    Console.Title = "Лабораторная работа №11";

    Human h = new Human("Иванов Иван Иванович", 50);
    h.Plus(5);
    h.Minus(1);
    h.DrawObject();

    Console.WriteLine("\n\n\n");

    Car car = new Car("Hyundai", "ix35", 120);
    car.Plus(25);
    car.Minus(11);
    car.DrawObject();

    Console.ReadKey();
}

```

8. Вид окна разработанного приложения представлен на рис. 9.1:

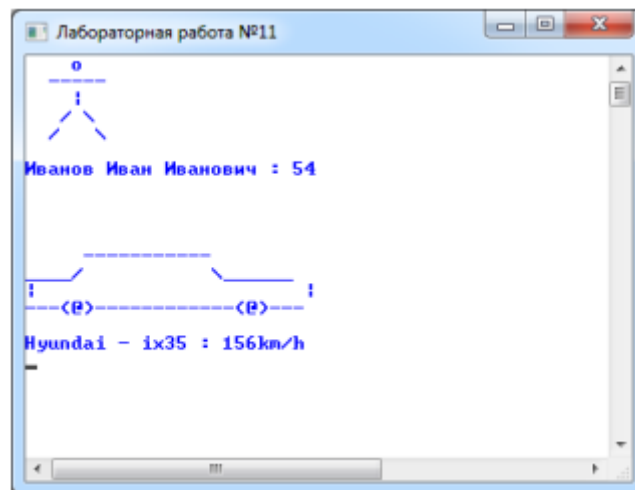


Рисунок 9.1 – Консольное приложение на основе интерфейсных типов

Диаграмма типов данных, созданная редактором Visual Studio, для разработанного приложения, показана на рис. 9.2.

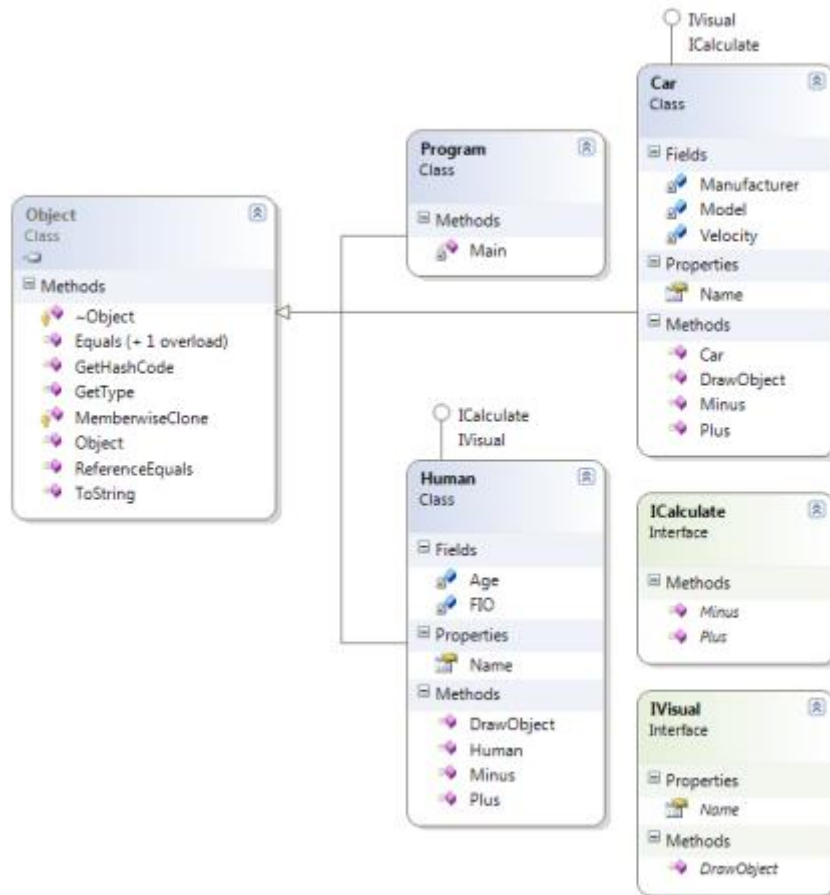


Рисунок 9.2 – Диаграмма классов приложения

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Спроектируйте классы, наполните их требуемой функциональностью, определите интерфейс, продемонстрируйте использование объектов класса и методов интерфейса. Постройте диаграмму классов своего приложения средствами Visual Studio или с использованием сторонних редакторов UML-диаграмм.

Продemonстрируйте явление полиморфизма при использовании классов из индивидуального задания.

## ВОПРОСЫ ДЛЯ ЗАЩИТЫ РАБОТЫ

1. Чем отличается наследование интерфейсов от наследования реализации?
2. Поясните, каким образом проявляется полиморфное поведение объектов при реализации классами интерфейсов?
3. Сколько интерфейсов может реализовывать класс?
4. Как объявляются интерфейсные типы? Для чего используются интерфейсы?

5. Даны определения нескольких типов. Укажите ошибки (если есть) в представленном фрагменте:

```
public class MainObject: IString, IDraw
{
    int data;

    public string GetStr()
    {
        return "Данные: " + data.ToString();
    }

    public void Draw()
    {
        Console.WriteLine(data.ToString());
    }
}

public interface IDraw
{
    void Draw();
}

public interface IString
{
    string GetStr();
}
```

6. Даны определения нескольких типов. Укажите ошибки (если есть) в представленном фрагменте:

<pre>public interface IDraw {     void Draw(); }  public interface IString {     string GetStr(); }  public class Parent {     protected int a; }</pre>	<pre>public class MainObject: IString, IDraw {     int data;      public string GetStr()     {         return "Данные: " + data.ToString();     }      public void Draw()     {         Console.WriteLine(data.ToString());     } }</pre>
---	---

```
static void Main(string[] args)
{
    MainObject ob1 = new MainObject();
    ob1.Draw();

    Parent ob2 = new Parent();
    ob2.Draw();
}
```

## ЛАБОРАТОРНАЯ РАБОТА 10. ОСНОВЫ РАБОТЫ С ФАЙЛАМИ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* научиться использовать механизмы файлового ввода-вывода.

Задачи лабораторной работы:

- научиться применять классы для работы с файлами;
- научиться применять классы для работы с каталогами;
- научиться использовать потоки ввода-вывода.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

**Классы .NET Framework для реализации операций ввода-вывода**

Весь ввод и вывод в .NET Framework подразумевает использование потоков. Поток – это абстрактное представление последовательного устройства. Последовательное устройство – это нечто такое, что хранит данные в линейной структуре и точно таким же образом обеспечивает доступ к ним: считывает или записывает по одному байту за одну единицу времени. Сохранение устройства абстрактным означает, что лежащие в основе источник/приемник данных могут быть скрыты. Такой уровень абстракции обеспечивает повторное использование кода и позволяет писать более обобщенные процедуры, потому что нет необходимости заботиться о действительной специфике передачи данных

Для обработки файлов в C# необходима ссылка на пространство имен System.IO. При открытии файла создается объект, с которым ассоциируется поток. Потоки обеспечивают механизмы связи между файлами и программами.

Среда .NET Framework предоставляет все необходимые инструменты для эффективного использования файлов в приложениях.

Основные классы, необходимые программисту:

1. `Object` – исходный базовый класс для всех классов платформы .NET Framework и корень иерархии типов.

2. `File` – статический служебный класс, предоставляющий множество статических методов, которые дают возможность перемещать, создавать, копировать, удалять файлы, опрашивать и обновлять атрибуты, а также создавать объекты потоков `FileStream`.

3. `Directory` – статический служебный класс, предоставляющий множество статических методов для перемещения, копирования и удаления каталогов.

4. `Path` – служебный класс, используемый для манипулирования путевыми именами

5. `MarshalByRefObject` – разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие, это базовый класс для всех классов `.NET`, позволяющих удаленное взаимодействие.

6. `FileInfo` – представляет физический файл на диске, имеет методы для манипулирования этим файлом. Для любого объекта, который читает или пишет в этот файл, должен быть создан объект `Stream`. Все методы `FileInfo` доступны из объектной переменной, поэтому, если необходимо выполнить только одно действие, более эффективным может оказаться использование метода `File`, а не соответствующего экземпляра метода `FileInfo`. Для всех методов `FileInfo` требуется путь к файлу, с которым проводится операция. Все статические методы класса `FileInfo` выполняют проверку безопасности для всех методов. Если необходимо использовать объект неоднократно, рекомендуется использовать соответствующий метод экземпляра `FileInfo`, поскольку в этом случае проверка безопасности будет требоваться не всегда.

7. `DirectoryInfo` – представляет физический каталог на диске и предоставляет методы уровня экземпляра для манипулирования каталогом. Класс `DirectoryInfo` работает точно так же, как класс `FileInfo`. Это объект, представляющий отдельный каталог на машине. Подобно классу `FileInfo`, многие из вызовов методов дублируются между `Directory` и `DirectoryInfo`.

8. `FileSystemInfo` – служит базовым классом для `FileInfo` и `DirectoryInfo`, обеспечивая возможность работы с файлами и каталогами одновременно, используя полиморфизм.

9. `Stream` – предоставляет универсальное представление последовательности байтов. Класс `Stream` является абстрактным базовым классом всех потоков.

10. `FileStream` – представляет файл, который может быть записан, прочитан или то и другое.

11. `TextReader` – представляет средство чтения, позволяющее считывать последовательные наборы знаков. Этот класс является абстрактным базовым классом для `StreamReader`, который считывает символы из потоков.

12. `TextWriter` – представляет средство записи, позволяющее записывать последовательные наборы символов. Этот класс является абстрактным базовым классом для `StreamWriter`, который записывают символы в потоки.

13. `StreamReader` – читает символьные данные из потока и может быть создан с использованием класса `FileStream` в качестве базового.

14. `StreamWriter` – пишет символьные данные в поток и может быть создан с использованием класса `FileStream` в качестве базового.

15. `Component` – предоставляет базовую реализацию интерфейса `IComponent` и делает возможным совместное использование объектов разными приложениями.

16. `FileSystemWatcher` – используется для мониторинга файлов и каталогов и представляет события, которые приложение может перехватить, когда в этих объектах происходят какие-то изменения.

Таким образом, эта система классов включает в себя классы для работы с файлами (`File`, `FileInfo`), каталогами (`Directory`, `DirectoryInfo`) и потоками (`FileStream`, `StreamReader`, `StreamWriter`).

В большинстве случаев для разработки бизнес-приложений достаточно лишь четырех классов для манипулирования файловой системой. Эти классы расположены в пространстве имен `System.IO` и предназначены для работы с файловой системой компьютера, то есть для создания, удаления переноса файлов и каталогов.

Первые два типа – `Directory` и `File` реализуют свои возможности с помощью статических методов, поэтому данные классы

можно использовать без создания соответствующих объектов (экземпляров классов).

Следующие типы – DirectoryInfo и FileInfo обладают схожими функциональными возможностями с Directory и File, но порождены от класса FileSystemInfo, поэтому реализуются путем создания соответствующих экземпляров классов.

Класс FileSystemInfo предоставляет базовый функционал. Значительная часть членов FileSystemInfo предназначена для работы с общими характеристиками файла или каталога (метками времени, атрибутами и т. п.). Рассмотрим некоторые свойства FileSystemInfo (таблица 10.1).

В FileSystemInfo предусмотрен набор методов. Например, метод Delete() – позволяет удалить объект файловой системы с жесткого диска, а Refresh() – обновить информацию об объекте файловой системы.

Таблица 10.1

Свойства класса FileSystemInfo

Свойство	Описание
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes
CreationTime	Позволяет получить или установить время создания объекта файловой системы
Exists	Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы
Extension	Позволяет получить расширение для файла
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы



Продолжение таблицы 10.1

Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определенное имя
------	---

В `FileSystemInfo` предусмотрен набор методов. Например, метод `Delete()` – позволяет удалить объект файловой системы с жесткого диска, а `Refresh()` – обновить информацию об объекте файловой системы.

### Классы для работы с каталогами файловой системы

Класс `DirectoryInfo` наследует члены класса `FileSystemInfo` и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные члены класса содержатся в таблице 10.2.

Таблица 10.2

Доступные члены класса `DirectoryInfo`

Член	Описание
<code>Create()</code> <code>CreateSubDirectory()</code>	Создают каталог (или подкаталог) по указанному пути в файловой системе
<code>Delete()</code>	Удаляет пустой каталог
<code>GetDirectories()</code>	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов <code>DirectoryInfo</code> )
<code>GetFiles()</code>	Позволяет получить доступ к файлам текущего каталога (в виде массива объектов <code>FileInfo</code> )
<code>MoveTo()</code>	Перемещает каталог и все его содержимое на новый адрес в файловой системе
<code>Parent</code>	Возвращает родительский каталог в иерархии файловой системы

Работа с типом `DirectoryInfo` начинается с того, что создается экземпляр класса (объект), при вызове конструктора в качестве параметра указывается путь к нужному каталогу. Если необходимо обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение `"."`. Например:

```
// Создаем объект DirectoryInfo,
// которому будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");

// Создаем объект DirectoryInfo,
// которому будет обращаться к каталогу d:\prim
DirectoryInfo dir2 = new DirectoryInfo(@"d:\prim");
```

Если создается объект `DirectoryInfo`, который связывается с несуществующим каталогом, то будет сгенерировано исключение `System.IO.DirectoryNotFoundException`.

Свойство `Attributes` класса `DirectoryInfo` позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице 10.3.

Таблица 10.3

Значения свойства `Attributes`

Значение	Описание
Archive	Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях - удаления старых файлов
Compressed	Определяет, что файл является сжатым
Directory	Определяет, что объект файловой системы является каталогом
Encrypted	Определяет, что файл является зашифрованным
Hidden	Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога)
Normal	Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может использоваться с другими атрибутами

## Продолжение таблицы 10.3

Offline	Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели
Readonly	Файл доступен только для чтения
System	Файл является системным (то есть файл является частью операционной системы или используется исключительно операционной системой)

Через класс DirectoryInfo программист может собрать информацию о дочерних подкаталогах. Например:

```
static void printDirect(DirectoryInfo dir)
{
    Console.WriteLine("***** " + dir.Name + " *****");
    Console.WriteLine("FullName: {0}", dir.FullName);
    Console.WriteLine("Name: {0}", dir.Name);
    Console.WriteLine("Parent: {0}", dir.Parent);
    Console.WriteLine("Creation: {0}", dir.CreationTime);
    Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
    Console.WriteLine("Root: {0}", dir.Root);
}

static void Main(string[] args)
{
    DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
    printDirect(dir);
    DirectoryInfo[] subDirects = dir.GetDirectories();
    Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length);
    foreach (DirectoryInfo d in subDirects)
    {
        printDirect(d);
    }
}
```

Метод CreateSubdirectory() позволяет создать в выбранном каталоге как единственный подкаталог, так и множество подкаталогов (в том числе, и вложенных друг в друга). Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim");

//создали подкаталог
dir.CreateSubdirectory("doc");

//создали вложенный подкаталог
dir.CreateSubdirectory(@"book\2008");
```

Метод MoveTo() позволяет переместить текущий каталог по заданному в качестве параметра адресу. При этом возможно произвести переименование каталога. Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
dir.MoveTo(@"d:\prim\letter\Николаев");
```

В данном случае каталог «bmr» перемещается по адресу «d:\prim\letter\Николаев». Так как имя перемещаемого каталога не совпадает с крайним правым именем в адресе нового местоположения каталога, то производится переименование.

Работать с каталогами файловой системы компьютера можно и при помощи класса `Directory`, функциональные возможности которого во многом совпадают с возможностями `DirectoryInfo`. Следует учитывать, что члены данного класса реализованы статически, поэтому для их использования нет необходимости создавать объект. Например:

```
// Создание подкаталога Новый
Directory.CreateDirectory(@"d:\prim\Новый");

// Перемещение каталога Новый в каталог 2012
Directory.Move(@"d:\prim\Новый",
               @"d:\prim\2012\Новый");

// переименовали каталог Новый в Старый
Directory.Move(@"d:\prim\2012\Новый",
               @"d:\prim\2012\Старый");
```

Следует учитывать, что удаление каталога возможно только когда он пуст.

На практике комбинируют использование классов `Directory` и `DirectoryInfo`.

### Классы для работы с файлами

Класс `FileInfo` предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах), а также производить различные операции, например, по созданию файла или его удалению. Класс `FileInfo` наследует члены класса `FileSystemInfo` и содержит дополнительный набор членов, который приведен в следующей таблице 10.4.

Таблица 10.4

Члены класса `FileInfo`

Член	Описание
<code>AppendText()</code>	Создает объект <code>StreamWriter</code> для добавления текста к файлу
<code>CopyTo()</code>	Копирует уже существующий файл в новый файл.

Продолжение таблицы 10.4

Create()	Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом.
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл.
Delete()	Удаляет файл, которому соответствует объект FileInfo.
Directory	Возвращает каталог, в котором расположен данный файл.
DirectoryName	Возвращает полный путь к данному файлу в файловой системе.
Length	Возвращает размер файла.
MoveTo()	Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл).
Name	Позволяет получить имя файла.
Open()	Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями.
OpenRead()	Создает объект FileStream, доступный только для чтения.
OpenText()	Создает объект StreamReader (о нем также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла.
OpenWrite()	Создает объект FileStream, доступный для чтения и записи.

Большинство методов FileInfo возвращает объекты классов FileStream, StreamWriter, StreamReader и т. п., которые позволяют различным образом взаимодействовать с файлом, например, производить чтение или запись в него. Например:

```

// Создание объекта FileInfo
FileInfo f = new FileInfo("text.txt");

// Создание файла и потока
StreamWriter fOut =
    new StreamWriter(f.Create());

// Запись текста в файл
fOut.WriteLine("ОДИН ДВА ТРИ...");

// Закрытие файла
fOut.Close();

// Получение информации о файле
Console.WriteLine("Name: {0}", f.Name);
Console.WriteLine("File size: {0}",
    f.Length);
Console.WriteLine("Creation: {0}",
    f.CreationTime);
Console.WriteLine("Attributes: {0}",
    f.Attributes.ToString());

```

Доступ к физическим файлам можно получать и через статические методы класса `File`. Большинство методов объекта `FileInfo` представляют в этом смысле зеркальное отражение методов объекта `File`.

### **Потоки в системе ввода-вывода**

Программы на языке C# выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) – это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные из различных источников (клавиатура, файл, память) и записывать в различные источники (принтер, экран, файл, память).

Центральную часть потоковой системы C# занимает класс `Stream` пространства имен `System.IO`. Класс `Stream` представляет байтовый поток и является базовым для всех остальных потоковых классов. Производными от класса `Stream` являются классы потоков:

1. `FileStream` – байтовый поток, разработанный для файлового ввода-вывода
2. `BufferedStream` – заключает в оболочку байтовый поток и добавляет буферизацию, которая во многих случаях увеличивает производительность программы.

3. `MemoryStream` – байтовый поток, который использует память для хранения данных

Программист может реализовать собственные потоковые классы. Однако для подавляющего большинства приложений достаточно встроенных потоков.

### **Байтовый поток**

Чтобы создать байтовый поток, связанный с файлом, создается объект класса `FileStream`. При этом в классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и (или) записи:

```
public FileStream(string path, FileMode mode);
```

Параметр `path` определяет имя файла, с которым будет связан поток ввода-вывода данных. Параметр `mode` определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением `FileMode`:

- `FileMode.Append` – предназначен для добавления данных в конец файла;
- `FileMode.Create` – предназначен для создания нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;
- `FileMode.CreateNew` – предназначен для создания нового файла, при этом файл с таким же именем не должен существовать;
- `FileMode.Open` – предназначен для открытия существующего файла;
- `FileMode.OpenOrCreate` – если файл существует, то открывает его, в противном случае создает новый;
- `FileMode.Truncate` – открывает существующий файл, но усекает его длину до нуля.

Если попытка открыть файл оказалась неудачной, то генерируется одно из исключений:

- `FileNotFoundException` – файл невозможно открыть по причине его отсутствия;
- `IOException` – файл невозможно открыть из-за ошибки ввода-вывода;
- `ArgumentNullException` – имя файла представляет собой `null`-значение;



- `ArgumentException` – некорректен параметр `mode`;
- `SecurityException` – пользователь не обладает правами доступа;
- `DirectoryNotFoundException` – некорректно задан каталог.

Другая версия конструктора позволяет ограничить доступ только чтением или только записью:

```
public FileStream(string path, FileMode mode, FileAccess access);
```

Параметры `path` и `mode` имеют то же назначение, что и в предыдущей версии конструктора. Параметр `access`, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением `FAccess`:

- `FAccess.Read` – только чтение;
- `FAccess.Write` – только запись;
- `FAccess.ReadWrite` – и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод `ReadByte( )`. После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достигнут конец файла, то метод `ReadByte( )` возвращает значение -1.

Для побайтовой записи данных в поток используется метод `WriteByte( )`.

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод `Close( )`. При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.



```

static void Main(string[] args)
{
    try
    {
        // Создание потока для чтения из файла
        FileStream fileIn =
            new FileStream("ФайлСТекстом.txt",
                FileMode.Open,
                FileAccess.Read);
        // Создание потока для записи в файл
        FileStream fileOut =
            new FileStream("ФайлКопия.txt",
                FileMode.Create,
                FileAccess.Write);
        int i;

        // В цикле производится чтение одного файла
        // и одновременная запись содержимого
        // во второй файл
        while ((i = fileIn.ReadByte()) != -1)
        {
            // запись очередного байта в поток
            fileOut.WriteByte((byte)i);
        }

        // Закройте файлы
        fileIn.Close();
        fileOut.Close();
    }
    catch (Exception EX)
    {
        Console.WriteLine(EX.Message);
    }
}

```

В данном примере создаются два потока fileIn (для чтения байтов из файла в поток) и fileOut (для записи байтов из потока в файл). Каждый из потоков ассоциируется со своим файлом. Затем в цикле производится побайтовое чтение файла «ФайлСТекстом.txt» до момента, когда функция ReadByte( ) вернет значение -1 (то есть достигнут конец файла). При этом на каждой итерации цикла считывается один байт и на этой же итерации этот байт записывается в файл «ФайлКопия.txt». После всех операция оба потока закрываются. Весь код, осуществляющий работу с файлами заключен в конструкцию try ... catch. Это позволяет повысить устойчивость программы к ошибкам.

### **Символьный поток**

Чтобы создать символьный поток нужно поместить объект класса Stream (например FileStream) внутрь объекта класса StreamWriter или объекта класса StreamReader. В этом случае

байтовый поток будет автоматически преобразовываться в символьный.

Класс `StreamWriter` предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр `stream` определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа `ArgumentException`, если поток `stream` не открыт для вывода, и исключение типа `ArgumentNullException`, если он (поток) имеет `null`-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр `path` определяет имя открываемого файла.

Например, создать экземпляр класса `StreamWriter` можно следующим образом:

```
StreamWriter fileOut =  
    new StreamWriter(  
        new FileStream(  
            "ФайлНиколаева.txt",  
            FileMode.Create,  
            FileAccess.Write));
```

И еще один вариант конструктора `StreamWriter`:

```
public StreamWriter(string path, bool append);
```

Параметр `path` определяет имя открываемого файла, а параметр `append` может принимать значение `true` – если нужно добавлять данные в конец файла, или `false` – если файл необходимо перезаписать.

Например:

```
// Добавляем символы в конец файла  
StreamWriter fileOut_1 =  
    new StreamWriter("МойФ.txt", true);
```

Объявив таким образом переменную `fileOut_1` для записи данных в поток можно обратиться к методу `WriteLine`:

```
// Использование WriteLine  
fileOut_1.WriteLine("Строка для записи");
```

В данном случае для записи используется метод, аналогичный статическому методу класса `Console`. Это действительно схожие механизмы ввода-вывода.

Класс `StreamReader` предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр `stream` определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа `ArgumentException`, если поток `stream` не открыт для ввода. Создать экземпляр класса `StreamReader` можно следующим образом:

```
StreamReader fileIn =  
    new StreamReader(  
        new FileStream(  
            "text.txt",  
            FileMode.Open,  
            FileAccess.Read));
```

Как и в случае с классом `StreamWriter` у класса `StreamReader` есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр `path` определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =  
    new StreamReader(  
        @"c:\temp\Николаев.txt");
```

В C# символы реализуются кодировкой `Unicode`. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора `StreamReader`:

```
StreamReader fileIn =  
    new StreamReader(  
        @"c:\temp\t.txt",  
        Encoding.GetEncoding(1251));
```

Параметр `Encoding.GetEncoding(1251)` говорит о том, что будет выполняться преобразование из кода `Windows-1251` (одна из модификаций кода `ASCII`, содержащая русские символы) в `Unicode`. Тип `Encoding` реализован в пространстве имен `System.Text`.

Для чтения данных из потока `fileIn` можно воспользоваться методом `ReadLine`. При этом если будет достигнут конец файла, то метод `ReadLine` вернет значение `null`.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов `StreamWriter` и `StreamReader`.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```

В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

### **Перенаправление стандартных потоков**

Тремя стандартными потоками, доступ к которым осуществляется через свойства `Console.Out`, `Console.In` и `Console.Error`, могут пользоваться все программы, работающие в пространстве имен `System`. Свойство `Console.Out` относится к стандартному выходному потоку. По умолчанию это консоль. Например, при вызове метода `Console.WriteLine()` информация автоматически передается в поток `Console.Out`. Свойство

Console.In относится к стандартному входному потоку, источником которого по умолчанию является клавиатура. Например, при вводе данных с клавиатуры информация автоматически передается потоку Console.In, к которому можно обратиться с помощью метода Console.ReadLine(). Свойство Console.Error относится к ошибкам в стандартном потоке, источником которого также по умолчанию является консоль. Однако эти потоки могут быть перенаправлены на любое совместимое устройство ввода/вывода, например, на работу с физическими файлами.

Перенаправить стандартный поток можно с помощью методов SetIn(), SetOut() и SetError(), которые являются членами класса Console:

```
static void SetIn(TextReader input);
static void SetOut(TextWriter output);
static void SetError(TextWriter output);
```

Пример перенаправления потоков представлен в следующей программе, демонстрирующей, что стандартный поток вывода перенаправляется в один файл, а поток ввода – в другой:

```
static void Main(string[] args)
{
    int[,] MyArray;
    StreamReader file = new StreamReader("input.txt");
    // Перенаправление на file
    Console.SetIn(file);
    string line = Console.ReadLine();
    string[] mas = line.Split(' ');
    int n = int.Parse(mas[0]);
    int m = int.Parse(mas[1]);
    MyArray = new int[n, m];
    for (int i = 0; i < n; i++)
    {
        line = Console.ReadLine();
        mas = line.Split(' ');
        for (int j = 0; j < m; j++)
        {
            MyArray[i, j] = int.Parse(mas[j]);
        }
    }
    PrintArray("Исходный массив:", MyArray, n, m);
    file.Close();
}

static void PrintArray(string a, int[,] mas, int n, int m)
{
    // Перенаправление на file
    StreamWriter file = new StreamWriter("output.txt");
    Console.SetOut(file);
    Console.WriteLine(a);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++) Console.Write("{0} ", mas[i, j]);
        Console.WriteLine();
    }
    file.Close();
}
```

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Учебная задача

Для выполнения лабораторной работы необходимо спроектировать многомодульное приложение, использующее файлы для ввода и вывода информации.

Например, требуется разработать приложение, которое позволяет:

- генерировать матрицу случайных чисел с сохранением ее в файл;
- считывать матрицу из файла;
- выводить матрицу на экран.

Для выполнения данного задания-примера достаточно в отдельном проекте реализовать класс. Этот проект следует скомпилировать в виде dll-файла. Затем, в основной программе просто необходимо использовать данную библиотеку.

Класс библиотеки назовем Matrix. В нем определим следующие функции:

```
class Matrix
{
    private float[,] matrix;
    int m, n;

    // Конструктор
    public Matrix()...

    // Генерация матрицы заданного размера
    public void GenerateMatrix(int M, int N)...

    // Сохранение сгенерированной матрицы в файл
    public void SaveMatrix(string pFileName)...

    // Загрузка сохраненной матрицы из файла
    public Boolean LoadMatrix(string pFileName)...

    // Вывод матрицы в консоль
    public void PrintMatrix()...
}
```

Рисунок 10.1 – Основные методы класса Matrix

Листинг метода GenegateMatrix:



```
// Генерация матрицы заданного размера
public void GenerateMatrix(int M, int N)
{
    m = M; n = N;

    Random r = new Random(DateTime.Now.Millisecond);

    matrix = new float[M, N];

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            matrix[i, j] = (float)r.Next(1000) / 973f;
}
```

Рисунок 10.2 – Метод для создания матрицы заданного размера и заполнения ее случайными числами

### Листинг метода SaveMatrix:

```
// Сохранение сгенерированной матрицы в файл
public void SaveMatrix(string pFileName)
{
    if (matrix.Length > 0)
    {
        if (File.Exists(pFileName))
            File.Delete(pFileName);

        FileInfo f = new FileInfo(pFileName);
        TextWriter tw = f.CreateText();

        tw.WriteLine(m.ToString());
        tw.WriteLine(n.ToString());

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                tw.WriteLine(i.ToString() + " " + j.ToString() + " " + matrix[i, j].ToString("E10"));

        tw.Close();
    }
}
```

Рисунок 10.3 – Метод для сохранения матрицы в файл

```
// Загрузка сохраненной матрицы из файла
public Boolean LoadMatrix(string pFileName)
{
    if (File.Exists(pFileName))
    {
        try
        {
            TextReader tr = File.OpenText(pFileName);
            m = Convert.ToInt32(tr.ReadLine());
            n = Convert.ToInt32(tr.ReadLine());

            matrix = new float[m, n];
            string line;
            string[] substring;

            for (int i = 0; i < m; i++)
                for (int j = 0; j < n; j++)
                {
                    line = tr.ReadLine();
                    substring = line.Split(new char[] { ' ' }, 3);
                    matrix[i, j] = Convert.ToSingle(substring[2]);
                }

            tr.Close();

            return true;
        }
        catch
        {
            return false;
        }
    }

    return false;
}
```

Рисунок 10.4 – Метод загрузки матрицы из файла

На рис. 10.4 представлен листинг метода LoadMatrix. Листинг метода PrintMatrix:

```
// Вывод матрицы в консоль
public void PrintMatrix()
{
    if (matrix.Length > 0)
    {
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
                Console.Write(matrix[i, j].ToString("E3") + " ");

            Console.WriteLine();
        }
    }
}
```

Рисунок 10.5 – Метод для вывода матрицы

Основная программа для использования класса-матрицы представлена на рис. 10.6



```

class Program
{
    static void Main(string[] args)
    {
        Console.BackgroundColor = ConsoleColor.White;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Clear();

        Matrix m = new Matrix();

        m.GenerateMatrix(10, 5);
        m.SaveMatrix("FileForMatrix.txt");

        if (m.LoadMatrix("FileForMatrix.txt"))
            m.PrintMatrix();

        Console.ReadKey();
    }
}

```

Рисунок 10.6 – Работа с библиотечным классом

В результате работы данного кода будет создан файл FileForMatrix.txt следующего содержания:

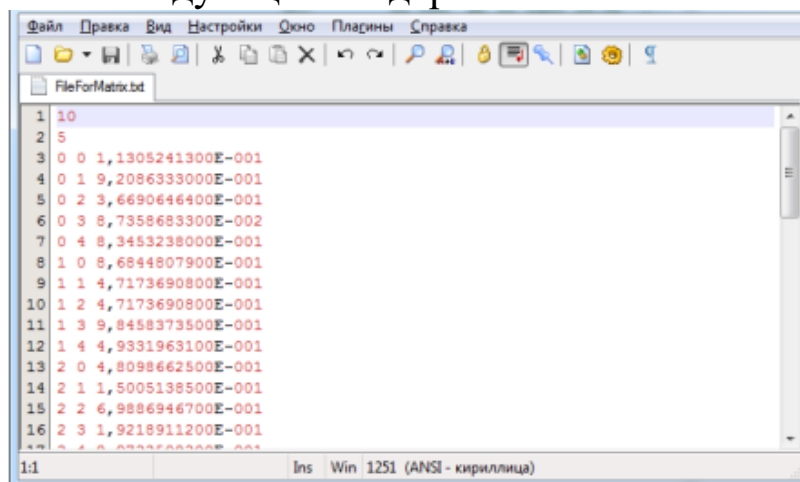


Рисунок 10.7 – Результирующий файл

На экран будет выведена следующая информация:

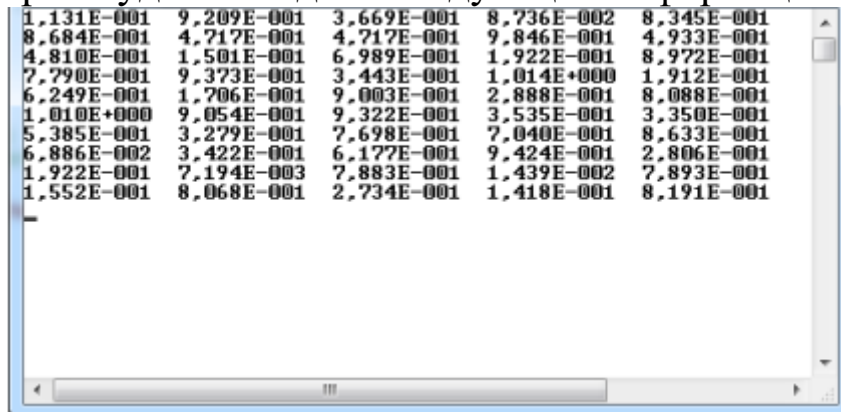


Рисунок 10.8 – Вывод программы в консоль

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Выполните индивидуальное задание, согласно предложенному варианту. В каждом варианте необходимо спроектировать многомодульное приложение (оптимально 2 модуля). Исходные данные в каждом варианте программа получает из входного файла.

Таблица 10.5

Индивидуальные задания

Вариант	Задание
1	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
2	Программа рассчитывает сумму двух матриц, которые хранятся в разных файлах.
3	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.
4	Программа рассчитывает сумму диагональных элементов двух матриц, которые хранятся в разных файлах.
5	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
6	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.
7	Программа рассчитывает сумму элементов главной диагонали первой матрицы и сумму элементов второстепенной диагонали второй матрицы. Матрицы хранятся в разных файлах.
8	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
9	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
10	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.

Продолжение таблицы 10.5

11	Программа рассчитывает сумму элементов, находящихся в четном столбце и нечетной строке, в двух матрицах, которые хранятся в разных файлах.
12	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
13	Программа рассчитывает произведение элементов в двух матрицах, которые хранятся в разных файлах.
14	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
15	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
16	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
17	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.
18	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
19	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
20	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
21	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
22	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.

## Продолжение таблицы 10.5

23	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
24	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
25	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.
26	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах. Для каждой матрицы (двух исходных и результирующей) выводится среднее арифметическое ее элементов.
27	Программа рассчитывает сумму и разность двух матриц, которые хранятся в разных файлах.
28	Программа находит минимальный, максимальный элементы в двух матрицах, которые хранятся в разных файлах.
29	Программа рассчитывает сумму диагональных элементов двух матриц, которые хранятся в разных файлах.
30	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
31	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.
32	Программа рассчитывает сумму элементов главной диагонали первой матрицы и сумму элементов второстепенной диагонали второй матрицы. Матрицы хранятся в разных файлах.
33	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
34	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах

## Продолжение таблицы 10.5

35	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
----	--

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Какие классы для работы с файловой системой вы знаете?
2. Что такое сборка?
3. Как определить проект по умолчанию в решении Visual Studio?
4. Какие классы отвечают за представление файлов в программе?
5. Что такое поток? Какие типы классов потоков используются при работе с файлами?
6. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов.
7. Перечислите классы для работы с каталогами.
8. Поясните принцип работы синтаксической конструкции `try ... catch`.

## ЛАБОРАТОРНАЯ РАБОТА 11. ОБРАБОТКА ДАННЫХ В ФАЙЛАХ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* научиться использовать механизмы файлового ввода-вывода.

Задачи лабораторной работы:

- научиться применять классы для работы с файлами;
- научиться применять классы для работы с каталогами;
- научиться использовать потоки ввода-вывода.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Повторить теоретический материал лабораторной работы 10. При выполнении данной лабораторной работы рекомендуется использовать символьный поток.

Чтобы создать символьный поток нужно поместить объект класса `Stream` (например `FileStream`) внутрь объекта класса `StreamWriter` или объекта класса `StreamReader`. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс `StreamWriter` предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр `stream` определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа `ArgumentException`, если поток `stream` не открыт для вывода, и исключение типа `ArgumentNullException`, если он (поток) имеет `null`-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр `path` определяет имя открываемого файла.

Например, создать экземпляр класса `StreamWriter` можно следующим образом:

```

StreamWriter fileOut =
    new StreamWriter(
        new FileStream(
            "ФайлНиколаева.txt",
            FileMode.Create,
            FileAccess.Write));

```

И еще один вариант конструктора StreamWriter:

```
public StreamWriter(string path, bool append);
```

Параметр path определяет имя открываемого файла, а параметр append может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```

// Добавляем символы в конец файла
StreamWriter fileOut_1 =
    new StreamWriter("МойФ.txt", true);

```

Объявив таким образом переменную fileOut\_1 для записи данных в поток можно обратиться к методу WriteLine:

```

// Использование WriteLine
fileOut_1.WriteLine("Строка для записи");

```

В данном случае для записи используется метод, аналогичный статическому методу класса Console. Это действительно схожие механизмы ввода-вывода.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода. Создать экземпляр класса StreamReader можно следующим образом:

```

StreamReader fileIn =
    new StreamReader(
        new FileStream(
            "text.txt",
            FileMode.Open,
            FileAccess.Read));

```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр `path` определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =
    new StreamReader
        (@":\temp\Николаев.txt");
```

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора `StreamReader`:

```
StreamReader fileIn =
    new StreamReader(
        @":\temp\t.txt",
        Encoding.GetEncoding(1251));
```

Параметр `Encoding.GetEncoding(1251)` говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Тип `Encoding` реализован в пространстве имен `System.Text`.

Для чтения данных из потока `fileIn` можно воспользоваться методом `ReadLine`. При этом если будет достигнут конец файла, то метод `ReadLine` вернет значение `null`.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов `StreamWriter` и `StreamReader`.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```



В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Учебная задача

Условие. На рисунке 11.1 представлен фрагмент файла `data_example.csv`, который содержит данные о пользователях в следующем формате: ID, Fname, Lname, Email, Gender, Salary, HasChildren, где ID – идентификатор, FName – имя, Lname – фамилия, Email – адрес электронной почты, Gender – пол, Salary – зарплата, HasChildren – есть дети или нет. Все поля разделены запятыми, данные каждого пользователя начинаются с новой строки. Пропусков в данных нет. Файл содержит около 1000 строк. Первая строка – заголовок. Необходимо ответить на следующие вопросы:

1. определить количество мужчин и женщин;
2. найти самую высокую зарплату среди женщин и среди мужчин (вывести этих пользователей);
3. посчитать количество пользователей с детьми и без;
4. посчитать суммарный фонд заработной платы для четырех категорий, образованных признаками Gender и HasChildren.

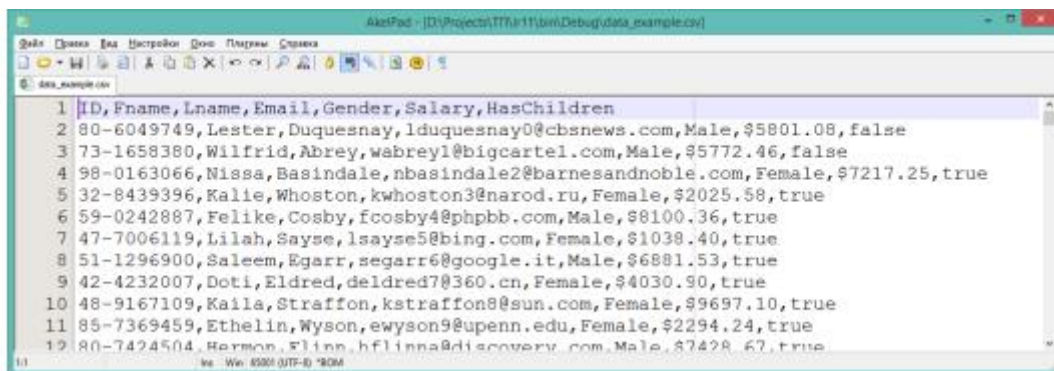


Рисунок 11.1 – Фрагмент исходного файла

Решение. Для отработки алгоритмов необходимо создать копию исходного файла данных, которая будет содержать 10-20 строк. На этом файле следует проводить разработку и тестирование приложения.

Спроектируем класс Person для представления одной строки файла. Проект разработанного класса показан на рисунке 11.2. Для представления поля Gender создадим тип-перечисление GenderType, для конвертации текстовой строки из файла в объект типа Person объявим метод Create(). Объявление основных типов реализовано в файле Person.cs (рисунок 11.3)

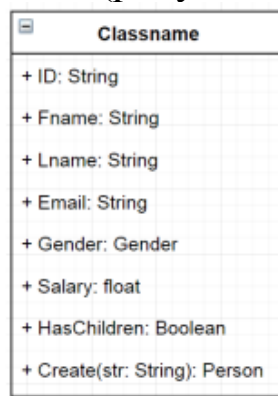


Рисунок 11.2 – Диаграмма UML для класса Person

```

1  using System;
2
3  namespace lr11
4  {
5      enum GenderType
6      {
7          Male,
8          Female
9      }
10
11     class Person
12     {
13         String ID { get; set; }
14         String FName { get; set; }
15         String Lname { get; set; }
16         String Email { get; set; }
17         public GenderType Gender { get; set; }
18         public float Salary { get; set; }
19         public Boolean HasChildren { get; set; }
20
21         Person Create(String str)
22         {
23             Person p = new Person();
24
25             return p;
26         }
27     }
28 }
  
```

Рисунок 11.3 – Листинг Person.cs

Теперь реализуем набросок главной программы (рисунок 11.4).

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.IO;
5
6  namespace lr11
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             StreamReader f_in = new StreamReader("data_example_small.csv");
13             #if !DEBUG
14                 TextWriter save_out = Console.Out;
15                 var new_out = new StreamWriter(@"lr11_output.txt");
16                 Console.SetOut(new_out);
17             #endif
18
19             List<Person> all = new List<Person>();
20             String line = f_in.ReadLine();
21             while((line = f_in.ReadLine()) != null)
22             {
23                 Console.WriteLine(line);
24             }
25
26             #if !DEBUG
27                 Console.SetOut(save_out);
28                 new_out.Close();
29             #else
30                 Console.ReadKey();
31             #endif
32         }
33     }
34 }

```

Рисунок 11.4 – Листинг модуля Program.cs с функцией Main()

Данный набросок программы выводит все считанный из файла строки на консоль (в режиме DEBUG). На рисунке 11.5 показан результат работы приложения в режиме DEBUG.

```

D:\Projects\T1\lr11\bin\Debug\lr11.exe
80-6049749, Lester, Duquesnay, lduquesnay@cbcsnews.com, Male, $5801.08, false
73-1658380, Hilfrid, Abrey, wabrey1@bigcartel.com, Male, $5772.46, false
98-0163066, Nissa, Basindale, nbasindale2@barnesandnoble.com, Female, $7217.25, true
32-8439396, Kallie, Whoston, kwhoston3@narod.ru, Female, $2025.58, true
59-0242887, Felike, Cosby, fcosby4@phpbb.com, Male, $8100.36, true
47-7006119, Lilah, Sayse, lsayse5@bing.com, Female, $1038.40, true
51-1296900, Saleem, Egarr, segarr6@google.it, Male, $6881.53, true
42-4232007, Doti, Eldred, deldred7@360.cn, Female, $4030.90, true
48-9167109, Kailla, Straffon, kstraffon8@sun.com, Female, $9697.10, true

```

Рисунок 11.4 – Листинг модуля Program.cs с функцией Main()

Теперь можно доработать метод Person.Create() для конвертации строки входного файла в объект типа Person (рисунок 11.5). В этом же листинге добавлены вспомогательные методы для удобства представления класса Person.

```

11 class Person
12 {
13     String ID { get; set; }
14     String FName { get; set; }
15     String Lname { get; set; }
16     String Email { get; set; }
17     public GenderType Gender { get; set; }
18     public float Salary { get; set; }
19     public Boolean HasChildren { get; set; }
20
21     public static Person Create(String str)
22     {
23         Person p = new Person();
24         string[] e = str.Split(',');
25         p.ID = e[0].Trim();
26         p.Fname = e[1].Trim();
27         p.Lname = e[2].Trim();
28         p.Email = e[3].Trim();
29         String tmp = e[4].Trim();
30         if (tmp == "Male")
31             p.Gender = GenderType.Male;
32         else p.Gender = GenderType.Female;
33         p.Salary = Convert.ToSingle(e[5].TrimStart('$').Replace('.', ','));
34         p.HasChildren = Convert.ToBoolean(e[6].Trim());

```

```

35
36     return p;
37 }
38
39 public override string ToString()
40 {
41     String s = string.Format(
42         "*****\n" +
43         "ID: {0}, Имя: {1}, Фамилия: {2}, ({3})\n" +
44         "E-mail: {4}, З/п: {5}, Дети: {6}",
45         ID, Fname, Lname, GenderToStr(Gender),
46         Email, Salary, HasChildrenToStr(HasChildren));
47
48     return s;
49 }
50
51 private static String GenderToStr(GenderType g)
52 {
53     if (g == GenderType.Male) return "М";
54     else return "Ж";
55 }
56
57 private static String HasChildrenToStr(Boolean g)
58 {
59     if (g == true) return "Есть";
60     else return "Нет";
61 }
62 }
63

```

Рисунок 11.4 – Доработка класса Person

Модифицируем метод Main(), теперь он позволяют заполнить список объектов типа Person (рисунок 11.5) и вывести список объектов в консоль в режиме debug.

В листинге представлен фрагмент, отражающий работу с коллекцией объектов Person. Работа программы на данном этапе разработке представлена на рисунке 11.6.

После подготовительных действий можно перейти к решению заданий учебной задачи.

1. Найдем количество пользователей мужского и женского пола. Для этого в метод Main() добавим следующий код (рисунок 11.7)

```

18 List<Person> all = new List<Person>();
19
20 try
21 {
22
23     String line = f_in.ReadLine();
24     while((line = f_in.ReadLine()) != null)
25     {
26         all.Add(Person.Create(line));
27     }
28 } catch (Exception ex)
29 {
30     Console.WriteLine(ex.Message);
31 }
32
33 Console.WriteLine("Всего пользователей: {0}", all.Count);
34 foreach (var p in all)
35     Console.WriteLine(p);

```

Рисунок 11.5 – Доработка метода Main()



Рисунок 11.6 – Демонстрация работы программы на этапе разработанного функционала класса Person

```

Console.WriteLine("***** Задача 1 *****");
int mCount = all.FindAll(p => p.Gender == GenderType.Male).ToList().Count;
Console.WriteLine("Количество мужчин: {0}", mCount);
Console.WriteLine("Количество женщин: {0}", all.Count - mCount);

```

Рисунок 11.7 – Вычисление количества пользователей мужского и женского пола

Таким образом, первая задача решена в 3 строки. Переходим к решению второй задачи.

2. Требуется найти самую высокую зарплату среди женщин и среди мужчин (вывести этих пользователей). Для нахождения этих величин используем код, представленный на рисунке 11.8.



```

Console.WriteLine("***** Задача 2 *****");
float male_max = (from p in all
    where p.Gender == GenderType.Male
    select p.Salary).Max();
Person rich_man = (from p in all
    where (p.Gender == GenderType.Male) &&
    (p.Salary == male_max) select p).First();
Console.WriteLine(rich_man);

float female_max = (from p in all
    where p.Gender == GenderType.Female
    select p.Salary).Max();
Person rich_woman = (from p in all
    where (p.Gender == GenderType.Female) &&
    (p.Salary == female_max)
    select p).First();
Console.WriteLine(rich_woman);

```

Рисунок 11.8 – Вычисление максимальной зарплаты для женщин и мужчин

Для решения задач используется язык LINQ, который является встроенным «подъязыком» технологии .NET. Можно решить задачи классическими алгоритмами с использованием циклов. Переходим к заданию 3.

3. Требуется посчитать количество пользователей с детьми и без. Листинг для данной задачи представлен на рисунке 11.9. В данном случае решаем классическим перебором всей коллекции объектов Person.

```

Console.WriteLine("\n***** Задача 3 *****");
int Count_Yes = 0, Count_No = 0;
for (int i = 0; i < all.Count; i++)
{
    if (all[i].HasChildren) Count_Yes++;
    else Count_No++;
}
Console.WriteLine("С детьми: {0}, без детей {1}",
    Count_Yes, Count_No);

```

Рисунок 11.8 – Вычисление количества пользователей с детьми и без

4. Решим последнюю задачу. Требуется посчитать суммарный фонд заработной платы для четырех категорий, образованных признаками Gender и HasChildren. То есть нужно разбить всех пользователей на 4 группы (рисунок 11.9) и в каждой посчи-

тать суммарный фонд заработной платы. В данной задаче дополнительно выведем суммарный фонд заработной платы по всем пользователям. Решение представлено на рисунке 11.10

	Есть дети	Нет детей
Мужчины		
Женщины		

Рисунок 11.9 – Группы, на которые разбивает всех пользователей признаки Gender и HasChildren

```

Console.WriteLine("\n***** Задача 4 *****");
float salary_total = (from p in all
                      select p.Salary).Sum();
float m_haschildren = (from p in all
                      where (p.Gender == GenderType.Male) && p.HasChildren
                      select p.Salary).Sum();
float f_haschildren = (from p in all
                      where (p.Gender == GenderType.Female) && p.HasChildren
                      select p.Salary).Sum();
float m_nochildren = (from p in all
                      where (p.Gender == GenderType.Male) && !p.HasChildren
                      select p.Salary).Sum();
float f_nochildren = (from p in all
                      where (p.Gender == GenderType.Female) && !p.HasChildren
                      select p.Salary).Sum();
var f = System.Globalization.CultureInfo.GetCultureInfo("en-us");
Console.WriteLine("Фонд з/п: {0}\n" +
    "Муж.+дети: {1},\t Жен.+дети: {2},\n" +
    "Муж.-дети: {3},\t Жен.-дети: {4}",
    salary_total.ToString("C3", f),
    m_haschildren.ToString("C", f), f_haschildren.ToString("C", f),
    m_nochildren.ToString("C", f), f_nochildren.ToString("C", f));

```

Рисунок 11.10 – Расчет фондов заработной платы

В результате всех доработок debug-программа выводит в консоль следующие данные, которые представляют собой решение подзадач в учебной задаче:



```

***** Задача 1 *****
Количество мужчин: 4
Количество женщин: 5

***** Задача 2 *****
ID: 59-0242887, Имя: Felike, Фамилия: Cosby, (М)
E-mail: fcosby4@phpbb.com, З/п: 8100,36, Дети: Есть
ID: 48-9167109, Имя: Kaila, Фамилия: Straffon, (Ж)
E-mail: kstraffon8@sun.com, З/п: 9697,1, Дети: Есть

***** Задача 3 *****
С детьми: 7, Без детей 2

***** Задача 4 *****
Фонд з/п: $50,564.660
Муж.+дети: $14,981.89, Жен.+дети: $24,009.23,
Муж.-дети: $11,573.54, Жен.-дети: $0.00

```

Рисунок 11.11 – Вывод приложения в режиме debug

После разработки debug-версии приложения можно перейти к сборке releasesборки и замене ограниченного, тестового набора данных на полный набор данных, содержащий большое количество записей. На рисунке 11.12 представлен выходной файл lr11\_output.txt, содержащий вывод release-сборки.

```

1 ***** Задача 1 *****
2 Количество мужчин: 495
3 Количество женщин: 505
4
5 ***** Задача 2 *****
6
7 ID: 14-3776103, Имя: Granny, Фамилия: Dresse, (М)
8 E-mail: gdressepu@bloglovin.com, З/п: 9991,84, Дети: Нет
9
10 ID: 90-6975762, Имя: Hollyanne, Фамилия: Windaybank, (Ж)
11 E-mail: hwindaybankkd@businesswire.com, З/п: 9977,54, Дети: Есть
12
13 ***** Задача 3 *****
14 С детьми: 499, Без детей 501
15
16 ***** Задача 4 *****
17 Фонд з/п: $5,522,720.000
18 Муж.+дети: $1,392,537.00, Жен.+дети: $1,358,142.00,
19 Муж.-дети: $1,372,988.00, Жен.-дети: $1,399,052.00
20

```

Рисунок 11.12 – Выходной файл приложения в режиме release

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

В таблице представлены задания, которые необходимо выполнить с использованием языка программирования. В каждом варианте необходимо использовать свой набор данных (файлы с данными необходимо разработать самостоятельно).

Таблица 11.1

## Индивидуальные задания

Вариант	Задание
1	1 Количество сотрудников в каждой из 3-х категорий; 2 Суммарный фонд з/п и сумму по каждой категории; 3 Количество сотрудников с дипломом; 4 Максимальную зарплату среди сотрудников без диплома.
2	1. Количество продуктов в каждой из 3-х категорий; 2. Суммарный объем поставок в валюте по каждой категории; 3. Суммарный объем в валюте всей выборки; 4. Количество просроченных продуктов.
3	1. Самый дорогой продукт для каждого из 4-х поставщиков; 2. Товар с минимальным количеством в каждой категории; 3. Объем просрочки в валюте; 4. Среднюю цену у каждого поставщика.
4	1. Количество сотрудников в каждой из 4-х компаний; 2. Среднюю з/п в каждой из 4-х компаний; 3. Количество сотрудников, владеющих английским языком; 4. Количество сотрудников, получающих более \$3000,00.
5	1. Количество билетов с пунктом отправления из Ставрополя; 2. Средняя цена билета для каждого из четырех пунктов отправления; 3. Количество билетов, купленных женщинами и мужчинами; 4. Количество билетов с одинаковыми пунктами отправления и прибытия.

## Продолжение таблицы 11.1

6	<p>Дан список целей, которые описываются параметрами.</p> <ol style="list-style-type: none"> <li>1. Определите среднее значение Param1 по каждой из категорий;</li> <li>2. Определите сумму параметров Param2 для каждой категории;</li> <li>3. Сколько целей снабжены оружием, а сколько – нет;</li> <li>4. Найти объект с максимальным значением Param1.</li> </ol>
7	<p>Дан список сотрудников.</p> <ol style="list-style-type: none"> <li>1. Количество сотрудников без логина;</li> <li>2. Средняя заработная плата по каждому из 4-х отделов;</li> <li>3. Максимальная з/п сотрудников у которых есть собака или кошка;</li> <li>4. Средняя з/п у сотрудников с оружием и без.</li> </ol>
8	<p>Дан список доменов. Требуется определить:</p> <ol style="list-style-type: none"> <li>1. Количество доменов для которых не указан domain или user;</li> <li>2. Средняя цена домена по каждой из 4-х категорий;</li> <li>3. Максимальная цена домена среди продающихся и не продающихся;</li> <li>4. Суммарная стоимость доменов без доменного имени.</li> </ol>
9	<p>В файле содержатся сведения о сотрудниках компании, их зарплатах и сумме налога с зарплаты.</p> <ol style="list-style-type: none"> <li>1. У какого сотрудника минимальная з/п? Какой сотрудник получает минимальную сумму после уплаты налогов?</li> <li>2. У скольких сотрудников не указан навык (skill)?</li> <li>3. Определите среднюю з/п для сотрудников с e-mail и без.</li> <li>4. Определите сумму налога по всей выборке</li> </ol>

## Продолжение таблицы 11.1

10	<p>В файле содержатся данные о счетах клиентов банка, которые отражают приход и расход по счету, а также сумму налога, которую следует оплатить.</p> <ol style="list-style-type: none"> <li>1. Сколько клиентов имеют отрицательный баланс счета, т.е. приход меньше расхода?</li> <li>2. Вычислите среднюю сумму налогов по счетам.</li> <li>3. Найдите клиента с максимальными расходами.</li> <li>4. Вычислите средний доход по счету для клиентов без e-mail.</li> </ol>
11	<p>В файле содержатся сведения о товарах на складе (ID, наименование, поставщик, цена, количество, категория, скидка в %).</p> <ol style="list-style-type: none"> <li>1. Определите количество товаров со скидкой более 25%.</li> <li>2. Вычислите суммарный объем (в валюте) товаров, для которых не указали поставщика.</li> <li>3. Для каждого из 4-х поставщиков определите среднюю цену товаров.</li> <li>4. Найдите минимальную цену товара с учетом скидки.</li> </ol>
12	<p>В файле содержатся сведения о производстве товаров работниками (ID, сотрудник, категория товара, з/п рабочего, количество произведенных товаров, цена за единицу товара).</p> <ol style="list-style-type: none"> <li>1. Определите сколько рабочих получают больше, чем вырабатывают продукции.</li> <li>2. Суммарный объем произведенной продукции (в валюте) без категории.</li> <li>3. Суммарный объем в валюте произведенной продукции по каждой из 4-х категорий.</li> <li>4. Самый эффективный сотрудник: с максимальной разницей между произведенным продуктом и зарплатой.</li> </ol>

## Продолжение таблицы 11.1

13	<p>В файле содержатся сведения о сотрудниках компании, их зарплатах и сумме налога с зарплаты.</p> <ol style="list-style-type: none"> <li>1. Максимальная и минимальная з/п. Вывести соответствующих сотрудников.</li> <li>2. Средняя з/п сотрудников без указанного навыка (skill)?</li> <li>3. Количество сотрудников без e-mail.</li> <li>4. Суммарный налог.</li> </ol>
14	<p>В файле содержатся данные о товарах (ID, наименование, категория (4 шт.), количество, цена, признак просрочки, признак наличия скидки).</p> <ol style="list-style-type: none"> <li>1. Средняя цена товаров для просроченных и непросроченных товаров.</li> <li>2. Суммарный объем в валюте товаров без категории.</li> <li>3. Самый дорогой товар в каждой категории.</li> <li>4. Количество наименований товаров без скидки</li> </ol>
15	<p>В файле содержатся данные о счетах клиентов банка, которые отражают доход (plus) и расход (minus) по счету, а также сумму налога, которую следует оплатить (tax).</p> <ol style="list-style-type: none"> <li>1. Суммарный расход по счетам клиентов, у которых не указан email.</li> <li>2. Клиент с самым большим балансом с учетом уплаты налогов.</li> <li>3. Средний доход по счетам с положительным балансом.</li> <li>4. Суммарная сумма налогов у клиентов с отрицательным балансом</li> </ol>
16	<ol style="list-style-type: none"> <li>1. Сумма фонда з/п в каждой из 3-х категорий сотрудников;</li> <li>2. Количество сотрудников с дипломом и без;</li> <li>3. Количество сотрудников с з/п менее \$50000,00;</li> <li>4. Максимальную зарплату среди сотрудников без диплома и минимальную у сотрудников с дипломом.</li> </ol>

## Продолжение таблицы 11.1

17	<ol style="list-style-type: none"> <li>1. Средняя цена в каждой из 3-х категорий;</li> <li>2. Товар с максимальной ценой;</li> <li>3. Суммарный объем в валюте непросроченных продуктов;</li> <li>4. Количество просроченных продуктов</li> </ol>
18	<p>В файле содержатся сведения о товарах на складе (ID, наименование, поставщик, цена, количество, категория, скидка в %).</p> <ol style="list-style-type: none"> <li>1. Определите среднюю цену с учетом скидки по каждому из 4-х поставщиков.</li> <li>2. Найдите самый дорогой и самый дешевый товары.</li> <li>3. По каждой из 4-х категорий найдите товар с минимальным остатком.</li> <li>4. Количество товаров без указания поставщика.</li> </ol>
19	<p>Дан список сотрудников.</p> <ol style="list-style-type: none"> <li>1. Сколько сотрудников не имеют логина или пароля;</li> <li>2. Средняя з/п сотрудников с семьей и без;</li> <li>3. Сколько сотрудников имеют машину, но не имеют семьи;</li> <li>4. Минимальная и максимальная з/п в каждом из 4-х отделов.</li> </ol>
20	<ol style="list-style-type: none"> <li>1. Суммарный объем в валюте для каждого из 4-х поставщиков;</li> <li>2. Количество наименований товаров дешевле \$15,00;</li> <li>3. Суммарный объем просрочки в валюте у каждого поставщика;</li> <li>4. Количество товаров с ценой более \$50,00.</li> </ol>

## Продолжение таблицы 11.1

21	<p>В файле содержатся данные о счетах клиентов банка, которые отражают приход и расход по счету, а также сумму налога, которую следует оплатить.</p> <ol style="list-style-type: none"> <li>1. Вычислите средний баланс по счетам с положительным балансом.</li> <li>2. Количество счетов с отрицательным балансом.</li> <li>3. Сколько клиентов имеют баланс, недостаточный для уплаты налога?</li> <li>4. Найдите клиента с максимальным балансом счета.</li> </ol>
22	<ol style="list-style-type: none"> <li>1. Максимальную з/п в каждой из 4-х компаний;</li> <li>2. Количество сотрудников в каждой из 4-х компаний;</li> <li>3. Количество сотрудников в каждой языковой группе;</li> <li>4. Количество сотрудников, получающих зарплату выше средней по всей выборке.</li> </ol>
23	<p>В файле содержатся сведения о производстве товаров работниками (ID, сотрудник, категория товара, з/п рабочего, количество произведенных товаров, цена за единицу товара).</p> <ol style="list-style-type: none"> <li>1. Определите сколько рабочих получают меньше, чем вырабатывают продукции.</li> <li>2. Количество единиц произведенной продукции без категории.</li> <li>3. Суммарный объем в валюте произведенной продукции по каждой из 4-х категорий.</li> <li>4. Самый неэффективный сотрудник: с минимальной разницей между произведенным продуктом и зарплатой.</li> </ol>

## Продолжение таблицы 11.1

24	<p>В файле содержатся данные о товарах (ID, наименование, категория (4 шт.), количество, цена, признак просрочки, признак наличия скидки).</p> <ol style="list-style-type: none"> <li>1. Максимальная цена товара для просроченных и непросроченных товаров.</li> <li>2. Количество товаров без категории.</li> <li>3. Самый дорогой товар с учетом скидки в каждой категории.</li> <li>4. Объем товаров в валюте с учетом скидки.</li> </ol>
25	<p>В файле содержатся сведения о сотрудниках компании, их зарплатах и сумме налога с зарплаты.</p> <ol style="list-style-type: none"> <li>1. Максимальная и минимальная з/п после уплаты налога. Вывести соответствующих сотрудников.</li> <li>2. Средняя з/п сотрудников без указанного навыка (skill).</li> <li>3. Количество сотрудников с e-mail.</li> <li>4. Объем фонда заработной платы.</li> </ol>
26	<p>В файле содержатся сведения о товарах на складе (ID, наименование, поставщик, цена, количество, категория, скидка в %).</p> <ol style="list-style-type: none"> <li>1. Определите количество наименований товаров в каждой из 4-х категорий.</li> <li>2. Вычислите суммарный объем (в валюте) товаров по каждому из 4-х поставщиков, а также по товарам, у которых не указан поставщик.</li> <li>3. Найдите самый дорогой товар с учетом скидки.</li> <li>4. Найдите товар с минимальным количеством.</li> </ol>



## Продолжение таблицы 11.1

27	<p>В файле содержатся сведения о производстве товаров работниками (ID, сотрудник, категория товара, з/п рабочего, количество произведенных товаров, цена за единицу товара).</p> <ol style="list-style-type: none"> <li>1. Определите сколько рабочих получают меньше, чем вырабатывают продукции.</li> <li>2. Количество единиц произведенной продукции без категории.</li> <li>3. Суммарный объем в валюте произведенной продукции по каждой из 4-х категорий.</li> <li>4. Количество сотрудников, получающих более 50% от суммы производимого продукта.</li> </ol>
28	<ol style="list-style-type: none"> <li>1. Максимальную з/п среди англоговорящих сотрудников;</li> <li>2. Средняя з/п в каждой из 4-х компаний;</li> <li>3. Количество сотрудников в каждой языковой группе;</li> <li>4. Сотрудник с максимальной з/п</li> </ol>
29	<p>В файле содержатся данные о счетах клиентов банка, которые отражают приход (plus) и расход (minus) по счету, а также сумму налога, которую следует оплатить (tax).</p> <ol style="list-style-type: none"> <li>1. Максимальный расход по всем клиентам.</li> <li>2. Количество счетов с положительным балансом.</li> <li>3. Суммарный положительный и отрицательный балансы.</li> <li>4. Средний баланс для клиентов с отрицательным балансом</li> </ol>

## Продолжение таблицы 11.1

30	<p>В файле содержатся данные о товарах (ID, наименование, категория (4 шт.), количество, цена, признак просрочки, признак наличия скидки).</p> <ol style="list-style-type: none"> <li>1. Максимальная цена товара для просроченных и непросроченных товаров.</li> <li>2. Суммарный объем в валюте товаров без категории с учетом скидки.</li> <li>3. Самый дорогой товар с учетом скидки в каждой категории.</li> <li>4. Количество наименований товаров без скидки.</li> </ol>
31	<p>В файле содержатся сведения о товарах на складе (ID, наименование, поставщик, цена, количество, категория, скидка в %).</p> <ol style="list-style-type: none"> <li>1. Определите количество наименований товаров в каждой из 4-х категорий.</li> <li>2. Вычислите суммарный объем (в валюте) по товарам, у которых не указан поставщик.</li> <li>3. Найдите самый дорогой товар в каждой категории.</li> <li>4. Найдите товар с максимальной скидкой</li> </ol>
32	<ol style="list-style-type: none"> <li>1. Средняя цена для каждого из 4-х поставщиков;</li> <li>2. Суммарный объем товаров от каждого поставщика;</li> <li>3. Количество просрочки от каждого поставщика;</li> <li>4. Количество товаров с ценой более \$50,00.</li> </ol>
33	<p>В файле содержатся данные о счетах клиентов банка, которые отражают приход (plus) и расход (minus) по счету, а также сумму налога, которую следует оплатить (tax).</p> <ol style="list-style-type: none"> <li>1. Максимальный доход по всем клиентам.</li> <li>2. Количество счетов с отрицательным балансом.</li> <li>3. Суммарный положительный и отрицательный балансы.</li> <li>4. Средний баланс для клиентов с отрицательным балансом.</li> </ol>

## Продолжение таблицы 11.1

34	<p>Дан список целей, которые описываются параметрами.</p> <ol style="list-style-type: none"> <li>1. Определите максимальное значение Param1 по каждой из категорий;</li> <li>2. Определите сумму параметров Param3 для каждой категории;</li> <li>3. Сколько целей снабжены оружием, а сколько – нет;</li> <li>4. Найти объект с максимальным значением Param2.</li> </ol>
35	<p>В файле содержатся сведения о товарах на складе (ID, наименование, поставщик, цена, количество, категория, скидка в %).</p> <ol style="list-style-type: none"> <li>1. Определите среднюю цену товаров со скидкой более 25%.</li> <li>2. Вычислите максимальную цену среди товаров, для которых не указа поставщик. Выведите этот товар.</li> <li>3. Для каждого из 4-х поставщиков определите товар с максимальной ценой.</li> <li>4. Найдите товар с максимальной ценой с учетом скидки и без учета скидки.</li> </ol>

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Дайте определение понятия «файл».
2. Опишите формат файла \*.csv.
3. Поясните назначение различных конфигураций (debug или release) при сборке приложения в среде Visual Studio.
4. Приведите пример кода, реализующего различное поведение программы при использовании различных режимов сборки (debug или release).
5. Что такое поток? Какие типы классов потоков используются при работе с файлами?
6. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов

## ЛАБОРАТОРНАЯ РАБОТА 12. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* научиться работать с циклами, условиями, ветвлениями.

Задачи лабораторной работы:

- освоить на практике работу с циклами, ветвлениями, условиями.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

#### Основы структурного программирования

Структурное программирование – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Методология предложена в 1970-х годах Э. Дейкстрой и др.

В соответствии с данной методологией любая программа строится без использования оператора `goto` из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются подпрограммы, речь о которых пойдет в следующей лабораторной работе. При этом разработка программы ведётся пошагово, методом «сверху вниз». В теории программирования уже давно было доказано, что для решения задач совершенно любой сложности, начиная от примитивных окон и заканчивая сложными информационными системами можно составить программу, которая будет состоять только из трех структур, о которых было упомянуто выше. Официально такое исследование в 1966 году было проведено Боймом Якопини.

Схематично, структуры следование, цикл и ветвление можно представить следующим образом (рис. 12.1).

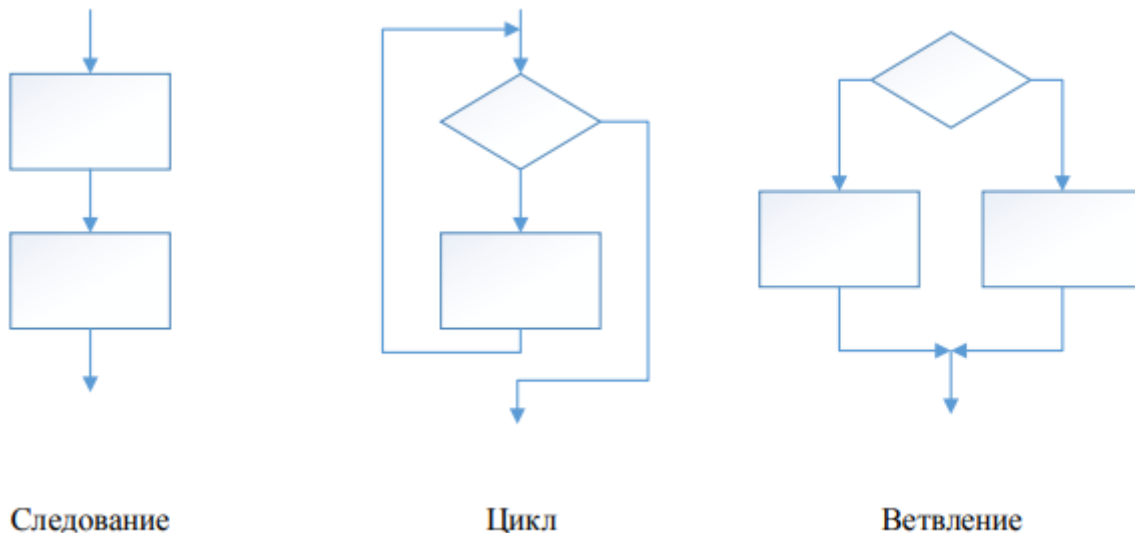


Рисунок 12.1 - Управляющие структуры

Ниже приведено краткое описание каждой структуры:

- **следованием** называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных);
- **цикл** задает многократное выполнение оператора;
- **ветвление** задает выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

Идеей использования базовых конструкций является получение программы простой структуры. Такую программу легко читать (а программы чаще приходится читать, чем писать), отлаживать и при необходимости вносить в нее изменения. Структурное программирование, как вы могли заметить, часто называли «программированием без goto», и в этом есть большая доля правды: частое использование операторов передачи управления в произвольные точки программы затрудняет прослеживание логики ее работы.

В большинстве языков высокого уровня существует несколько реализаций базовых конструкций; в C# есть четыре вида циклов и два вида ветвлений (на два и на произвольное количество направлений). Они введены для удобства программирования, и в каждом случае надо выбирать наиболее подходящие средства. Главное, о чем нужно помнить даже при написании са-

мых простых программ, — что они должны состоять из четкой последовательности блоков строго определенной конфигурации.

### Циклы

Как было указано выше, циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз.

В C# имеются следующие виды циклов:

1. For
2. Foreach
3. While
4. do...while

Давайте разберем каждый вид отдельно и рассмотрим несколько примеров.

#### Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика])
{
    // действия
}
```

Рассмотрим небольшой пример использования цикла for. Стандартный пример, прохождение массива из 10 элементов.

```
for (int i = 0; i < 9; i++)
{
    Console.WriteLine("Квадрат числа {0} равен {1}", i, i * i);
}
```

Первая часть объявления цикла — `int i = 0` — создает и инициализирует счетчик `i` (буква `i` выбрана в связи с устоявшейся традицией, хотя использоваться может и любой другой символ). Счетчик необязательно должен представлять тип `int`. Это может быть и другой числовой тип, например, `float`. И перед выполнением цикла его значение будет равно 0. В данном случае это тоже самое, что и объявление переменной.

Вторая часть — условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока `i` не достигнет 9. И третья часть — приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: `i--`.

В итоге блок цикла сработает 9 раз, пока значение `i` не станет равным 9. И каждый раз это значение будет увеличиваться на

1. Нам необязательно указывать все условия при объявлении цикла. Например, цикл может быть реализован следующим образом:

```
int i = 0;
for (; ; )
{
    Console.WriteLine("Квадрат числа {0} равен {1}", ++i, i * i);
    System.Threading.Thread.Sleep(500);
}
```

Формально определение цикла осталось тем же, только теперь блоки в определении у нас пустые: `for (; ;)`. У нас нет инициализированной переменной-счетчика, нет условия, поэтому цикл будет работать вечно – бесконечный цикл.

Мы также можем опустить ряд блоков, как показано в примере ниже.

```
int i = 0;
for (; i < 9; )
{
    Console.WriteLine("Квадрат числа {0} равен {1}", ++i, i * i);
}
```

Этот пример по сути эквивалентен первому примеру: у нас также есть счетчик, только создан он вне цикла. У нас есть условие выполнения цикла. И есть приращение счетчика уже в самом блоке `for`.

### Цикл `foreach`

Исторически сложилось, что одним из наиболее трудных для понимания циклов является цикл – `foreach`. На самом деле в нем ничего сложно и сейчас мы в этом убедимся.

Цикл `foreach` предназначен для перебора элементов в контейнерах. Формальное объявление цикла `foreach`:

```
foreach (тип_данных название_переменной in контейнер)
{
    // действия
}
```

Для того чтобы ситуация стала более понятной, приведем небольшой пример. Допустим, у нас есть массив из 5 элементов и нам надо обратиться к каждому из элементов массива, чтобы вывести его значение на экран. Код приведен ниже.

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in array)
{
    Console.WriteLine(i);
}
```

Обратите внимание, что в примере в качестве контейнера выступает массив данных типа `int`. Поэтому мы объявляем переменную с типом `int`.

Внимательный читатель может заметить, что абсолютно тоже можно сделать и при помощи цикла `for`. Вы можете проделать этот пример самостоятельно. Несмотря на лаконичную конструкция цикла `foreach`, цикл `for` более гибкий по сравнению с `foreach`. Если `foreach` последовательно извлекает элементы контейнера и только для чтения, то в цикле `for` мы можем перескакивать на несколько элементов вперед в зависимости от приращения счетчика, а также можем изменять элементы.

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < array.Length; i++)
{
    array[i] = array[i] * 2;
    Console.WriteLine(array[i]);
}
```

Разберите самостоятельно, что делает пример приведенный выше

### Цикл `do`

В цикле `do` сначала выполняется код цикла, а потом происходит проверка условия в инструкции `while`. И пока это условие истинно, цикл повторяется. Например, следующий код уменьшает значение переменной `i` на единицу, до тех пор, пока переменная больше нуля.

```
int i = 6;
do
{
    Console.WriteLine(i);
    i--;
}
while (i > 0);
```

Наверно, вы уже догадались, что код цикла сработает 6 раз, пока `i` не станет равным нулю. Но важно отметить, что цикл `do` гарантирует хотя бы однократное выполнение действий, даже если условие в инструкции `while` не будет истинно. Фактически, задавая начальное значение для `i = -1` цикл гарантированно выполнится один раз. Попробуйте проверить это самостоятельно.

### Цикл `while`



В отличие от цикла `do` цикл `while` сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

```
int i = 6;
while (i > 0)
{
    Console.WriteLine(i);
    i--;
}
```

Попробуйте задать начальное условие для переменной `i` равное `-1` и запустить цикл. Сравните полученный результат с циклом `do...while`.

### Операторы `continue` и `break`

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором `break`. Пример приведен ниже.

```
int[] array = new int[] { 1, 2, 3, 4, 12, 9 };
for (int i = 0; i < array.Length; i++)
{
    if (array[i] > 10)
        break;
    Console.WriteLine(array[i]);
}
```

Поскольку в цикле идет проверка, больше ли элемент массива 10. То мы никогда не увидим на консоли последние два элемента, так как, увидев, что элемент массива больше 10, сработает оператор `break`, и цикл завершится.

Теперь поставим себе другую задачу. А что, если мы хотим, чтобы при проверке цикл не завершался, а просто переходил к следующему элементу. Для этого мы можем воспользоваться оператором `continue`:

```
int[] array = new int[] { 1, 2, 3, 4, 12, 9 };
for (int i = 0; i < array.Length; i++)
{
    if (array[i] > 10)
        continue;
    Console.WriteLine(array[i]);
}
```

В этом случае цикл, когда дойдет до числа 12, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующему элементу массива.

## Ветвления / Условные конструкции

Условные конструкции — один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий. В языке C# используются следующие условные конструкции: `if..else` и `switch..case`.

### Конструкция `if/else`

Конструкция `if/else` проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код. Давайте рассмотрим небольшой пример использования оператора.

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
```

После ключевого слова `if` ставится условие. И если это условие выполняется, то срабатывает код, который помещен далее в блоке `if` после фигурных скобок. В качестве условий выступают ранее рассмотренные операции сравнения.

В данном случае у нас первое число больше второго, поэтому выражение `num1 > num2` истинно и возвращает `true`, следовательно, управление переходит к строке `Console.WriteLine("Число {0} больше числа {1}", num1, num2);`

Но что, если мы захотим, чтобы при несоблюдении условия также выполнялись какие-либо действия? В этом случае мы можем добавить блок `else`:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
else
{
    Console.WriteLine("Число {0} меньше числа {1}", num1, num2);
}
```

Но при сравнении чисел мы можем насчитать три состояния: первое число больше второго, первое число меньше второго

и числа равны. Используя конструкцию `else if`, мы можем обрабатывать дополнительные условия:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
else if (num1 < num2)
{
    Console.WriteLine("Число {0} меньше числа {1}", num1, num2);
}
else
{
    Console.WriteLine("Число num1 равно числу num2");
}
```

Также мы можем соединить сразу несколько условий, используя логические операторы:

```
int num1 = 8;
int num2 = 6;
if(num1 > num2 && num1 > 8)
{
    Console.WriteLine("Число {0} больше числа {1}", num1, num2);
}
```

Обратите внимание на конструкцию `&&`, которая выполняет логическое умножение, другими словами операцию «И». Это говорит о том, что блок будет истинен только при выполнении обоих условий. Не трудно догадаться, что существует логическое сложение, оператор «ИЛИ» – в языке C# обозначается как `&|` две прямых черты.

### **Важно!**

*Распространенной ошибкой является использование оператора присваивания «=» внутри блока `if`. Важно понимать отличие между оператором присваивания и оператором проверки на равенство «==». Внутри блока `if` может использоваться только оператор проверки на равенство, так как он может вернуть значение истина или ложь, в зависимости от того, равны переменные или нет. Таким образом правильно будет `if (a == b)`. Будьте внимательны.*

### **Конструкция switch**

Конструкция `switch/case` аналогична конструкции `if/else`, так как позволяет обработать сразу несколько условий. Рассмотрим пример.

```

Console.WriteLine("Нажмите Y или N");
string selection = Console.ReadLine();
switch (selection)
{
    case "Y":
        Console.WriteLine("Вы нажали букву Y");
        break;
    case "N":
        Console.WriteLine("Вы нажали букву N");
        break;
    default:
        Console.WriteLine("Вы нажали неизвестную букву");
        break;
}

```

После ключевого слова `switch` в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора `case`. И если совпадение будет найдено, то будет выполняться определенный блок `case`. В конце блока `case` ставится оператор `break`, чтобы избежать выполнения других блоков. Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок `default`, как в примере выше.

### Тернарная операция

Тернарная операция имеет следующий синтаксис: [первый операнд – условие] ? [второй операнд] : [третий операнд]. Здесь сразу три операнда. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно `true`, то возвращается второй операнд; если условие равно `false`, то третий. В некотором роде, тернарная операция представляет собой сокращенный синтаксис операции `if/else`, но все-таки имеет ряд отличий. Пример ниже:

```

int x = 3;
int y = 2;
Console.WriteLine("Нажмите + или -");
string selection = Console.ReadLine();

int z = selection == "+" ? (x + y) : (x - y);
Console.WriteLine(z);

```

Здесь результатом тернарной операции является переменная `z`. Если мы выше вводим `"+"`, то `z` будет равно второму операнду – `(x+y)`. Иначе `z` будет равно третьему операнду.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

### Реализация приложения «Записная книжка»

В качестве примера рассмотрим создание простой записной книжки. Читатель может либо проделать всю последовательность шагов и выполнить тестовый проект, либо сразу выполнять задание в соответствии со своим вариантом.

Хотелось бы обратить внимание, что данный пример не является оптимальным решением задачи. И в ходе прохождения лабораторного практикума, мы будем модернизировать его, используя новые технологии и подходы. А пока не будем забегать вперед и создадим новое консольное приложение под названием Notebook.

*Примечание. В данной лабораторной работе не будет описано создание проекта и работа со средой Visual Studio. Предполагается, что на момент выполнения лабораторной работы студент уже обладает обозначенными знаниями.*

В процессе выполнения лабораторной работы мы будем добавлять весь код в функцию Main без использования каких-либо дополнительных конструкций.

### Создание хранилища записей

В нашем случае в качестве будет использоваться обыкновенная переменная – строковый массив. Это говорит о том, что все данные, записанные в записную книжку, будут удалены после каждого закрытия программы. Согласен с вами, не самая удобная записная книжка. Для хранения строковых данных использовать несколько видов контейнеров. Одним из самых простых решений – массив строк. Мы пропустим данный вариант, так как он не является ключевой частью лабораторной работы. Для удобства реализации в качестве хранилища записей будущей записной книжки используем так называемый список List. Детально ознакомиться со всеми особенностями списка List можно, вызвав справку. Пока же просто будем использовать список List, не вдаваясь в подробности данной структуры. И так, создаем список записей, как показано в примере ниже.

```
List<String> notes = new List<string>();
```

В данном случае указывает на то, что наш список хранит строки. Одна строка – одна запись в списке. Конструкция new

List() нужна для инициализации нового списка. И так, наш новый список называется notes, это объект типа List. Списки очень удобны в плане добавления и удаления записей. Так, для добавления новой записи в наш список notes воспользуемся командой Add. Добавим несколько первоначальных значений, как показано в примере ниже.

Добавим пару начальных записей.

```
notes.Add("Первая запись");
notes.Add("Вторая запись");
```

И так, мы создали наше, хоть и временное, но хранилище данных. Теперь после каждого включения программы в нем будет находиться две записи. Самое время внедрить основные элементы структурного программирования, циклы и ветвления.

### **Внедрение циклов и ветвлений**

Вся логика нашего приложения будет заключаться в том, что пользователю предоставляется список команд:

- добавить запись;
- удалить запись по номеру n;
- посмотреть все записи;
- посмотреть список доступных команд;
- выйти из приложения.

Выбор каждой команды будет осуществляться по нажатию соответствующей клавиши. Как вы, возможно, уже догадались, для реализации подобной логики нам потребуются операторы switch и главный цикл while. И так, обо всем по порядку.

Прежде чем реализовывать циклы, добавим переменную, в которой будет храниться выбранное действие пользователем (нажатая клавиша).

Удобнее всего использоваться тип char, в котором будет храниться один единственный символ. Добавляем строку:

```
char action = 'h';
```

Мы назвали переменную action (действие) и присвоили значение 'h'. В будущем этот символ будет означать просмотр списка всех команд, это необходимо для того, чтобы при запуске программы первым делом отобразился список доступных команд.

Далее создадим наш главный цикл, в котором будет производиться ввод и выполнение команд.



```
while (action != 'q')
{
    // основной цикл
}
```

Изучив предыдущий материал, слушатель без труда определит, что подобный цикл `while` выполняется, пока в переменную `action` не записан символ `q`, другими словами, пока пользователь не нажмет клавишу для выхода из приложения.

Теперь необходимо осуществить выполнение действий, соответствующих введенной команде. Для этого будем использовать оператор `switch`. Как вы помните, оператор `switch` включает один или несколько разделов переключения. Каждый раздел переключения содержит одну или несколько меток `case`, за которыми следует один или несколько операторов. Добавляем каркас нашего будущего ветвления. Пока обработаем нажатие только одной клавиши. В примере ниже после нажатия на ничего не произойдет, наполнение оператора будет добавлено позже.

```
switch (action)
{
    case '1':
        // если action равен '1' то выполнить описанные
        // здесь операции
        break;
}
```

Раз мы добавили первую команду, – нажатие на клавишу, давайте реализуем логику этого запроса. По нажатия на мы будем выводить для пользователя все записи. Для отображения записей выполним проход по списку, и вывод каждой записи в отдельную строку.

```
case '1':
    foreach (var note in notes)
    {
        Console.WriteLine((notes.IndexOf(note)+1) + " " +note);
        // т.к. в списке нумерация начинается с 0, то прибавляем 1
    }
    break;
```

Вспомним материал из предыдущего раздела. `Foreach` – цикл, который проходит по всем записям в переменной `notes`, при этом в теле цикла обращение к текущей записи происходит через переменную `note`. Фактически на каждом шаге цикла переменная

note смещается вправо, получает следующее значение, выводит его на экран и идет дальше (рис. 12.2).

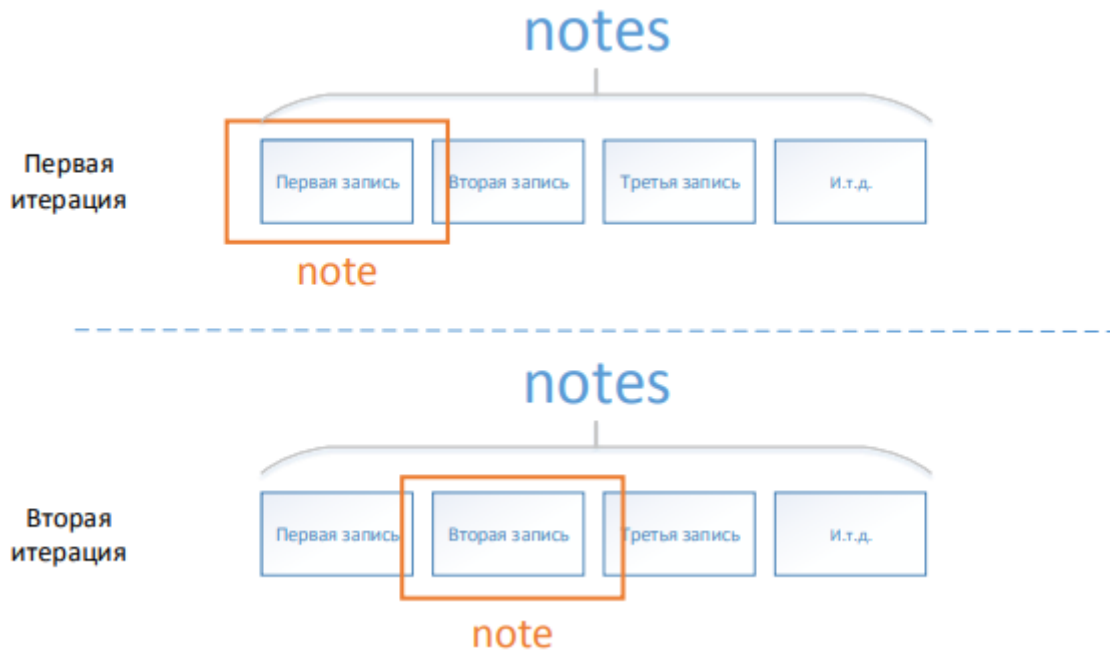


Рисунок 12.2 - Обход списка операцией `foreach`

Для вывода записи используется команда `Console.WriteLine`. Для получения номера записи используется команда `notes.IndexOf`. Фактически функция `IndexOf` объекта `notes` возвращает номер элемента списка, который был в нее передан. Стоит заметить, что в программе нумерация записей начинается с 0, поэтому при выводе мы прибавляем к текущему номеру записи 1 с помощью вот такой конструкции `IndexOf(note)+1`).

Данная конструкция позволит выводить записи в формате: «1) Название элемент». Далее определим операцию вывода доступных команд, мы договоримся, что список доступных команд будет выводиться при старте программы и при нажатии пользователем клавиши .

```
case 'h':
    Console.WriteLine("Доступные команды:");
    Console.WriteLine("a - добавить запись");
    Console.WriteLine("d - удалить запись с номером n");
    Console.WriteLine("l - список всех записей");
    Console.WriteLine("h - список доступных команд");
    Console.WriteLine("q - выйти из программы");
    break;
```

Реализуем следующую операцию – добавление новой записи. Пусть она выполняется при нажатии пользователем клавиши.



```
case 'a':
    Console.Write("Введите сообщение: ");
    var newNote = Console.ReadLine(); // считываем сообщение
    notes.Add(newNote); // добавляем сообщение в конец списка
    break;
```

С помощью команды `Console.ReadLine` мы считываем сообщение, вводимое пользователем. А с командой `Add` мы уже знакомы, она добавила введенную пользователем строку в конец списка.

Теперь определим операцию удаления записи. Удаление будет происходить, если пользователь нажмет клавишу.

```
case 'd':
    Console.Write("Введите номер записи для удаления: ");
    int n = Convert.ToInt32(Console.ReadLine())-1;
    if (n < notes.Count && n > -1)
    {
        notes.RemoveAt(n);
    } else
    {
        Console.WriteLine("Записи с указанным номером не существует");
    }
    break;
```

В данном блоке, пользователю предлагается ввести номер записи, которую он хочет удалить. Стоит обратить внимание на то, что команда `Console.ReadLine` возвращает в результате строку типа `string`, а нам нужен номер записи типа `int` (число), поэтому мы преобразуем строку типа `string` в номер типа `int` с помощью команды `Convert.ToInt32`.

Так же мы вычитаем 1 из введенного номера, поскольку нумерация записей в программе начинается с 0. Затем мы проверяем, что введенный номер принадлежит отрезку `[0; notes.Count]`, где `notes.Count` – команда для получения количества записей в списке.

Если номер введен верно, то удаляем запись из списка с помощью команды `notes.RemoveAt`, иначе выводим сообщение об ошибке. Если же ни одна из операций не была выполнена, значит, пользователь ввел неизвестную команду. Чтобы оповестить его об этом, используем секцию `default` оператора `switch`.

```
default:
    Console.WriteLine("Неизвестная команда");
    break;
```

На этом мы заканчиваем работу с оператором switch. Все возможные действия пользователя были обработаны.

Последнее, что остается сделать, это обеспечить ввод новой команды. Наверное, вы обратили внимание, что до сих пор мы не реализовали участок, который бы предлагал пользователю ввести команду. Добавим его сразу после блока switch. Обратите внимание, что блок ввода новой команды, тем не менее, должен находиться внутри цикла while, иначе программа завершит свою работу после выполнения первого действия.

```
Console.Write("Введите команду: ");
action = Console.ReadKey().KeyChar; // считываем новое действие
Console.WriteLine(); // для переноса вывода на новую строку
```

Команда Console.ReadKey в отличие от Console.ReadLine считывает нажатие одной клавиши, а метод .KeyChar позволяет получить символ нажатой клавиши, который мы и сравниваем в операторе switch, описанном выше.

Используя псевдокод, структура приложения должна быть следующая:

```
функция Main
{
    Реализация хранилища

    Общий цикл While
    {
        Оператор switch
        {

        }

        Блок ввода новой команды
    }
}
```

### Запуск программы

На этом реализация одной из самых простых версий справочника завершена. К сожалению, общий листинг программы приведен не будет. Результат выполнения программы изображен на рис. 12.3. При выполнении тестового задания студенты могут реализовать другие команды и использовать свои обозначения для каждой команды.

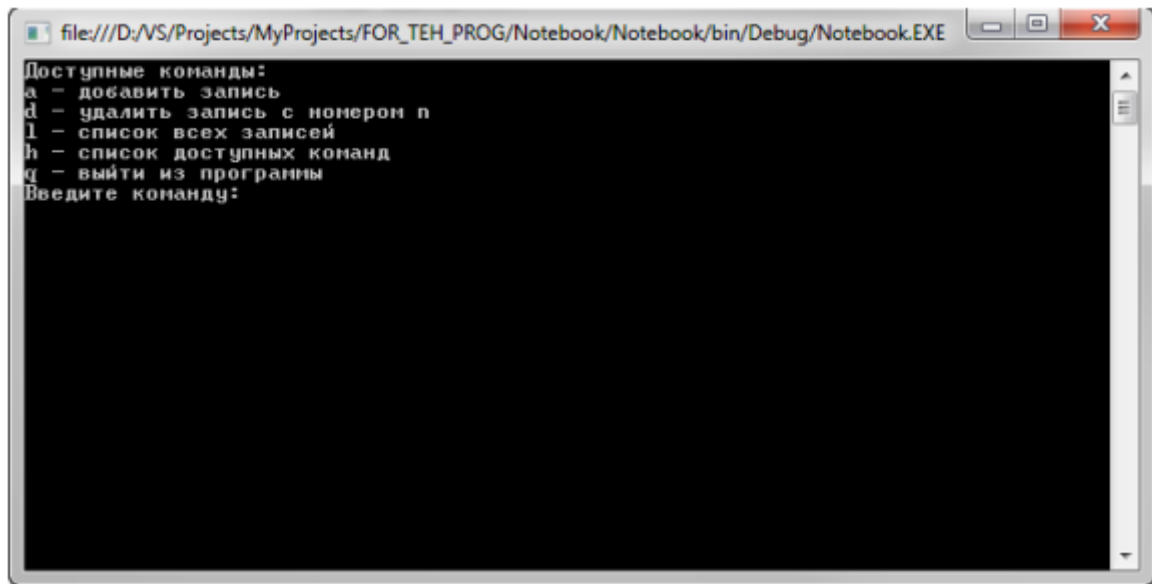


Рисунок 12.3 – Результат работы программы

### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

1. Реализовать консольное приложение в соответствии с вариантом.
2. Реализовать блок-схему основных элементов структурного программирования на основе примитивов, приведенных в методических указаниях.
3. Для ввода команд пользователем использовать цифры.
4. Добавить одну дополнительную команду, которая не была описана в методических указаниях.
5. Оформить отчет с описанием хода выполнения вашей лабораторной работы.

Таблица 12.1

#### Индивидуальные задания

Вариант	Задание
1,13	Телефонный справочник (в одной строке вводить: «ФИО, Телефон»).
2,14	Записная книжка (в одной строке вводить: «Запись, Дата»).
3,15	Список записей к врачу (в одной строке вводить: «Название врача, Дата»)
4,16	Список клиентов автомойки (в одной строке вводить: «Марка машины, Время»).

Продолжение таблицы 12.1

5,17	Список заказов в кафе (в одной строке вводить: «Номер стола, Блюдо»).
6,18	Путеводитель (в одной строке вводить: «Название места, Описание»).
7,19	Гостевая книга (в одной строке вводить: «Сообщение, Время»).
8,20	Почтовые данные (в одной строке вводить: «Адрес, Индекс»)
9,21	Список студентов (в одной строке вводить: «ФИО, Номер группы»).
10,22	Список рейсов (в одной строке вводить: «Название рейса, Дата прибытия»).
11,23	Список билетов на поезд (в одной строке вводить: «Номер вагона, Номер места»).
12,24	База автомобилей (в одной строке вводить: «Номер машины, Марка машины»).

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие циклы в С# существуют?
2. Что такое следование?
3. Что такое цикл?
4. Что такое ветвление?
5. Формальное определение цикла FOR.
6. Формальное определение цикла WHILE.
7. Формальное определение DO WHILE.
8. Формальное определение IF.
9. Формальное определение SWITCH CASE.

## ЛАБОРАТОРНАЯ РАБОТА 13. ОСНОВЫ РАБОТЫ С XML-ДОКУМЕНТАМИ НА ПЛАТФОРМЕ .NET FRAMEWORK

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* приобрести умение создавать документы XML, а также выполнять их запись и чтение в приложениях на языке C#.

Задачи лабораторной работы

- освоить основные особенности языка XML;
- научиться создавать XML-документов с помощью XML-редактора MS Visual Studio;
- приобрести умение программно создавать документы XML с помощью модели DOM;
- научиться выполнять сериализацию объектов в документы XML, а также десериализацию объектов из XML-документов.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

**Общие сведения о языке XML. Структура документа XML. Синтаксис XML**

В настоящее время наиболее распространенной открытой технологией описания данных является язык XML.

XML (англ. eXtensible Markup Language – расширяемый язык разметки) – язык разметки, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком.

Основные отличия между XML и HTML:

- XML и HTML создавались с различными целями: XML – для описания данных, HTML – для представления данных (как данные выглядят);
- Элементы XML не заданы в самом языке с самого начала. Требуется определять свои собственные элементы. В HTML-документах можно использовать только те элементы, которые определены в стандарте HTML;
- В XML имеет значение регистр написания символов (прописные/строчные). Использование неверного регистра при написании имени элемента является ошибкой.

В настоящее время XML превратился в стандарт обмена данными между программными приложениями. Кроме того, XML применяется в качестве средства для хранения данных, а также в различного конфигурационных файлах.

Спецификация языка XML утверждается Консорциумом Всемирной паутины (W3C) и описывает текстовые документы, называемые XML-документами (XML document).

XML-документ является простым текстовым файлом с расширением .xml. Для XML-документов в качестве кодировок символов обычно используют кодировки Юникод (например, UTF-8 или UTF-16).

Обработка документов XML требует программы, называемой парсером XML (XML parser) или синтаксическим анализатором XML.

Парсер XML – это программа, которая отслеживает выполнение синтаксических правил, определенных текущей версией стандарта XML. При обнаружении несоответствия стандарту отображается сообщение об ошибке, после чего обработка документа прекращается.

В число популярных парсеров XML входят Microsoft Internet Explorer XML Parser, Firefox XML Parser, Apache Xerces, XML4J от IBM. Часто XML-анализаторы встраиваются в среды разработки приложений (например, в MS Visual Studio).

XML образует целое семейство технологий, основными из которых являются:

- SAX – простой программный интерфейс для XML;
- DOM – объектная модель документа, которая обеспечивает программный доступ к отдельным частям XML-документа;
- DTD – определение типа документа;
- XSD – язык описания структуры XML-документа;
- XPath – язык путей XML;
- XQuery – язык запросов XML;
- XSLT – язык для преобразования XML-документов;
- SOAP и REST – протоколы, предназначенные для передачи структурированной информации в формате XML;

- RSS – семейство XML-форматов, предназначенных для описания лент новостей, анонсов статей и т. п.

Используя XML, можно создать другие специализированные языки разметки для представления информации любого вида. Например, MathML (Mathematical Markup Language), CML (Chemical Markup Language), KML (Keyhole Markup Language) и др.

### **Пролог XML и корневой элемент. Комментарии XML.**

XML-документ состоит из пролога и корневого элемента.

Пролог XML располагается в самом начале XML-документа и содержит информацию, относящуюся к документу в целом – о кодировке символов, структуре документа, таблицах стилей и др. Пролог не обязательно должен присутствовать в XML-документе, но рекомендуется включать в него объявление XML.

Объявление XML (англ. XML declaration) идентифицирует текст как XML-данные и указывает используемую версию XML. Объявление может иметь следующий вид:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
```

В XML-объявлении допускается указание следующих атрибутов:

- version – указывает версию XML (в настоящее время существует только одна версия XML – 1.0);
- encoding – определяет кодировку символов документа (по умолчанию применяется кодировка UTF-8);
- standalone – принимает значение «yes» (значение по умолчанию), если этот документ не ссылается на внешние объекты (например, на внешнее определение типа документа); в противном случае присваивается значение «no».

Ошибкой является помещение любых символов, включая символ пробела, перед объявлением XML.

Документ XML имеет один элемент, содержащий все прочие элементы и называемый корневым элементом. Корневой элемент должен стоять первым после пролога XML.

Попытка создания более одного корневого элемента в XML-документе является ошибкой.

Комментарии XML используются в целях документирования и начинаются с «<!--» и заканчиваются «-->»:

<!-- Это комментарий -->

Внутри комментария нельзя использовать комбинацию символов «--», поскольку двойной дефис играет роль признака окончания комментария.

### **Элементы, теги и атрибуты XML.**

Символы, составляющие XML-документ, делятся на разметку (markup) и символьные данные (character data).

Блоки разметки в XML-документе называют элементами XML. Элементы образуют разделы информации, которые можно обрабатывать программно или с помощью таблиц стилей.

Имена элементов XML должны подчиняться следующим правилам:

- имена могут быть любой длины и содержать буквы, цифры, символы подчеркивания, дефисы и точки; нельзя ставить внутри имени элемента пробелы;
- имена элементов должны начинаться с буквы или символа подчеркивания «\_»; не допускается начинать имя элемента с цифры или специального символа (!, ?, @, #, \$ и др.);
- имена не могут начинаться с xml (или XML или Xml ...).

Имена элементов следует делать короткими, но осмысленными.

Элементы задаются с помощью тегов, которые представляют собой имена, заключенные в угловые скобки (<>).

Тег, открывающий разметку, называется начальным (открывающим) тегом, а тег, закрывающий разметку, – конечным (закрывающим) тегом. Конечные теги отличаются от начальных тем, что содержат символ левой косой черты «/» сразу за символом «<».

Поскольку в XML имеет значение регистр написания символов то, начальные и конечные теги должны быть записаны в одном регистре.

Содержимым элемента является все, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы.

Элементы могут также содержать имена и значения атрибутов в начальных тегах



Атрибуты элементов XML предоставляют дополнительную информацию об элементе. При задании имен атрибутов следует придерживаться тех же правил, что и для имен элементов.

Элементы могут иметь любое число атрибутов. Ошибкой является предоставление элементу двух и более атрибутов с одинаковыми именами. В XML каждому атрибуту должно быть присвоено определённое значение. Присваивание атрибуту значения осуществляется с помощью знака равенства.

В XML значения атрибутов должны быть заключены в двойные или одинарные кавычки.

Элементы, содержащие начальный и конечный тег, а также содержимое называются элементами-контейнерами:

```
<message date="15.03.2015">Привет!</message>
```

Отсутствие конечного тега у элемента-контейнера при наличии начального является ошибкой.

Пустые элементы включают лишь один тег и не имеют содержания. Тег пустого элемента должен завершаться символом «>»:

```
<message date="21.12.2015" text="Привет!" />
```

Элементы могут вкладываться один в другой. При использовании вложенных элементов следует обращать внимание на последовательность открывающих и закрывающих тегов.

Порядок расположения закрывающих тегов должен быть обратным порядку расположения открывающих тегов:

```
<!-- Неправильно -->
```

```
<address><city>Новокузнецк</address></city>
```

```
<!-- Правильно -->
```

```
<address><city>Новокузнецк</city></address>
```

**Отношения между элементами в XML-документе. Уровни правильности XML-документа.**

Вложение одних элементов в другие образует иерархическую структуру XML-документа, которая может быть описана в терминах родственных отношений (родитель, потомок, предок, дочерний элемент, сестринский элемент) или деревьев (корень, ветвь, лист) (рис. 13.1).

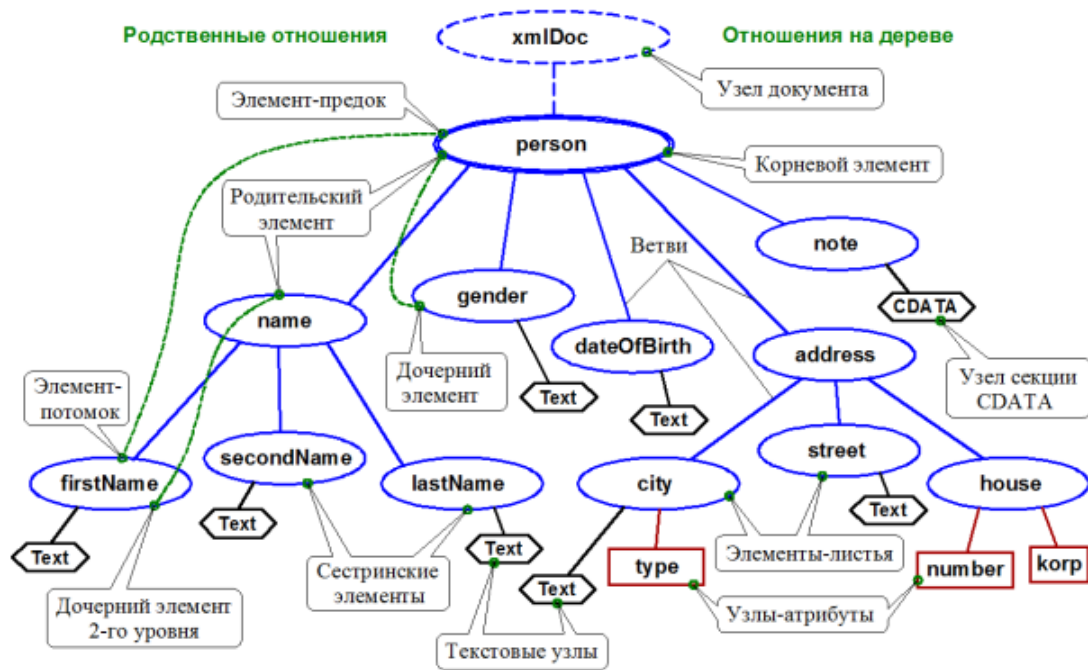


Рисунок 13.1. Отношения между узлами в иерархической структуре XML-документа

В терминах родственных отношений элемент *a* является родительским элементом для элемента *b*, если элемент *b* непосредственно вложен в *a*. При этом элемент *b* является дочерним элементом для элемента *a*. Если элементы *b* и *c* непосредственно вложены в элемент *a*, то элементы *b* и *c* называются сестринскими элементами.

Элемент *a* называют элементом-предком для элемента *b*, если элемент *b* вложен в элемент *a*, но вложение не является непосредственным. При этом элемент *b* называют элементом-потомком элемента *a*.

При использовании метафор дерева листьями называются элементы, не содержащие никаких других элементов, как листья на конце ветви. Элементы-листья обычно содержат только текстовый узел или вообще ничего.

Нет правил, указывающих, когда для размещения данных использовать дочерние элементы, а когда – атрибуты. Для улучшения читаемости XML-документа человеком следует размещать основные данные в дочерних элементах, а различные вспомогательные данные – в атрибутах.

Стандартом определены два уровня правильности XML-документа:

- **правильно построенный** (well-formed) XML-документ – документ, который соответствует всем общим правилам синтаксиса XML (правильное именованное элементов и атрибутов, правильное вложение элементов, только один корневой элемент и др.);
- **действительный** (valid) XML-документ – правильно построенный документ, который соответствует некоторым дополнительным правилам. Обычно такие правила хранятся в специальных файлах – схемах XML, где самым подробным образом описана структура документа, все допустимые названия элементов, атрибутов и многое другое.

Пример правильно оформленного XML-документа, содержащего сведения о сотруднике, приведён в листинге 13.1.

Листинг 1.1. Код XML-документа «Сведения о сотруднике»

---

```
<?xml version="1.0" encoding="utf-8" ?><!-- объявление -->
<employee><!-- корневой элемент -->
  <name>
    <firstName>Иван</firstName>
    <secondName>Викторович</secondName>
    <lastName>Кузнецов</lastName>
  </name>
  <gender>м</gender>
  <dateOfBirth>1985-03-20</dateOfBirth>
  <dateOfJoining>1999-01-04</dateOfJoining>
  <address>
    <city type="город">Кемерово</city>
    <street>пр. Ленина</street>
    <house number="70" /><!-- пустой элемент -->
    <apartment>22</apartment>
  </address>
  <phoneList>
    <workPhone>22-36-54</workPhone>
    <homePhone>11-35-04</homePhone>
    <mobilePhone id="1">8-942-345-7891</mobilePhone>
    <mobilePhone id="2">8-914-679-5038</mobilePhone>
  </phoneList>
  <designation>Системный архитектор</designation>
  <basic>30000</basic>
  <notes>
    <!-- секция CDATA -->
    <![CDATA[Закончил КузГТУ в 2006 г. по специальности
"Информационные системы и технологии".]]>
  </notes>
</employee>
```

---

**Ссылки на сущности, секции CDATA, инструкции по обработке.**

Ссылки на сущности (entity references) являются указанием XML-процессору подставить вместо них заранее определенную строку символов – сущность.

Вместо любого символа можно ввести его код в десятичной форме, предварив запись символами «&#» и закончив точкой с запятой «;». Шестнадцатеричная запись кода символа предваряется «&#x» и завершается «;».

Объектные ссылки:

- &amp; – символ «&» (амперсанд);
- &lt; – символ «<» (меньше);
- &gt; – символ «>» (больше);
- &apos; – символ «'» (апостроф);
- &quot; – символ «"» (двойная кавычка).

В XML-документах могут присутствовать большие текстовые фрагменты, включающие множество символов <, & и ", которые не следует интерпретировать в качестве элементов разметки. В этом случае можно воспользоваться секцией CDATA.

В секциях CDATA (Character Data Section) находятся символьные данные, которые не должны анализироваться XML-парсером.

Начало секции CDATA обозначается разметкой «<![CDATA[», а завершение – разметкой «]]>».

Внутри секции CDATA не должна присутствовать последовательность символов «]]>». Также не допускается вложение секций CDATA друг в друга.

В документах XML могут присутствовать инструкции по обработке (processing instructions), которые содержат указания программе-обработчику документа. Инструкции по обработке заключаются между символами «<?» и «?>».

Примером инструкции по обработке может служить первая строка пролога документа XML – объявление XML.

Инструкции по обработке могут располагаться в любом месте документа, но целиком в пределах одного элемента.

## **Редакторы XML. Особенности редактора XML в MS Visual Studio.**

Хотя для написания XML-документа может быть использован обычный текстовый редактор (например, Notepad), гораздо удобнее для этой цели использовать специализированный XML-редактор (XML editor).

Редакторы XML предлагают такие дополнительные возможности, как автоматическое завершение тегов, представление структуры XML-документа в виде дерева с узлами, проверка синтаксиса на соответствие общим правилам XML, а также правилам, задаваемым XML-схемой. В число наиболее известных редакторов XML входят Altova XMLSpy, Stylus Studio, Oxygen XML Editor, Microsoft Visual Studio.

Редактор XML, встроенный в среду MS Visual Studio, обеспечивает удобную работу с XML-документами, включая XML-схемы и таблицы стилей XSLT.

Поскольку XML-документы содержат структурированное содержимое, то XML-редактор Visual Studio поддерживает структурирование документов: можно разворачивать или сворачивать узлы в редакторе для того, чтобы показать или скрыть содержимое узла.

Также в XML-редакторе Visual Studio полностью поддерживается проверка синтаксиса XML 1.0. Все синтаксические ошибки выделяются подчеркиванием красными волнистыми линиями. Синими волнистыми линиями подчеркиваются семантические ошибки, обнаруженные на основе проверки правильности по определению DTD или по схеме XSD.

XML-редактор с помощью технологии IntelliSense предоставляет соответствующие подсказки и помощь по форматированию. При этом редактор автоматически вставляет необходимые элементы синтаксиса (например, добавляет закрывающие теги).

Редактор XML запускается при открывании внутри Visual Studio файла с расширением .xml. Также редактор вызывается для таких распространенных расширений файлов, как .dtd, .xsd, .xsl и .config.

Для создания нового XML-документа через Visual Studio 2012 необходимо в меню File выбрать New File. В открывшемся окне шаблонов (рис. 13.2) следует выбрать шаблон XML File

(XML-файл) и нажать кнопку Open. Откроется редактор XML, в котором для нового документа автоматически добавлено объявление XML.

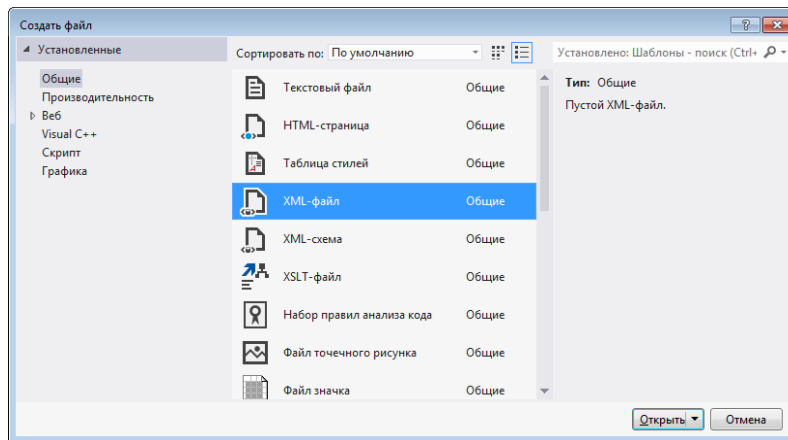


Рисунок 13.2. Окно Создать файл с выбранным шаблоном XML-файл (Visual Studio)

### Работа с XML на платформе .NET Framework.

Платформа .NET обеспечивает всестороннюю поддержку языка XML с предоставлением программисту множества способов для чтения, анализа и поиска, а также для создания кода и проверки его корректности.

В настоящее время в .NET Framework поддерживаются следующие стандарты W3C, связанные с XML:

- XML 1.0, включая поддержку DTD;
- пространства имён XML;
- схемы XML на языке XSD;
- выражения XPath;
- преобразования XSLT;
- объектные модели DOM Level 1 и DOM Level 2;
- протокол SOAP 1.1.

Помимо стандартных средств для работы с XML в .NET Framework также присутствует технология LINQ to XML.

Для работы с XML-документами MS .NET Framework 4.5 предоставляет ряд классов, относящихся к следующим основным пространствам имен (рис. 13.4):

- System.Xml – содержит наиболее важные классы для работы с XML-документами, в том числе класс XmlDocument, служащий для представления XML-документа, а также классы

XmlReader и XmlWriter, которые читают XML-данные из файла и записывают их;

- System.Xml.Serialization – содержит классы, используемые для сериализации объектов в XML-документы или в потоки;
- System.Xml.Schema – содержит классы, предназначенные для работы с XML-схемами;
- System.Xml.XPath – содержит классы, выполняющие поиск в XML-документах с помощью выражений на языке XPath;
- System.Xml.Xsl – содержит классы для работы с преобразованиями XSLT;
- System.Xml.Linq – содержит классы, использующие технологию LINQ to XML, которая предоставляет удобный интерфейс для доступа к XML-данным; работа с запросами в LINQ to XML подобна SQL.

### Объектная модель документа (DOM).

Хотя XML-документы являются текстовыми файлами, извлечение из них данных с помощью методов последовательной выборки является непрактичным, особенно когда данные должны добавляться или удаляться динамически.

При обработке XML-документов часто требуется работать с деревом документа, читая и изменяя данные в отдельных узлах, а также добавляя и удаляя узлы. Указанные возможности обеспечиваются с помощью объектной модели документа, предложенной консорциумом W3C.

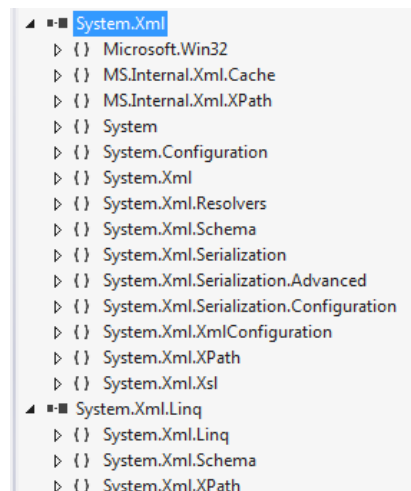


Рисунок 13.3. Пространства имен для работы с XML в окне обозревателя объектов (Visual Studio)



Объектная модель документа (DOM – Document Object Model) – это не зависящий от платформы и языка программный интерфейс (API), позволяющий программам получить доступ к содержимому HTML и XML-документов. Модель DOM позволяет читать, обрабатывать и изменять XML-документ программным образом.

Каждый компонент XML-документа представляет собой узел на дереве DOM. В модели DOM существует несколько типов узлов, основными из которых являются:

- Document – контейнер для всех узлов в дереве;
- Element – представляет узел элемента;
- Attr – атрибут элемента;
- Comment – узел комментария;
- EntityReference – представляет текст ссылки на сущность;
- CDATASection – представляет секцию CDATA;
- ProcessingInstruction – узел инструкций по обработке.

В спецификации DOM выделяют следующие уровни:

- DOM уровень 1 – описывает построение дерева документа и правила его обхода, не учитывающие особенности конкретных языков;
- DOM уровень 2 – вводит объектную модель каскадных стилей, пространства имен, фильтры, обработку событий;
- DOM уровень 3 – способы записи дерева в файл (сериализацию) и чтения его из файла (десериализацию), правила связи с языком XPath и др.

Одна из особенностей модели DOM – способ обработки атрибутов. Атрибуты в DOM не являются узлами, состоящими в родительских, дочерних и сестринских отношениях. Атрибуты считаются собственностью узла элемента и представляют собой пару «имя-значение».

Для чтения данных из XML-документов также применяют программный интерфейс SAX (Simple API for XML). В отличие от DOM, SAX не загружает в память весь XML-документ, а при помощи синтаксического анализатора последовательно читает его содержимое, генерируя события по мере чтения. Данная осо-



бенность делает SAX удобным для чтения больших документов XML, но у SAX есть ряд ограничений. Например, этот API не поддерживает структуры данных, необходимые для сложного поиска, и непригоден для модификации XML-документов.

### **Обработка XML-документов с использованием DOM.**

Пространство имен System.Xml обеспечивает программное представление XML-документов, фрагментов, узлов и наборов узлов. Оно основывается на рекомендациях базовой модели DOM уровня 1 и модели DOM уровня 2 консорциума W3C.

Основные классы пространства имен System.Xml:

- XmlDocument – представляет XML-документ, являющийся контейнером для всех узлов документа; рассматривается как корень документа, что не всегда совпадает с корневым элементом XML;
- XmlDeclaration – представляет объявление XML;
- XmlElement – представляет узел элемента;
- XmlAttribute – представляет атрибут элемента;
- XmlComment – представляет узел комментария;
- XmlText – представляет текстовое содержимое элемента или атрибута;
- XmlCDataSection – представляет секцию CDATA.

Все указанные классы являются производными от абстрактного класса XmlNode, представляющего произвольный узел XML.

У класса XmlNode можно отметить следующие свойства:

- XmlNodeList ChildNodes – возвращает все дочерние узлы данного узла в форме списка типа XmlNodeList;
- string InnerXml – возвращает или задаёт разметку XML, отражающую дочерние узлы данного узла;
- string InnerText – возвращает или задаёт текстовое содержимое данного узла;
- XmlNode ParentNode – возвращает родительский узел данного узла (только для тех узлов, которые могут его иметь).

Можно отметить следующие методы класса XmlNode, используемые для вставки узлов в дерево XML-документа:

- `XmlNode AppendChild(XmlNode newNode)` – добавляет указанный узел `newNode` в конец списка дочерних узлов данного узла;
- `XmlNode PrependChild(XmlNode newNode)` – добавляет узел `newNode` в начало списка дочерних узлов заданного узла.

Главным классом, реализующим модель DOM, является класс `XmlDocument`, который представляет XML-документ в целом.

К основным методам класса `XmlDocument` относятся:

- `XmlDeclaration CreateXmlDeclaration(string version, string encoding, string standalone)` – создает объект `XmlDeclaration` с указанными значениями атрибутов `version`, `encoding` и `standalone`;
- `XmlComment CreateComment(string data)` – создает объект `XmlComment`, содержащий указанные данные `data`;
- `XmlElement CreateElement(string name)` – создает объект `XmlElement` с заданным именем `name`;
- `XmlAttribute CreateAttribute(string name)` – создает объект `XmlAttribute` с заданным именем `name`;
- `XmlCDataSection CreateCDataSection(string data)` – создает объект `XmlCDataSection`, содержащий указанные данные `data`;
- `void Load(string uri)` – загружает XML-документ из заданного URI-адреса;
- `void LoadXml(string xml)` – загружает XML-документ из строковых данных `xml`;
- `void Save(string uri)` – сохраняет XML-документ в файле с указанным URI.

Класс `XmlElement` имеет схожий с `XmlDocument` набор методов и свойств. В качестве метода класса `XmlElement` можно отметить `SetAttribute`, который устанавливает имя и значение атрибута для заданного элемента.

Атрибуты в модели DOM являются не узлами, а свойствами узла элемента и содержатся в коллекции `XmlAttributeCollection`, связанной с элементом. Для создания новых атрибутов можно использовать два способа:

- использовать метод `SetAttribute` для добавления атрибута в коллекцию атрибутов выбранного элемента;

- создать экземпляр класса `XmlAttribute` с помощью метода `CreateAttribute()` и использовать метод `SetAttributeNode()`, чтобы добавить полученный узел в коллекцию атрибутов выбранного элемента.

### **Классы `XmlReader` и `XmlWriter`.**

Пространство имён `System.Xml` предоставляет абстрактные классы `XmlWriter` и `XmlReader` и, которые обеспечивают интерфейс для чтения и записи XML-данных, а также поддерживают синтаксический анализ XML-документов.

Хотя платформа .NET Framework располагает конкретными реализациями классов `XmlReader` и `XmlWriter` (например, `XmlTextReader`, `XmlNodeReader`, `XmlTextWriter` и др.), рекомендуется создавать экземпляры `XmlReader` и `XmlWriter` с использованием статического метода `Create`. Это позволяет получить преимущества от всех новых функций, добавленных к классам `XmlReader` и `XmlWriter` в .NET Framework 2.0.

Для классов `XmlReader` и `XmlWriter` доступно несколько перегруженных версий метода `Create`. В простейшем случае объекты этих классов создаются следующим образом:

```
XmlReader xmlRdr = XmlReader.Create(uri);  
XmlWriter xmlWrt = XmlWriter.Create(uri);
```

где `uri` – строка, содержащая путь к XML-документу.

Класс `XmlReader` читает XML-данные из потока или файла. Он предоставляет однопроходный доступ только для чтения к XML-данным без кэширования.

К основным свойствам и методам класса `XmlReader` относятся:

- `XmlNodeType NodeType` – возвращает тип текущего узла (перечисление `XmlNodeType` содержит такие константы, как `Document`, `Element`, `Comment`, `CDATA` и др.);
- `string Name` – возвращает имя текущего узла;
- `string Value` – возвращает текстовое содержимое текущего узла;
- `void Close()` – переводит экземпляр класса `XmlReader` в состояние «закрыт» (`Closed`);

- `bool Read()` – читает следующий узел из потока и возвращает `true`, если узел существует, в противном случае возвращает `false`;
- `string ReadString()` – возвращает содержимое элемента или текстового узла в виде строки;
- `string ReadInnerXml()` – читает и возвращает всё содержимое текущего узла в форме строки, включая разметку.

Для обработки атрибутов класс `XmlReader` имеет следующие свойства и методы:

- `AttributeCount` – возвращает число атрибутов в элементе;
- `GetAttribute()` – возвращает значение атрибута;
- `bool HasAttributes()` – возвращает значение, показывающее, имеются ли атрибуты у текущего узла;
- `MoveToNextAttribute` – переходит к следующему атрибуту данного узла и возвращает `true`, если атрибут существует, в противном случае возвращает `false`.

При создании объекта `XmlReader` с помощью метода `Create` также можно указать дополнительные настройки, задаваемые экземпляром класса `XmlReaderSettings`:

```
XmlReaderSettings xmlRdS = new XmlReaderSettings();
XmlReader xmlRdr = XmlReader.Create(uri, xmlRdS);
```

Основные свойства класса `XmlReaderSettings`:

- `bool IgnoreWhitespace` – возвращает или устанавливает значение, определяющее будут ли игнорироваться незначимые символы пробела;
- `bool IgnoreComments` – возвращает или устанавливает значение, указывающее, следует ли игнорировать комментарии.

Класс `XmlWriter` предоставляет последовательный однопоточный доступ для записи XML-данных без кэширования в потоки и файлы. При помощи `XmlWriter` можно записывать несколько документов в один и тот же выходной поток.

Основными методами класса `XmlWriter` являются:

- `void WriteStartDocument(bool standalone)` – записывает объявление XML с номером версии 1.0 и выбранным значением атрибута `standalone`;

- `void WriteComment(string text)` – записывает комментарий;
- `void WriteStartElement(string name)` – записывает открывающий тег элемента с указанным именем `name`;
- `void WriteEndElement()` – записывает закрывающий тег элемента;
- `void WriteString(string text)` – записывает текстовое содержимое элемента;
- `void WriteAttributeString(string name, string value)` – записывает атрибут с указанным именем `name` и значением `value`;
- `void WriteCData(string text)` – записывает секцию CDATA с указанным текстом;
- `void Close()` – переводит объект `XmlReader` в состояние «закрит» (`Closed`) и закрывает связанный с ним поток.

Для создания `XmlWriter` с помощью метода `Create` также можно использовать дополнительные настройки, задаваемые экземпляром класса `XmlWriterSettings`:

```
XmlWriterSettings xmlWrS = new XmlWriterSettings();
XmlWriter xmlWrt = XmlWriter.Create(uri, xmlWrS);
```

Можно отметить следующие свойства класса `XmlWriterSettings`:

- `bool Indent` – возвращает или задаёт значение, указывающее, следует ли использовать отступ для элементов XML (значение по умолчанию `false`); при задании значения `true` код XML-документа будет записан с отступами;
- `Encoding Encoding` – получает или устанавливает тип используемой кодировки текста (значение по умолчанию `Encoding.UTF8`).

**Сериализация объектов в XML. Десериализация объектов из XML. Класс `XmlSerializer`.**

При разработке объектно-ориентированной программы может возникнуть задача сохранения определённых объектов и их последующего восстановления. Для решения этой задачи можно использовать такие средства, как сериализация и десериализация.

Сериализация представляет собой процесс преобразования объекта в поток байтов. Обратный процесс называется десериализацией.

Сериализация позволяет сохранить состояние объекта в файле и воссоздавать его при необходимости. Также с помощью сериализации разработчик может передавать объект удаленному приложению посредством веб-службы.

При разработке приложений для .NET Framework можно использовать двоичную сериализацию или XML-сериализацию.

При двоичной сериализации используется двоичная кодировка, обеспечивающая компактную сериализацию объекта для хранения в двоичных файлах или передачи в сетевых потоках. Данный способ характеризуется высокой скоростью выполнения.

XML-сериализация обеспечивает сохранение объектов в XML-документах. В отличие от двоичной сериализации в данном способе имеется более удобочитаемый код, а также большая гибкость совместного доступа и использования объекта в целях взаимодействия.

Сериализация работает только с открытыми типами и открытыми элементами этих типов. Кроме того, в классе сериализуемого объекта должен быть в явном виде объявлен конструктор по умолчанию.

Пространство имен `System.Xml.Serialization` содержит класс `XmlSerializer`, позволяющий сериализовать объекты в XML и десериализовать объекты из XML.

При создании экземпляра класса `XmlSerializer` необходимо указать информацию об объекте, который требуется сериализовать.

Основными методами класса `XmlSerializer` являются:

- `void Serialize(XmlWriter xmlWrt, object obj)` – выполняет сериализацию объекта `obj` в XML;
- `object Deserialize(XmlReader xmlRdr)` – выполняет десериализацию объекта из XML;
- `bool CanDeserialize(XmlReader xmlRdr)` – возвращает значение, указывающее возможность выполнения десериализации объекта из XML.

По умолчанию класс `XmlSerializer` сериализует все открытые поля и автоматические свойства как элементы XML.

### **Атрибуты и атрибутные классы C#. Атрибуты сериализации объектов.**

Атрибуты определяют декларативную информацию, не относящуюся к программному коду. Эта информация преобразуется компилятором в метаинформацию, сопровождающую сборку.

Исполняемая программа может получить доступ к метаинформации сборки с помощью процесса, называемого отражением (англ., reflection). В .NET Framework есть классы, позволяющие анализировать метаинформацию и тем самым влиять на дальнейший ход выполнения программы.

Атрибуты представляют собой объекты специального вида классов, называемых атрибутными классами. Все атрибутные классы являются потомками абстрактного класса Attribute. В библиотеке FCL определено большое число атрибутных классов, расположенных в разных пространствах имён. Имена таких классов заканчиваются суффиксом Attribute.

Атрибуты могут задаваться при объявлении разных сущностей языка C# – типов (классов, структур, интерфейсов и др.), полей, методов, параметров метода и возвращаемых значений. Кроме того, атрибуты могут быть заданы для модуля и всей сборки.

Атрибут, связываемый с сущностью, заключается в квадратные скобки, непосредственно предшествуя описанию сущности:

[Имя атрибутного класса]  
Объявление сущности

С одной сущностью может быть связано несколько атрибутов.

Программист может сам создавать собственные атрибутные классы и связывать сущности программы с атрибутами этого класса. Создаваемый атрибутный класс должен являться потомком Attribute и иметь соответствующий суффикс в имени.

Атрибут [Serializable] позволяет снабдить класс стандартным механизмом сериализации. Атрибут [Serializable] может быть задан для классов, структур, перечислений и делегатов.

Если поля какого-либо класса не должны (или не могут) участвовать в сериализации, то такие поля можно пометить атри-

бутом [NonSerialized]. Это позволяет сократить размер хранимых данных, исключив сохранение фиксированных значений, случайных значений и кратковременных данных.

В пространстве имён System.Xml.Serialization содержатся следующие основные атрибуты:

- [XmlElement] – указывает, что необходимо выполнить сериализацию элемента класса в качестве XML-элемента;
- [XmlAttribute] – указывает, что необходимо выполнить сериализацию элемента класса в качестве XML-атрибута;
- [XmlText] – указывает, что элемент класса должен обрабатываться как текстовый узел XML, когда содержащий его класс сериализуется или десериализуется.

## **МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

### **Разработка XML-документа.**

Требуется разработать XML-документ, содержащий сведения об оплате за коммунальные услуги. В качестве сведений выступает информация о жилых домах (код, улица, номер, описание), квартирах (код, номер, площадь), жильцах (код, ФИО, дата рождения), показаниях счетчиков (дата, расход холодной и горячей воды в м<sup>3</sup>, расход электроэнергии в квт·ч) и по квартплате (дата, всего, пеня).

Идентификаторы (коды домов, квартир и жильцов), единицы измерения и даты разместим в атрибутах. Описания домов расположим в секциях CDATA. Остальные данные будут являться текстовыми узлами (Text) элементов.

Дерево XML-документа представлено на рис. 13.4. Код XML-документа, построенного в соответствии с деревом, приведён в листинге 13.2.



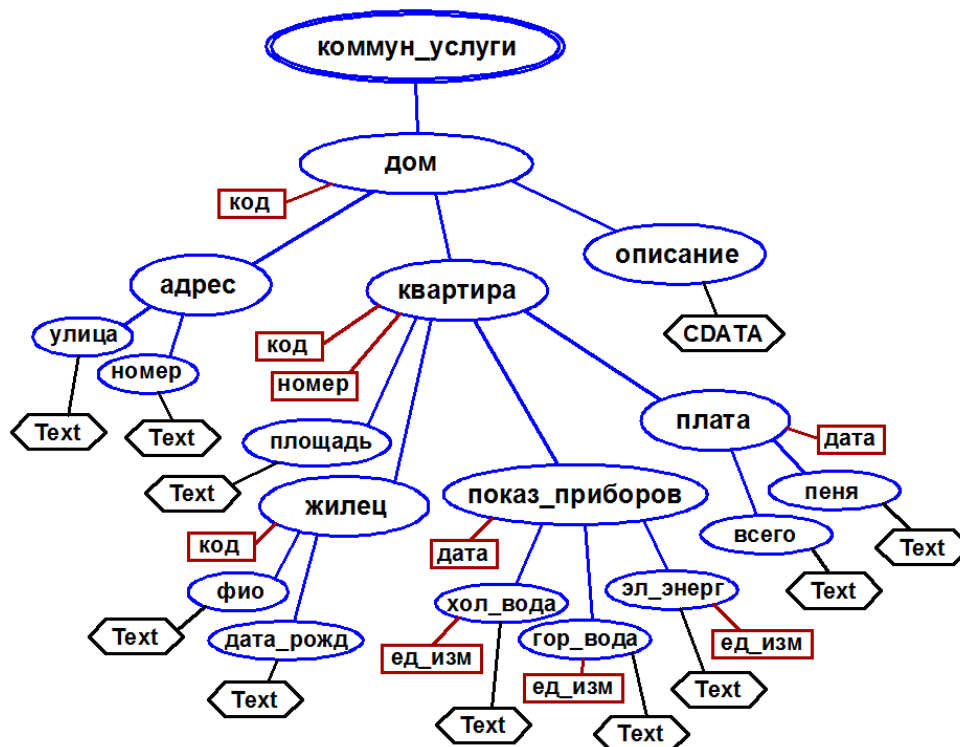


Рисунок 13.4. Дерево XML-документа

## Листинг 13.2. Код XML-документа «CommService.xml»

---

```

<?xml version="1.0" encoding="utf-8"?>
<!-- Данные о коммунальных услугах и их оплате -->
<коммунал_услуги>
  <дом код="h18">
    <адрес>
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес>
    <описание>
      <![CDATA[Крупнопанельный 9-этажный дом. Построен в 1978 г. Физиче-
ский износ 14% (дата обслед. 15.06.2008 г.)]]>
    </описание>
    <квартира код="a234" номер="57">
      <площадь ед_изм="м2">28</площадь>
      <жилец код="c11568">
        <фио>Костенко Игорь Сергеевич</фио>
        <дата_рожд>1978-11-10</дата_рожд>
      </жилец>
      <показ_приборов дата="2014-05-02">
        <хол_вода ед_изм="м3">19.04</хол_вода>
        <гор_вода ед_изм="м3">6.89</гор_вода>
        <эл_энерг ед_изм="квтч">39.27</эл_энерг>
      </показ_приборов>
      <плата дата="2015-05-04">
        <всего>1896.45</всего>
        <пеня>0</пеня>
      </плата>
    </квартира>
  </дом>
</коммунал_услуги>

```

---

---

```

<квартиракод="a236"номер="59">
<площадьед_изм="м2">42</площадь>
<желецкод="с27788">
<фио>Соловьев Дмитрий Андреевич</фио>
<дата_рожд>1988-07-22</дата_рожд>
</желец>
<желецкод="с27789">
<фио>Соловьева Елена Николаевна</фио>
<дата_рожд>1989-10-03</дата_рожд>
</желец>
<показ_приборовдата="2015-05-04">
<хол_водаед_изм="м3">172.24</хол_вода>
<гор_водаед_изм="м3">65.07</гор_вода>
<эл_энергед_изм="квтч">125.69</эл_энерг>
</показ_приборов>
<платадата="2015-05-05">
<всего>2396.45</всего>
<пеня>0</пеня>
</плата>
</квартира>
</дом>
<домкод="h72">
<адрес>
<улица>Терешковой</улица>
<номер>12</номер>
</адрес>
<описание>
<![CDATA[Кирпичный 5-этажный дом. Построен в 1972 г. Физический из-
нос 18% (дата обслед. 24.08.2003 г.)]]>
</описание>
<квартиракод="a358"номер="35">
<площадьед_изм="м2">36</площадь>
<желецкод="с34670">
<фио>Курганков Георгий Михайлович</фио>
<дата_рожд>1972-02-26</дата_рожд>
</желец>
<показ_приборовдата="2015-08-05">
<хол_водаед_изм="м3">405.21</хол_вода>
<гор_водаед_изм="м3">159.35</гор_вода>
<эл_энергед_изм="квтч">209.76</эл_энерг>
</показ_приборов>
<платадата="2015-05-12">
<всего>50896.56</всего>
<пеня>4507.34</пеня>
</плата>
</квартира>
</дом>
</коммун_услуги>

```

---

### **Создание и загрузка XML-документа с помощью модели DOM на языке C#.**

Создадим одну ветвь XML-документа из примера предыдущего примера с помощью консольного приложения на языке C#, используя модель DOM. Кроме того, приложение должно загружать код полученного XML-документа в окно консоли.

Для этого реализуем в классе Program консольного приложения следующие методы:

- `void XmlCreate(string uri)` – создаёт XML-документ и сохраняет его по заданному пути `uri`;
- `void XmlLoad(string uri)` – считывает XML-документ по заданному пути `uri` и выводит код документа в окно консоли.

Исходный код методов `Main()`, `XmlCreate()` и `XmlLoad()` приведён в листинге 13.3.

Код полученного в ходе работы программы XML-документа показан в листинге 13.4.

Листинг 13.3. Исходный код методов `Main()` и `XmlCreate()` консольного приложения

```

6 | using System.Xml; // Импорт пространства имён для работы с XML
7
8 | namespace XmlDOM
9 | {
10 |     class Program
11 |     {
12 |         static void Main(string[] args)
13 |         {
14 |             Console.Title = "Создание и чтение XML-документа с использованием модели DOM";
15 |             // URL, задающий путь к XML-документу
16 |             string url = @"H:\XML\FragmCommService.xml";
17 |             XmlCreate(url);
18 |             XmlLoad(url);
19 |             Console.Read();
20 |         }
21 |
22 |         /// <summary>
23 |         /// Создать XML-документ "с нуля", используя DOM
24 |         /// </summary>
25 |         /// <param name="url">Путь к XML-документу</param>
26 |         static void XmlCreate(string url)
27 |         {
28 |             XmlDocument xmlDoc = new XmlDocument();
29 |             // Добавление объявления XML в XML-документ
30 |             xmlDoc.AppendChild(xmlDoc.CreateXmlDeclaration("1.0", "utf-8", "yes"));
31 |             // Добавление комментария в XML-документ
32 |             xmlDoc.AppendChild(xmlDoc.CreateComment("Данные о коммунальных услугах " +
33 |                 "и их оплате"));
34 |             XmlElement rootElem = xmlDoc.CreateElement("коммун_услуги");
35 |             // Добавление корневого элемента в XML-документ
36 |             xmlDoc.AppendChild(rootElem);
37 |             // Задание пространства имён в корневом элементе
38 |             rootElem.SetAttribute("xmlns", "http://www.g42.kommunal.ru/uslugi/oplata");
39 |
40 |             XmlElement houseElem = xmlDoc.CreateElement("дом");
41 |             rootElem.AppendChild(houseElem);
42 |             houseElem.SetAttribute("код", "h18"); // Задание атрибута и его значения
43 |
44 |             XmlElement adrElem = xmlDoc.CreateElement("адрес");
45 |             houseElem.AppendChild(adrElem);
46 |             XmlElement streetElem = xmlDoc.CreateElement("улица");
47 |             // Задание текстового содержимого элемента-контейнера
48 |             streetElem.InnerText = "Волгоградская";
49 |             adrElem.AppendChild(streetElem);
50 |             XmlElement numberElem = xmlDoc.CreateElement("номер");
51 |             numberElem.InnerText = "8";
52 |             adrElem.AppendChild(numberElem);
53 |
54 |             XmlElement dscElem = xmlDoc.CreateElement("описание");
55 |             houseElem.AppendChild(dscElem);
56 |             // Задание секции CDATA
57 |             XmlCDATASection cData = xmlDoc.CreateCDATASection("Крупнопанельный " +
58 |                 "9-этажный дом. Построен в 1978 г. Физический износ 14% " +
59 |                 "(дата обслед. 15.06.2008 г.)");
60 |             dscElem.AppendChild(cData);

```

```

62     XmlElement apartElem = xmlDoc.CreateElement("квартира");
63     houseElem.AppendChild(apartElem);
64     apartElem.SetAttribute("код", "a234");
65     apartElem.SetAttribute("номер", "57");
66     XmlElement squarElem = xmlDoc.CreateElement("площадь");
67     apartElem.AppendChild(squarElem);
68     squarElem.SetAttribute("ед_изм", "м2");
69     squarElem.InnerText = "28";
70
71     XmlElement personElem = xmlDoc.CreateElement("желец");
72     apartElem.AppendChild(personElem);
73     personElem.SetAttribute("код", "c11568");
74     XmlElement nameElem = xmlDoc.CreateElement("фio");
75     personElem.AppendChild(nameElem);
76     nameElem.InnerText = "Костенко Игорь Сергеевич";
77     XmlElement birthElem = xmlDoc.CreateElement("дата_рожд");
78     personElem.AppendChild(birthElem);
79     birthElem.InnerText = "10.11.1978";
80
81     XmlElement consumElem = xmlDoc.CreateElement("показ_приборов");
82     apartElem.AppendChild(consumElem);
83     consumElem.SetAttribute("дата", "02.05.2014");
84     XmlElement coldWtElem = xmlDoc.CreateElement("хол_вода");
85     consumElem.AppendChild(coldWtElem);
86     coldWtElem.SetAttribute("ед_изм", "м3");
87     coldWtElem.InnerText = "19,04";
88     XmlElement hotWtElem = xmlDoc.CreateElement("гор_вода");
89     consumElem.AppendChild(hotWtElem);
90     hotWtElem.SetAttribute("ед_изм", "м3");
91     hotWtElem.InnerText = "6,89";
92     XmlElement powerElem = xmlDoc.CreateElement("эл_энерг");
93     consumElem.AppendChild(powerElem);
94     powerElem.SetAttribute("ед_изм", "квтч");
95     powerElem.InnerText = "39,27";
96
97     XmlElement paymtElem = xmlDoc.CreateElement("плата");
98     apartElem.AppendChild(paymtElem);
99     paymtElem.SetAttribute("дата", "04.05.2014");
100    XmlElement totalElem = xmlDoc.CreateElement("всего");
101    paymtElem.AppendChild(totalElem);
102    totalElem.InnerText = "1896,49";
103    XmlElement penalElem = xmlDoc.CreateElement("пеня");
104    paymtElem.AppendChild(penalElem);
105    penalElem.InnerText = "0";
106
107    // Сохранение документа в XML-файле
108    xmlDoc.Save(url);
109    Console.WriteLine("XML-документ создан по адресу: " + url);
110    Console.Read();
111 }
112
113 /// <summary>
114 /// Загрузить XML, используя DOM
115 /// </summary>
116 /// <param name="url">Путь к XML-документу</param>
117 static void XmlLoad(string url)
118 {
119     XmlDocument xmlDoc = new XmlDocument();
120     xmlDoc.Load(url);
121     Console.WriteLine("Код XML-документа:\n" + xmlDoc.InnerXml);
122     Console.Read();
123 }
124 }

```

Листинг 13.4. Код полученного XML-документа  
FragmCommService.xml

```

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--Данные о коммунальных услугах и их оплате-->
<коммун_услуги xmlns="http://www.g42.kommunal.ru/uslugi/oplata">
  <дом код="h18">
    <адрес>
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес>
    <описание><![CDATA[Крупнопанельный 9-этажный дом. Построен в 1978 г. Физический износ 14% (дат
    <квартира код="a234" номер="57">
      <площадь ед_изм="м2">28</площадь>
      <жильец код="c11568">
        <фio>Костенко Игорь Сергеевич</фio>
        <дата_рожд>10.11.1978</дата_рожд>
      </жильец>
      <показ_приборов дата="02.05.2014">
        <хол_вода ед_изм="м3">19,04</хол_вода>
        <гор_вода ед_изм="м3">6,89</гор_вода>
        <эл_энерг ед_изм="квтч">39,27</эл_энерг>
      </показ_приборов>
      <плата дата="04.05.2014">
        <всего>1896,49</всего>
        <пеня>0</пеня>
      </плата>
    </квартира>
  </дом>
</коммун_услуги>

```

Результат работы консольного приложения представлен на рис. 13.5.

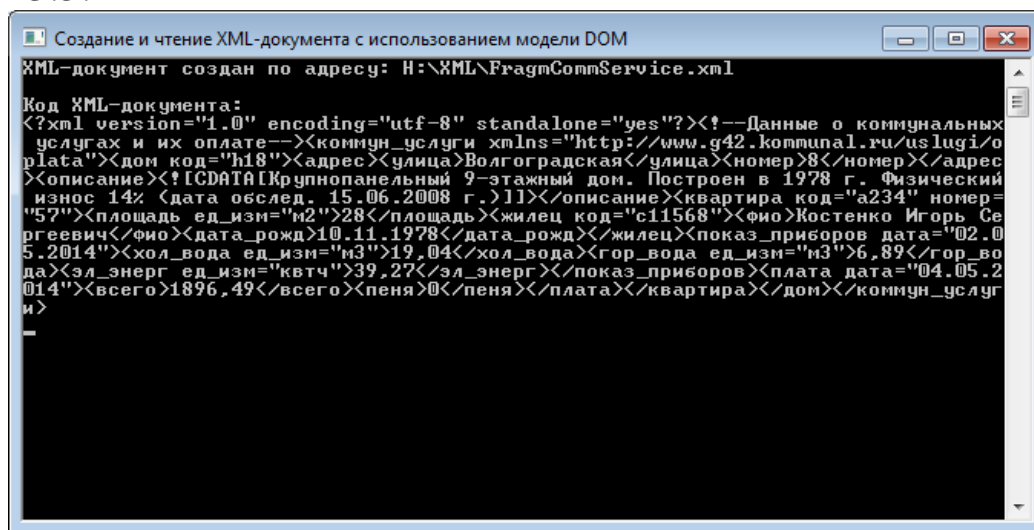


Рисунок 13.5. Результат работы консольного приложения

**Сериализация и десериализация объекта с помощью консольного приложения на языке C#.**

Требуется разработать консольное приложение, которое осуществляет сериализацию в XML и десериализацию из XML объекта класса Person (человек).

В классе `Person` должны присутствовать следующие данные: имя, фамилия, пол, дата рождения, серия и номер паспорта, адрес проживания.

Для задания адреса проживания создадим класс `Address`, в котором будут определены следующие данные: почтовый индекс, город, улица, номер дома, номер квартиры.

В классе `Person` будет задана ссылка на экземпляр класса `Address`.

Сериализация таких данных, как серия и номер паспорта, а также почтовый индекс будет выполняться в атрибуты XML. Остальные данные будут сериализованы, как текстовое содержимое XML-элементов.

Исходный код класса `Address` представлен в листинге 13.5, а класса `Person` – в листинге 13.6.

## Листинг 13.5. Исходный код класса Address

```

6 using System.Xml.Serialization; // Импорт пространства имён для сериализации в XML
7
8 namespace XmlSerialization
9 {
10     /// <summary>
11     /// Представляет адреса мест проживания
12     /// </summary>
13     [Serializable]
14     public class Address
15     {
16         /// <summary>
17         /// Почтовый индекс (сериализуется в атрибут)
18         /// </summary>
19         [XmlAttribute]
20         public string ZipCode { get; set; }
21
22         /// <summary>
23         /// Город
24         /// </summary>
25         public string City { get; set; }
26
27         /// <summary>
28         /// Улица
29         /// </summary>
30         public string Street { get; set; }
31
32         /// <summary>
33         /// Номер дома
34         /// </summary>
35         public string House { get; set; }
36
37         /// <summary>
38         /// Номер квартиры
39         /// </summary>
40         public string Apartment { get; set; }
41
42         /// <summary>
43         /// Конструктор по умолчанию
44         /// </summary>
45         public Address()
46         {
47             ZipCode = "650000";
48             City = "Кемерово";
49             Street = "пр. Ленина";
50             House = "10";
51             Apartment = "10";
52         }
53
54         /// <summary>
55         /// Возвращает строку с данными об адресе проживания
56         /// </summary>
57         /// <returns>Адрес места проживания</returns>
58         public override string ToString()
59         {
60             return (string.Format("- почтовый индекс: {0}\n" +
61                                   "- город: {1}\n" +
62                                   "- улица: {2}\n" +
63                                   "- дом: {3}\n" +
64                                   "- квартира: {4}\n",
65                                   ZipCode, City, Street, House, Apartment));
66         }
67     }
68 }

```



## Листинг 13.6. Исходный код класса Person

```

6 using System.Xml.Serialization; // Импорт пространства имён для сериализации в XML
7
8 namespace XmlSerialization
9 {
10     /// <summary>
11     /// Представляет людей
12     /// </summary>
13     [Serializable]
14     public class Person
15     {
16         /// <summary>
17         /// Имя
18         /// </summary>
19         public string FirstName { get; set; }
20
21         /// <summary>
22         /// Фамилия
23         /// </summary>
24         public string LastName { get; set; }
25
26         /// <summary>
27         /// Пол
28         /// </summary>
29         public string Gender { get; set; }
30
31         /// <summary>
32         /// Дата рождения
33         /// </summary>
34         public DateTime BirthDate { get; set; }
35
36         /// <summary>
37         /// Серия и номер паспорта (сериализуется в атрибут)
38         /// </summary>
39         [XmlAttribute]
40         public string Passport { get; set; }
41
42         /// <summary>
43         /// Адрес проживания
44         /// </summary>
45         public Address Address { get; set; }
46
47         /// <summary>
48         /// Номер телефона
49         /// </summary>
50         public string Phone { get; set; }
51
52         /// <summary>
53         /// Конструктор по умолчанию
54         /// </summary>
55         public Person()
56         {
57             FirstName = "Иван";
58             LastName = "Иванов";
59             Gender = "М";
60             BirthDate = new DateTime(1990, 1, 1);
61             Passport = "32 02 000000";
62             Address = new Address();
63             Phone = "0-000-000-0000";
64         }

```

```

66     /// <summary>
67     /// Возвращает строку с данными о человеке
68     /// </summary>
69     /// <returns>Данные о человеке</returns>
70     public override string ToString()
71     {
72         return (string.Format("Данные о человеке:\n" +
73             "** Имя: {0}\n" +
74             "** Фамилия: {1}\n" +
75             "** Пол: {2}\n" +
76             "** Дата рождения: {3}\n" +
77             "** Серия и номер паспорта: {4}\n" +
78             "** Телефон: {5}\n" +
79             "** Адрес:\n" +
80             "{6}", FirstName, LastName, Gender, BirthDate,
81             Passport, Phone, Address.ToString()));
82     }
83 }

```

Исходный код класса Program консольного приложения приведён в листинге 13.7.

Листинг 13.7. Исходный код класса Program консольного приложения

```

6  using System.Xml;
7  using System.Xml.Serialization; // Импорт пространства имён для XML-сериализации
8
9  namespace XmlSerialization
10 {
11     class Program
12     {
13         static void Main(string[] args)
14         {
15             Console.Title = "Сериализация в XML";
16
17             Address addr = new Address()
18             {
19                 ZipCode = "651001",
20                 City = "Новокузнецк",
21                 Street = "ул. Metallургов",
22                 House = "32а",
23                 Apartment = "46"
24             };
25
26             Person pers = new Person()
27             {
28                 FirstName = "Екатерина",
29                 LastName = "Николаева",
30                 BirthDate = new DateTime(1988, 7, 25),
31                 Gender = "Ж",
32                 Passport = "32 02 123456",
33                 Address = addr,
34                 Phone = "8-912-174-3845"
35             };

```

```

37         Console.WriteLine("До сериализации в XML:\n" + pers.ToString());
38
39         string uri = @"H:\XML\Person.xml"; // URI XML-документа
40         // Сериализуем объект pers в XML
41         Serialize(uri, pers);
42
43         XmlDocument xmlDoc = new XmlDocument();
44         xmlDoc.Load(uri);
45         Console.WriteLine("Код полученного XML-документа:\n"
46             + xmlDoc.InnerXml + "\n");
47
48         // Десериализуем из XML объект newPers
49         Person newPers = Deserialize(uri);
50
51         Console.WriteLine("После десериализации из XML:\n" + newPers.ToString());
52
53         Console.Read();
54     }
55
56     /// <summary>
57     /// Сериализует объект в XML-файл
58     /// </summary>
59     /// <param name="uri">URI XML-файла</param>
60     /// <param name="pers">Сериализуемый объект типа Person</param>
61     static void Serialize(string uri, Person pers)
62     {
63         // Объект для XML-сериализации и десериализации
64         XmlSerializer sr = new XmlSerializer(typeof(Person));
65         // Объект с настройками для XmlReader
66         XmlWriterSettings xmlWrS = new XmlWriterSettings();
67         xmlWrS.Indent = true; // Задаём отступ для XML-элементов
68         // Объект для записи XML-файлов
69         XmlWriter xmlWrt = XmlWriter.Create(uri, xmlWrS);
70         // Сериализуем объект pers в XML
71         sr.Serialize(xmlWrt, pers);
72         xmlWrt.Close(); // Закрываем поток данных для xmlWrt
73     }
74
75     /// <summary>
76     /// Десериализует объект из XML-файла
77     /// </summary>
78     /// <param name="uri">URI XML-файла</param>
79     static Person Deserialize(string uri)
80     {
81         // Объект для XML-сериализации и десериализации
82         XmlSerializer xmlSrz = new XmlSerializer(typeof(Person));
83         // Объект с настройками для XmlReader
84         XmlReaderSettings xmlRdS = new XmlReaderSettings();
85         // Объект для чтения XML-файлов
86         XmlReader xmlRdr = XmlReader.Create(uri, xmlRdS);
87         // Десериализуем объект типа Person
88         Person pers = xmlSrz.Deserialize(xmlRdr) as Person;
89         // Закрываем поток данных для xmlRdr
90         xmlRdr.Close();
91         return pers;
92     }
93 }

```

Результат работы консольного приложения представлен на рис. 13.6. Полученный с помощью сериализации XML-документ показан в листинге 13.8.

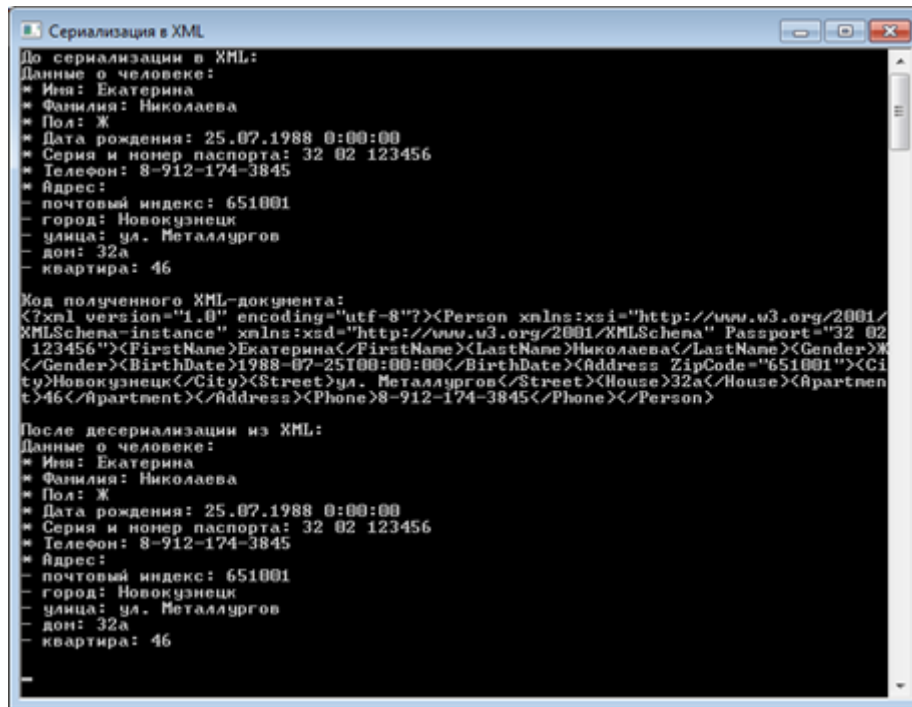


Рисунок 13.6. Результат работы приложения в окне консоли

Листинг 13.8. Код XML-документа, полученного путем сериализации экземпляра класса Person

```
<?xml version="1.0" encoding="utf-8"?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  Passport="32 02 123456">
  <FirstName>Екатерина</FirstName>
  <LastName>Николаева</LastName>
  <Gender>Ж</Gender>
  <BirthDate>1988-07-25T00:00:00</BirthDate>
  <Address ZipCode="651001">
    <City>Новокузнецк</City>
    <Street>ул. Металлургов</Street>
    <House>32а</House>
    <Apartment>46</Apartment>
  </Address>
  <Phone>8-912-174-3845</Phone>
</Person>
```

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

### Основные этапы выполнения работы.

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. На основе предложенного задания(табл. 13.1) разработать иерархическую структуру данных для XML-документа и изобразить её в виде дерева. Согласовать дерево документа с преподавателем и при необходимости внести в него рекомендуемые изменения.

3. На основе полученного дерева создать правильно построенный XML-документ. Для основных элементов первого уровня предусмотреть не менее двух экземпляров, для основных элементов последующих уровней – два или три экземпляра.

4. Используя классы пространства имен System.Xml, реализующие модель DOM, написать и отладить консольное приложение на языке C# для создания XML-документа «с нуля». Предусмотреть создание одной ветки XML-документа, полученного в п. 3.

5. Разработать класс, содержащий указанные данные (табл. 13.2). Создать в консольном приложении на языке C# экземпляр заданного класса и произвести его сериализацию в XML, а также десериализацию из XML. Часть данных должна быть сериализована в форме атрибутов.

6. Оформить и защитить отчет по лабораторной работе.

Таблица 13.1

Варианты заданий для разработки XML-документов

Вариант	Данные для разработки XML-документа
1,13	<b>Расписание рейсов междугородных автобусов.</b> Документ должен содержать сведения о пунктах отправления (код, название (например, «Кемеровский АВ», «Прокопьевская АС»), пунктах прибытия (код, название), рейсах (код, дни недели (например, «ежедневно», «еженедельно: пн, чт»), время отправления, время прибытия, цена)
2,14	<b>Товарный склад.</b> Документ должен содержать сведения о стеллажах (код, тип, число ярусов), товарах (код, наименование, количество, вес, дата поступления, дата выпуска, срок хранения, цена и др.), фирме-производителе (название, контактная информация (телефон, адрес))
3,15	<b>Ресторан.</b> Документ должен содержать сведения о сделанных заказах (код, дата, время, стол, выполнение (да нет) и др.), блюдах (код, название, время приготовления, цена) и ингредиентах (код, название, объем)

Продолжение таблицы 13.1

4,16	<p><b>Расписание занятий.</b></p> <p>Документ должен содержать сведения по дням недели о проводимых занятиях (номер пары, название учебной дисциплины, тип занятия (лекция, практика, лабораторная), аудитория (номер), ФИО и должность преподавателя). Расписание отличается на четных и нечетных неделях</p>
5,17	<p><b>Книжный магазин.</b></p> <p>Документ должен содержать сведения об отделах книжного магазина (код, название), книгах (код, название, авторы, издательство, год, число страниц, цена) и журналах (код, название, год, номер, цена)</p>
6,18	<p><b>Кинотеатр.</b></p> <p>Документ должен содержать сведения о зрительных залах (код, номер, число мест), фильмах (код, название, год выпуска, кино-компания, режиссёр, актеры и др.), сеансах (код, дата, время начала, цена билета)</p>
7,19	<p><b>Проектно-строительная компания.</b></p> <p>Документ должен содержать сведения об отделах (код, название), сотрудниках (код, фамилия, имя, отчество, пол, дата рождения, дата устройства, должность) и выполняемых ими проектах (код, наименование, срок выдачи, стоимость)</p>
8,20	<p><b>Обувной магазин.</b></p> <p>Документ должен содержать сведения об отделах (код, название), обуви (код, название, размер, материал, цвет, производитель, цена, скидка) и аксессуарах (код, название, цена)</p>
9,21	<p><b>Перевозка грузов.</b></p> <p>Документ должен содержать сведения о водителях (код, ФИО, дата рождения, телефон и др.), автомобилях (код, марка, тип кузова и др.) и заказах (код, дата/время, адрес доставки, тип груза, вес груза и др.)</p>

Продолжение таблицы 13.1

10,22	<b>Поликлиника.</b> Документ должен содержать сведения о врачах (код, ФИО, специальность), пациентах (код, ФИО, дата рождения, пол, номер полиса) и обращениях (код, дата обращения, болезнь, продолжительность, результат лечения и др.)
11,23	<b>Банковские услуги.</b> Документ должен содержать сведения о клиентах (код, фамилия, имя, отчество, паспортные данные, телефон), кредитах (код, вид, сумма, дата, срок, процентная ставка) и вкладах (код, вид, номер счёта, сумма, процентная ставка, дата)
12,24	<b>Прогноз погоды.</b> Документ должен содержать сведения о днях (дата, день недели), времени суток (код, тип (утро, день, вечер, ночь), температура, давление, влажность), ветре (направление, скорость) и атмосферных явлениях (облачность (ясно, малооблачно, облачно, пасмурно), осадки)

Таблица 13.2

Варианты заданий для создания объекта и его сериализации в XML-документ

Вариант	Данные для создания объекта и его сериализации в XML	Данные-атрибуты XML
1,13	<b>Студент.</b> ФИО, группа, пол, дата рождения, средний бал.	Группа
2,14	<b>Автомобиль.</b> Фирма, модель, год выпуска, цена, расход топлива.	Фирма
3,15	<b>Сотрудник.</b> ФИО, пол, дата рождения, должность, зарплата.	Должность
4,16	<b>Книга.</b> Название, автор, цена, число страниц, год издания.	Цена

Продолжение таблицы 13.2

5,17	<b>Учебная дисциплина.</b> Название, ФИО преподавателя, форма контроля, семестр, число часов.	Семестр
6,18	<b>Банковский вклад.</b> Номер счета, ФИО вкладчика, дата вклада, сумма, процент.	Номер счета
7,19	<b>Предмет обуви.</b> Наименование, производитель, число пар, размер, цена.	Производитель
8,20	<b>Телевизор.</b> Фирма, модель, размер экрана, вес, цена.	Фирма
9,21	<b>Билет на междугородный транспорт.</b> Рейс, пункт назначения, время отправления, длительность, номер места.	Рейс
10,22	<b>Квартира.</b> Дом, номер, этаж, площадь, цена.	Дом
11,23	<b>Заказ на перевозку груза.</b> Номер, дата, адрес доставки, вес груза, стоимость перевозки.	Номер
12,24	<b>Спортсмен.</b> ФИО, вид спорта, дата рождения, пол, рост, вес.	Вид спорта

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каково главное назначение языка XML?
2. В чем заключаются основные отличия XML от HTML?
3. Что представляет собой объявление XML-документа?
4. Каковы требования к именам элементов XML?
5. Какие XML-документы называют «правильно построенными» и «действительными»?
6. Чем различаются элементы-контейнеры и пустые элементы XML?
7. Что называют корневым элементом в XML и сколько таких элементов может быть в документе?
8. Для чего предназначены секции CDATA и как они записываются в XML-документе?



9. Каково назначение инструкций по обработке и как они записываются в XML-документе?
10. Какие средства используются в MS .NET Framework для работы с XML?
11. Что называют объектной моделью документа (DOM)?
12. Для чего предназначены классы XmlDocument, XmlElement и XmlAttribute?
13. Как с помощью класса XmlDocument можно выполнить сохранение и загрузку XML-документа?
14. Каково назначение классов XmlReader и XmlWriter?
15. Что называют сериализацией и десериализацией объектов?
16. Что понимают под атрибутами и атрибутными классами языка C#?
17. Как с помощью класса XmlSerializer выполняется сериализация объектов в XML?
18. Какие атрибуты языка C# могут использоваться для управления сериализацией объектов в XML?

## ЛАБОРАТОРНАЯ РАБОТА 14. СОЗДАНИЕ ЗАПРОСОВ К ДОКУМЕНТУ XML ПРИ ПОМОЩИ ЯЗЫКА XPATH

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* приобрести умение адресовать различные части XML-документа с помощью выражений на языке XPath для выполнения запросов к документу.

Задачи лабораторной работы:

- освоить написание выражений на языке XPath;
- научиться создавать приложения на языке C#, использующие модель данных XPath (класс XPathNavigator).

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Общие сведения о языке XPath.

Для удобной работы с XML-документом необходимы средства, позволяющие точно адресовать ту или иную его часть: отдельный узел (элемент, атрибут, секцию CDATA и др.), множество узлов, какой-либо участок или множество участков документа. Та-кие средства предоставляет язык XPath, разработанный консорциумом W3C.

XPath (XML Path Language) – язык адресации частей XML-документа, позволяющий обнаруживать определенные узлы в дереве XML-документа и выполнять обработку XML-данных.

В настоящее время существуют следующие версии языка XPath, определяемые рекомендациями консорциума W3C:

- XPath 1.0 (предложен как рекомендация 16 ноября 1999 г.): определены средства для поиска определённых узлов на дереве XML-документа; обеспечивается поддержку базовых типов данных: набор узлов, строка, логический тип, числовой тип; заданы основные операции и функции для работы с XML-данными.
- XPath 2.0 (утвержден как рекомендация 23 января 2007 г.): включена поддержка атомарных значений и встроенных типов данных XSD-схем; введены дополнительные операции и функции; включена возможность объявлять переменные в выражениях XPath и др.
- XPath 3.0 (предложен как рекомендация 8 апреля 2014 г.): введена поддержка анонимных функций и динамический вы-

зов функций; определены специальные литералы для пространств имён; добавлены новые операторы работы со строками.

Одной из задач языка XPath является работа с преобразованиями XSLT, которые позволяют преобразовывать XML-документы к другим формам (например, к HTML), а также работа с запросами на языке XQuery.

В языке XPath используется синтаксис, отличный от синтаксиса, принятого в XML.

Основу языка XPath составляют выражения различных типов (логические, числовые, строковые), в которых записываются константы, переменные и функции. Выражения связываются операциями, характерными для данного типа. В результате вычисления выражения получается указание на определенный участок документа.

Очень часто выражение строится подобно пути к файлу в файловой системе, откуда и происходит название языка «XML Path». При этом XML-документ рассматривается как дерево, корнем которого служит узел документа (рис. 14.1). Узлами служат элементы, их атрибуты и символьные данные.

Язык XPath различает несколько видов узлов в дереве XML-документа. К основным видам узлов относятся:

- Узел документа (document node). Данный узел не следует путать с узлом корневого элемента, который вложен в узел документа наряду с другими узлами.
- Узлы-элементы (element nodes). Имя узла-элемента состоит из префикса пространства имен и локального имени элемента, образующих вместе расширенное имя.
- Узлы-атрибуты (attributes nodes). Данные узлы принадлежат определенному узлу-элементу, но не считаются потомками этого узла. Узел-атрибут тоже определяется расширенным именем.
- Узлы пространств имен (namespace nodes). С каждым узлом-элементом связаны узлы тех пространств имен, в область действия которых входит данный элемент. Так же как и узлы-атрибуты, они не считаются потомками узла-элемента. Именем узла пространства имен является префикс.

- Текстовые узлы (text nodes), которые представляют строку символов, записанную между начальным и конечным тегами элемента.

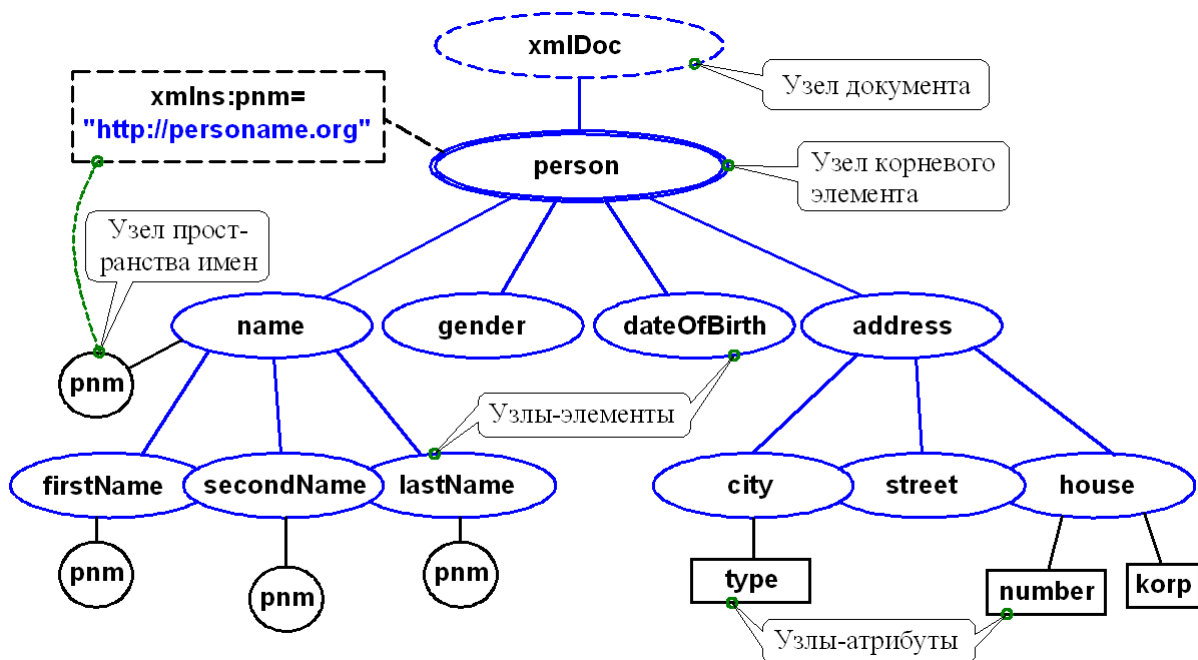


Рисунок 14.1 - Дерево XML-документа и основные типы узлов XPath

Узел любого вида можно представить строкой символов. Строка, представляющая узел документа или узел-элемент, состоит из всех строк, которые являются содержимым его узлов-потомков. Строка, представляющая узел-атрибут, содержит его значение. Строка, которая представляет узел пространства имен, содержит его строку URI.

**Контекст выражения. Шаги поиска. Специальные символы в XPath-выражении.**

Результат выражения XPath зависит от контекста, в котором задано это выражение. Главной составляющей контекста выражения является контекстный узел – узел, который обрабатывается в текущий момент.

Контекст XPath-выражения состоит из контекстного узла, а также из связанной с ним среды, включающей следующие компоненты:

- положение контекстного узла в дереве документа относительно одноуровневых узлов;

- размер контекста, то есть число одноуровневых элементов контекстного узла плюс один;
- объявления пространств имен в области выражения и др.

В версии XPath 2.0 для записи листьев дерева документа появились атомарные значения (atomic values). Эти значения относятся к одному из встроенных простых типов XSD (xs:string, xs:boolean, xs:decimal, xs:double и др.).

В результате вычисления каждого XPath-выражения получается адрес определенного участка документа в виде упорядоченной последовательности узлов и/или атомарных значений.

Чаще всего выражение языка XPath показывает путь к определенному участку документа. Такие выражения состоят из нескольких шагов поиска (location steps), выполняемых слева направо. Последовательность, полученная на предыдущем шаге, передается следующему шагу, который использует ее как исходный материал для поиска.

Например, следующее XPath-выражение состоит из трех шагов поиска:

`/institute/group/student`

Шаги поиска данного выражения содержат имена элементов `institute`, `group` и `student`. В результате вычисления первого шага этого выражения получится последовательность узлов, вложенных в узел документа `institute`. На втором шаге из этой последовательности выбираются узлы `group`, на третьем – узлы `student`.

В общем случае шаги поиска, состоят из трех частей:

- ось (axis), которая задает направление поиска (вверх или вниз по дереву документа);
- проверка узла (node test), которая выбирает в области поиска определенные узлы по имени и их типу;
- предикат (predicate), который содержит условия для проверки отобранных узлов и производит окончательный выбор.

Синтаксис шага поиска имеет следующий вид:

`ось::проверка_узла[предикат]*`

Примером шага поиска может быть следующее выражение:

`following::student[@id='s204718']`

В этом выражении производится выбор узла `student`, который является соседним к контекстному узлу (ось `following`) и имеет атрибут `id` со значением «`s204718`».

В выражениях XPath используются следующие специальные символы:

- «/» – задает дочерний элемент, имя которого указано слева; если этот символ стоит в начале выражения, то будут выбраны дочерние элементы корневого узла;
- «//» – задает рекурсивный спуск, то есть поиск заданного элемента на любой глубине; если этот оператор пути стоит в начале выражения, рекурсивный спуск будет вестись из корневого узла;
- «.» – указывает контекстный узел;
- «..» – указывает элемент-родитель для контекстного узла;
- «\*» – символ подстановки; выбирает все элементы независимо от их имени;
- «@» – указывает атрибут; является префиксом имени атрибута;
- «@\*» – символ подстановки для атрибута; выбирает все атрибуты независимо от имени;
- «:» – разделитель пространства имен; отделяет префикс пространства имени от собственно имени элемента или атрибута.

**Оси поиска. Дочерние оси и оси потомков. Родительские оси и оси предков.**

В языке XPath оси служат для того, чтобы выделить в дереве документа различные подмножества узлов относительно контекстного узла.

В дереве документа можно выделить два противоположных направления поиска:

- вниз – от узла документа по ветвям к вложенным узлам-элементам и листьям;
- вверх – от листьев или текущих узлов к узлу документа.

Можно отметить следующие оси поиска, которые задают поиск «вниз» и называются дочерними осями и осями потомков:

- `self` – сам контекстный узел;

- **child** – все дочерние узлы контекстного узла, кроме узлов-атрибутов и узлов пространств имен; эта ось является осью по умолчанию;
- **descendant** – все узлы-потомки контекстного узла, не включая узлы-атрибуты и узлы пространств имен;
- **descendant-or-self** – сам контекстный узел и все его потомки;
- **following** – все узлы, расположенные на дереве документа после контекстного узла; не включает потомков контекстного узла, узлы-атрибуты и узлы пространств имен;
- **attribute** – узлы-атрибуты контекстного узла-элемента; для других видов узлов эта ось пуста;
- **namespace** – узлы пространства имен контекстного узла-элемента; для других видов узлов эта ось пуста.

Следующие оси направляют поиск «вверх» и называются родительскими осями и осями предков:

- **parent** – родительский узел контекстного узла; располагается непосредственно «над» контекстным узлом;
- **ancestor** – все узлы-предки контекстного узла;
- **ancestor-or-self** – контекстный узел вместе со всеми своими предками;
- **preceding** – узлы, предшествующие контекстному узлу в документе; не содержит предков контекстного узла, узлы-атрибуты и узлы пространств имен.

На рис. 14.2 показаны области, определяемые различными осями относительно контекстного узла.

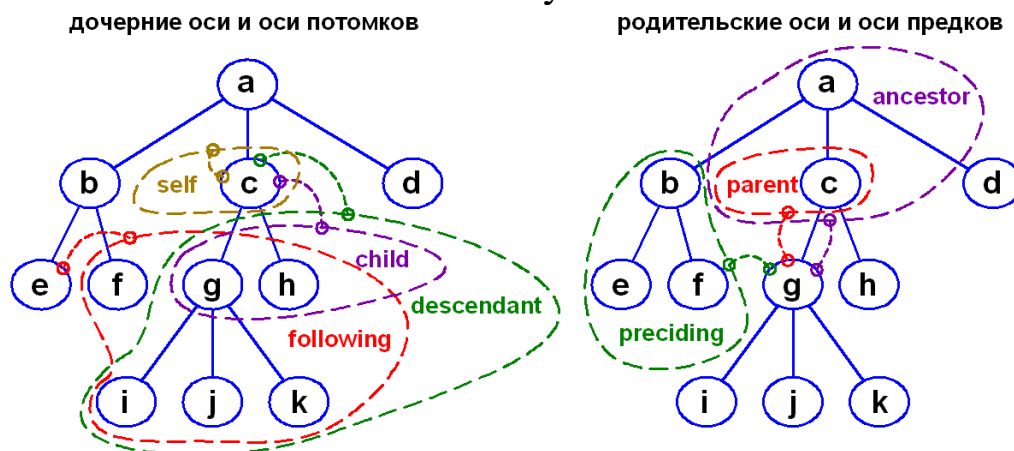


Рисунок 14.2 – Области на дереве XML-документа, определяемые осями

Проверки узла и предикаты.

После того как с помощью оси определена область поиска, в ней отыскиваются узлы определенных видов или узлы с определенными именами. Для этого применяется проверка узла. В соответствии со своими задачами тесты узла делятся на две группы:

- проверка по имени узла (`name test`), которая отбирает узлы по имени, записанному в проверке; содержит расширенное имя типа `QName`, состоящее из префикса и локального имени;
- проверка по виду узла (`kind test`), которая записывается в функциональной форме и может быть выбрана из нескольких вариантов (`node`, `text`, `comment` и др.).

Если при записи проверки по имени узла вместо префикса имени стоит звездочка, например `*:item`, то отбираются узлы с локальным именем `item` в любом пространстве имен.

Если вместо локального имени стоит звездочка, например, `tde:*`, то отбираются узлы с любыми локальными именами, объявленные в пространстве имен `tde`.

Можно выделить следующие разновидности проверки по виду узла:

- `node()` – отбирает узел любого вида;
- `text()` – отбирает текстовые узлы;
- `comment()` – отбирает узлы-комментарии;
- `element()` – отбирает все узлы-элементы; эквивалентно `element(*)` или `element(*, *)`;
- `element(имя_элемент)` – отбирает все узлы-элементы с именем `имя_элемент`;
- `element(имя_элемент, тип)` – отбирает все узлы-элементы с именем `имя_элемент` и типом `тип`, определенным в схеме XSD;
- `attribute()` – отбирает все узлы-атрибуты; эквивалентно `attribute(@*)` и `attribute(@*, *)`;
- `attribute(@имя_атрибут)` – отбирает все узлы-атрибуты с именем `имя_атрибут`;
- `attribute(@name, type)` – отбирает все узлы-атрибуты с именем `имя_атрибут` и типом `тип`;
- `processing-instruction()` – отбирает все инструкции по обработке.



После отбора, проведенного осью поиска и проверкой узла, из полученной последовательности узлов можно выделить те, которые удовлетворяют определенным условиям, задаваемым предикатом.

Предикат представляет собой логическое выражение (принимает значение true или false), записываемое в квадратных скобках.

Последовательность узлов, к которым применяется предикат, сортируется в направлении, указанном осью (от начала документа к его концу или наоборот). Отсортированные узлы нумеруются, начиная с 1, и просматриваются по очереди. Узлы, для которых значение предиката равно false, отбрасываются.

Номер текущего узла называется его позицией в отсортированной последовательности узлов. Точное значение позиции текущего узла возвращаются функции position().

Функция count() служит для нахождения числа узлов в последовательности. Количество узлов в последовательности также можно получить функцией last().

Операции и функции XPath.

Выражения XPath состояются из констант и вызовов функций, связанных знаками операций и скобками.

В выражениях XPath применяются два типа скобок:

- круглые скобки ( ) – группируют операции для явного задания порядка их выполнения;
- квадратные скобки [ ] – применяются для задания предиката.

Язык XPath поддерживает большое число операций (арифметические, логические, операции сравнения, операции со множествами и др.).

Основные виды арифметических операций XPath:

- «+» – выполняет сложение;
- «-» – выполняет вычитание;
- «\*» – Выполняет умножение;
- div – выполняет деление с плавающей запятой;
- mod – возвращает остаток от деления.

Основные логические операции XPath:

- and – логическое И;

- or – логическое ИЛИ;
- not() – отрицание (НЕ);

Операции сравнения XPath:

- = – равенство;
- != – не равно;
- < – меньше;
- <= – меньше или равно;
- > – больше;
- >= – больше или равно.

При использовании операций сравнения в XML-документе (например, в таблице стилей XSLT) символы < и > необходимо за-менять с помощью &lt; и &gt;. При использовании выражения XPath с моделью DOM (например, в коде на C#) символы < и > за-менять не требуется.

В язык XPath встроено множество функций, облегчающих поиск узлов и обработку XML-данных.

К числовым функциям XPath относятся:

- celing(x) – округление аргумента x до большего целого числа; например floor(3,5) равно 4, а floor(-3,5) равно -3;
- floor(x) – округление аргумента x до меньшего целого числа; например, floor(3,5) равно 3, а floor(-3,5) равно -4;
- round(x) – округление аргумента x до ближайшего целого числа; эквивалентна функции floor(x + 0,5);
- sum(expr) – возвращает сумму чисел, получаемых выражением expr.

Некоторые строковые функции XPath:

- concat(s1, s2, s3, ...) – объединяет строки s1, s2, s3, ... в од-ну строку;
- substring(s, n) – выделяет из строки s подстроку, начинающуюся с позиции n; номера символов начинаются с 1;
- contains(s, sub) – возвращает true, если строка s содержит подстроку sub;
- string-length(s) – возвращает число символов в строке s;
- upper-case(s) – переводит все буквы строки s в верхний регистр;

- `lower-case(s)` – переводит все буквы строки `s` в нижний регистр.

Функции, работающий с узлами XPath:

- `count(expr)` – возвращает число узлов в последовательности, задаваемой выражением `expr`;
- `position()` – возвращает порядковый номер (начинается с 1) текущего узла;
- `name()` – расширенное имя текущего узла (с префиксом пространства имен);
- `local-name()` – локальное имя текущего узла (без префикса пространства имен);
- `text()` – возвращает текстовое содержимое текущего узла;
- `namespace-uri()` – возвращает идентификатор пространства имён текущего узла;
- `root()` – возвращает узел документа, в котором расположен текущий узел.

К логическим функциям XPath относятся:

- `true()` – возвращает значение `true`;
- `false()` – возвращает значение `false`;
- `not()` – возвращает значение `true`, если значением аргумента является `false`, и наоборот.

### **Выбор узлов XML с помощью XPath-навигации.**

На платформе .NET классы для работы с XPath размещены в пространстве имен `System.Xml.XPath` (рис. 14.3). Основным классом данного пространства имен является `XPathNavigator`, который служит для перемещения по XML-документу с использованием языка XPath.

Класс `XPathNavigator` имеет следующие свойства:

- `Name` – возвращает полное имя текущего узла;
- `Value` – возвращает значение `string`;
- `NodeType` – возвращает тип узла (элемент, атрибут);

Методы:

- `Select()` – выбирает набор узлов с помощью заданного выражения XPath;
- `SelectSingleNode()` – выбирает первый узел, удовлетворяющий заданному выражению XPath;

- Evaluate() – возвращает значение, получаемое в результате использования функции XPath.

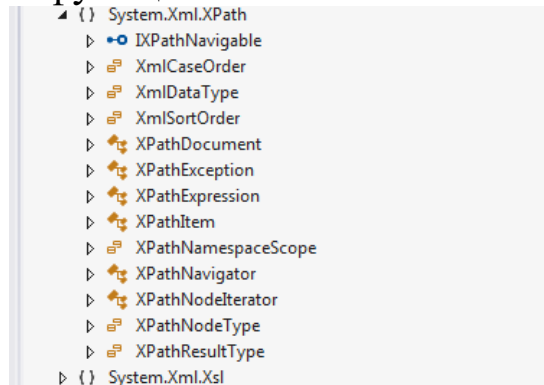


Рисунок 14.3 – Классы пространства имен System.Xml.XPath в окне обозревателя объектов

Для перебора коллекции узлов, полученных при обработке выражения XPath, используется класс XPathNodeIterator. Данный класс имеет свойство Current, которое получает объект XPathNavigator, соответствующий текущему контекстному узлу. Также класс XPathNodeIterator содержит метод MoveNext(), позволяющий перемещаться к следующему узлу в выбранном наборе узлов.

#### **Добавление и удаление узлов в XML-документе.**

Модель DOM XML-документа содержит методы, которые позволяют использовать язык XPath для поиска конкретного одиночного узла или всех узлов, соответствующих некоторым условиям.

Классы модели DOM предоставляют два метода выбора узлов с помощью XPath:

- XmlNode SelectSingleNode(string xPath) – возвращает первый узел XmlNode, соответствующий заданному выражению xPath;
- XmlNodeList SelectNodes(string xPath) – возвращает список узлов XmlNodeList, содержащий выбранные с помощью выражения xPath узлы.

Для удаления выбранного узла из модели DOM XML-документа используется метод RemoveChild(XmlNode xn). Этот метод удаляет все дочерние узлы, принадлежащие выбранному узлу. Для удаления только дочерних узлов выбранного узла можно использовать метод RemoveAll().

Удалить атрибуты заданного узла можно с помощью следующих методов:

- `RemoveAllAttributes()` – удаляет все атрибуты;
- `RemoveAttribute(string atrName)` – удаляет один атрибут из коллекции по заданному имени `atrName`;
- `RemoveAttributeAt(int atrIndex)` – удаляет из коллекции один атрибут по индексному номеру `atrIndex`.

## МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Составление выражений на языке XPath.

Требуется построить выражения на языке XPath для получения следующей информации из XML-документа:

1. ФИО всех жильцов, пользующихся коммунальными услугами.
2. Число всех обслуживаемых домов.
3. 3.Номера квартир в доме с заданным порядковым номером.
4. Число жильцов дома с указанным кодом.
5. ФИО жильцов проживающих в доме с заданным порядковым номером и в указанной квартире.
6. Суммарный расход электроэнергии в обслуживаемых домах.
7. Суммарная плата, внесенная жильцами дома с заданным кодом.
8. Номера обслуживаемых домов, расположенные на выбранной улице.
9. Квартиры с площадью больше, чем заданная площадь.
10. Номера квартир, в которых расход холодной воды или горячей воды больше заданного.
11. Число квартир с количеством жильцов не меньше заданного.
12. Число квартир с ненулевой пеней.
13. Среднее потребление электроэнергии на одну квартиру.
14. ФИО жильца по указанному номеру в документе.
15. Номер квартиры, в которой проживает жилец с указанным ФИО (использовать ось `parent`).

16. Описания домов, для которых получены показания приборов по выбранной дате (использовать ось *ancestor*).

17. Число квартир, которые следуют в документе за квартирой с указанным кодом (использовать ось *following*).

18. ФИО жильцов, которые предшествуют в документе жильцу с заданным кодом (использовать ось *preceding*).

XPath-выражения для выполнения заданных запросов будут иметь следующий вид:

```

1. коммун_услуги/дом/квартира/жилец/фио
2. count(коммун_услуги/дом)
3. коммун_услуги/дом[...]/квартира/@номер
4. count(коммун_услуги/дом[@код='...']/квартира/жилец)
5. //дом[...]/квартира[@номер='...']/жилец/фио
6. sum(//эл_энерг)
7. sum(//дом[@код='...']/*/плата/всего)
8. //дом[адрес/улица]/*/номер
9. //квартира[площадь > ...]/@номер
10. //квартира[показ_приборов/хол_вода > ... or
    показ_приборов/гор_вода > ...]/@номер
11. count(//квартира[count(жилец) >= ...])
12. count(//квартира[плата/пеня != 0])
13. sum(//квартира/показ_приборов/эл_энерг) div
count(//квартира)
14. descendant::жилец[3]/фио
15. //жилец[фио='...']/parent::квартира/@номер
16. //показ_приборов[@дата='...']/ancestor::дом/описание
17. count(//квартира[@код='...']/following::квартира)
18. //жилец[@код='...']/preceding::жилец/фио

```

**Для обработки полученных выражений XPath требуется разработать консольное приложение на языке C#.**

Обработка XPath-выражений с помощью консольного приложения на языке C#.

Требуется разработать консольное приложение на языке C# для обработки выражений XPath, с использованием класса XPathNavigator.

В классе Program консольного приложения добавим статический метод XPathProcess, который обрабатывает XPath-выражения и выводит на консоль результаты выполнения запросов.

Исходный код полученного консольного приложения представлен в листингах 14.1.

Результат выполнения приложения в окне консоли показан на рис. 14.4.

Листинг 14.1. Исходный код класса Program консольного приложения

```

6 using System.Xml.XPath; // Импорт пространства имён для работы с моделью данных XPath
7
8 namespace XPathExprs
9 {
10     class Program
11     {
12         /// <summary>
13         /// Обрабатывает выражение XPath с выводом результатов на консоль
14         /// </summary>
15         /// <param name="xpnav">Экземпляр XPathNavigator</param>
16         /// <param name="exprs">XPath-выражение</param>
17         /// <param name="title">Заголовок</param>
18         static void XPathProcess(XPathNavigator xpnav, string exprs, string title)
19         {
20             double result = 0.0;
21             // Признак того, является ли результат XPath-выражения числом
22             bool isNumber = double.TryParse(xpnav.Evaluate(exprs).ToString(), out result);
23             if (isNumber == false)
24             {
25                 Console.WriteLine(title);
26                 // Создание итератора для перебора узлов
27                 XPathNodeIterator xpnIter = xpnav.Select(exprs);
28                 // Объект XPathNavigator для работы с текущим узлом
29                 XPathNavigator navCurNode = xpnIter.Current;
30                 while (xpnIter.MoveNext())
31                 {
32                     Console.WriteLine("- {0}", navCurNode.Value);
33                 }
34             }
35             else Console.WriteLine(title + " {0}", result);
36         }
37
38         /// <summary>
39         /// Выводит ФИО всех жильцов, пользующихся коммунальными услугами
40         /// </summary>
41         /// <param name="xpNav">Объект XPathNavigator</param>
42         static void Query1(XPathNavigator xpNav)
43         {
44             string title = "1. ФИО всех жильцов, пользующихся коммунальными услугами:";
45             string exprs = "коммун_услуги/дом/квартира/жильец/фio";
46             XPathProcess(xpNav, exprs, title);
47         }
48
49         /// <summary>
50         /// Выводит число обслуживаемых домов
51         /// </summary>
52         /// <param name="xpNav">Объект XPathNavigator</param>
53         static void Query2(XPathNavigator xpNav)
54         {
55             string title = "2. Число обслуживаемых домов:";
56             string exprs = "count(коммун_услуги/дом)";
57             XPathProcess(xpNav, exprs, title);
58         }
59     }
60 }

```

```

60  /// <summary>
61  /// Номера квартир в доме с заданным порядковым номером
62  /// </summary>
63  /// <param name="xpNav">Объект XPathNavigator</param>
64  /// <param name="houseNum">Порядковый номер дома</param>
65  static void Query3(XPathNavigator xpNav, int houseNum)
66  {
67      string title = string.Format("3. Все квартиры дома под номером {0}:", houseNum);
68      string exprs = string.Format("коммун_услуги/дом[{0}]/квартира/@номер", houseNum);
69      XPathProcess(xpNav, exprs, title);
70  }
71
72  /// <summary>
73  /// Выводит число жильцов дома с указанным кодом
74  /// </summary>
75  /// <param name="xpNav">Объект XPathNavigator</param>
76  /// <param name="houseId">Код дома</param>
77  static void Query4(XPathNavigator xpNav, string houseId)
78  {
79      string title = string.Format("4. Число жильцов дома с кодом {0}:", houseId);
80      string exprs = string.Format("count(коммун_услуги/дом[@код='{0}']" +
81          "/квартира/желец)", houseId);
82      XPathProcess(xpNav, exprs, title);
83  }
84
85  /// <summary>
86  /// Выводит ФИО жильцов проживающих в доме с заданным порядковым номером
87  /// и в указанной квартире
88  /// </summary>
89  /// <param name="xpNav">Объект XPathNavigator</param>
90  /// <param name="houseNum">Порядковый номер дома в документе</param>
91  /// <param name="apart"></param>
92  static void Query5(XPathNavigator xpNav, int houseNum, int apart)
93  {
94      string title = string.Format("5. ФИО жильцов, проживающих в доме " +
95          "с порядковым номером {0} и квартире №{1}:", houseNum, apart);
96      string exprs = string.Format("//дом[{0}]/квартира[@номер='{1}']/желец/фio",
97          houseNum, apart);
98      XPathProcess(xpNav, exprs, title);
99  }
100
101  /// <summary>
102  /// Выводит суммарный расход электроэнергии в обслуживаемых домах
103  /// </summary>
104  /// <param name="xpNav">Объект XPathNavigator</param>
105  static void Query6(XPathNavigator xpNav)
106  {
107      string title = "6. Суммарный расход электроэнергии в обслуж. домах, квт:";
108      string exprs = "sum(//эл_энерг)";
109      XPathProcess(xpNav, exprs, title);
110  }

```



```

112 /// <summary>
113 /// Выводит суммарную плату, внесенную жильцами дома с указанным кодом
114 /// </summary>
115 /// <param name="xpNav">Объект XPathNavigator</param>
116 /// <param name="houseId">Код дома</param>
117 static void Query7(XPathNavigator xpNav, string houseId)
118 {
119     string title = string.Format("7. Суммарная плата, внесенная жильцами дома " +
120     "с кодом {0}, руб.:", houseId);
121     string exprs = string.Format("sum(//дом[@код='{0}']/*/плата/всего)", houseId);
122     XPathProcess(xpNav, exprs, title);
123 }
124
125 /// <summary>
126 /// Выводит номера обслуживаемых домов на заданной улице
127 /// </summary>
128 /// <param name="xpNav">Объект XPathNavigator</param>
129 /// <param name="street">Название улицы</param>
130 static void Query8(XPathNavigator xpNav, string street)
131 {
132     string title = string.Format("8. Номера обслуживаемых домов по улице {0}:",
133     street);
134     string exprs = string.Format("//дом[адрес/улица='{0}']/*/номер", street);
135     XPathProcess(xpNav, exprs, title);
136 }
137
138 /// <summary>
139 /// Выводит коды квартир с площадью больше заданного
140 /// </summary>
141 /// <param name="xpNav">Объект XPathNavigator</param>
142 /// <param name="square">Площадь, м2</param>
143 static void Query9(XPathNavigator xpNav, int square)
144 {
145     string title = string.Format("9. Коды квартир с площадью более {0} м2:",
146     square);
147     string exprs = string.Format("//квартира[площадь>{0}]/@код", square);
148     XPathProcess(xpNav, exprs, title);
149 }
150
151 /// <summary>
152 /// Выводит номера квартир, в которых расход холодной воды или
153 /// горячей воды больше заданного
154 /// </summary>
155 /// <param name="xpNav">Объект XPathNavigator</param>
156 /// <param name="coldWater">Расход холодной воды</param>
157 /// <param name="hotWater">Расход горячей воды</param>
158 static void Query10(XPathNavigator xpNav, double coldWater, double hotWater)
159 {
160     string title = string.Format("10. Квартиры, в которых расход " +
161     "холодной воды >{0} или горячей воды >{1}:", coldWater, hotWater);
162     string exprs = string.Format("//квартира[показ_приборов/хол_вода>{0} or " +
163     "показ_приборов/гор_вода>{1}]/@номер", coldWater, hotWater);
164     XPathProcess(xpNav, exprs, title);
165 }

```

```

167 /// <summary>
168 /// Выводит число квартир с количеством жильцов не меньше заданного
169 /// </summary>
170 /// <param name="xpNav">Объект XPathNavigator</param>
171 /// <param name="lodgerAmount">Количество жильцов в квартире</param>
172 static void Query11(XPathNavigator xpNav, int lodgerAmount)
173 {
174     string title = string.Format("11. Число квартир с количеством" +
175     " жильцов >= {0}:", lodgerAmount);
176     string exprs = string.Format("count(//квартира[count(желец) >= {0}])",
177     lodgerAmount);
178     XPathProcess(xpNav, exprs, title);
179 }
180
181 /// <summary>
182 /// Выводит число квартир с ненулевой пеней
183 /// </summary>
184 /// <param name="xpNav">Объект XPathNavigator</param>
185 static void Query12(XPathNavigator xpNav)
186 {
187     string title = "12. Число квартир с ненулевой пеней:";
188     string exprs = "count(//квартира[плата/пеня != 0])";
189     XPathProcess(xpNav, exprs, title);
190 }
191
192 /// <summary>
193 /// Выводит среднее потребление электроэнергии одной квартирой
194 /// </summary>
195 /// <param name="xpNav">Объект XPathNavigator</param>
196 static void Query13(XPathNavigator xpNav)
197 {
198     string title = "13. Среднее потребление электроэнергии одной квартирой, квт:";
199     string exprs = "sum(//эл_энерг) div count(//квартира)";
200     XPathProcess(xpNav, exprs, title);
201 }
202
203 /// <summary>
204 /// Выводит ФИО жильца по указанному номеру в документе
205 /// </summary>
206 /// <param name="xpNav">Объект XPathNavigator</param>
207 /// <param name="lodgerNum">Порядковый номер жильца</param>
208 static void Query14(XPathNavigator xpNav, int lodgerNum)
209 {
210     string title = string.Format("14. ФИО жильца под номером {0} (ось parent):",
211     lodgerNum);
212     string exprs = string.Format("descendant::желец[{0}]/фio", lodgerNum);
213     XPathProcess(xpNav, exprs, title);
214 }

```

```

216 /// <summary>
217 /// Выводит номер квартиры, в которой проживает жилец с указанным ФИО
218 /// </summary>
219 /// <param name="xpNav">Объект XPathNavigator</param>
220 /// <param name="lodgerName">ФИО жильца</param>
221 static void Query15(XPathNavigator xpNav, string lodgerName)
222 {
223     string title = string.Format("15. Номер квартиры, в которой проживает {0} " +
224         "(ось parent):", lodgerName);
225     string exprs = string.Format("//жилец[фio='{0}']/parent::квартира/@номер",
226         lodgerName);
227     XPathProcess(xpNav, exprs, title);
228 }
229
230 /// <summary>
231 /// Выводит описания домов, для которых получены показания приборов
232 /// по выбранной дате
233 /// </summary>
234 /// <param name="xpNav">Объект XPathNavigator</param>
235 /// <param name="date">Дата получения показаний приборов</param>
236 static void Query16(XPathNavigator xpNav, string date)
237 {
238     string title = string.Format("16. Описания домов, в которых " +
239         "показания приборов относятся к дате {0} (ось ancestor):", date);
240     string exprs = string.Format("//показ_приборов[@дата='{0}']/ancestor::дом" +
241         "/описание", date);
242     XPathProcess(xpNav, exprs, title);
243 }
244
245 /// <summary>
246 /// Выводит число квартир, которые следуют в документе за квартирой с
247 /// указанным кодом
248 /// </summary>
249 /// <param name="xpNav">Объект XPathNavigator</param>
250 /// <param name="apartId">Код квартиры</param>
251 static void Query17(XPathNavigator xpNav, string apartId)
252 {
253     string title = string.Format("17. Число квартир, которые следуют в документе" +
254         " за квартирой с кодом {0} (ось following):", apartId);
255     string exprs = string.Format("count(//квартира[@код='{0}']/following::квартира)",
256         apartId);
257     XPathProcess(xpNav, exprs, title);
258 }
259
260 /// <summary>
261 /// Выводит ФИО жильцов, которые предшествуют в документе жильцу
262 /// с заданным кодом
263 /// </summary>
264 /// <param name="xpNav"></param>
265 /// <param name="lodgerId">Код жильца</param>
266 static void Query18(XPathNavigator xpNav, string lodgerId)
267 {
268     string title = string.Format("18. ФИО жильцов, которые предшествуют " +
269         " в документе жильцу с кодом {0} (ось preceding):", lodgerId);
270     string exprs = string.Format("//жилец[@код='{0}']/preceding::жилец/фio",
271         lodgerId);
272     XPathProcess(xpNav, exprs, title);
273 }

```

```

275 static void Main(string[] args)
276 {
277     Console.Title = "Создание запросов к XML-документу с помощью XPath-выражений";
278     // URI, задающий путь к XML-документу
279     string uri = @"I:\XmlDocs\CommService.xml";
280     // Объект XPathDocument для чтения XML-документа
281     XPathDocument xpDoc = new XPathDocument(uri);
282     // Объект XPathNavigator для обработки XPath-выражений
283     XPathNavigator xpNav = xpDoc.CreateNavigator();
284     // Выполнение запросов к XML-документу
285     Query1(xpNav);
286     Query2(xpNav);
287     Query3(xpNav, 1);
288     Query4(xpNav, "h18");
289     Query5(xpNav, 1, 59);
290     Query6(xpNav);
291     Query7(xpNav, "h18");
292     Query8(xpNav, "Волгоградская");
293     Query9(xpNav, 30);
294     Query10(xpNav, 150.0, 100.0);
295     Query11(xpNav, 2);
296     Query12(xpNav);
297     Query13(xpNav);
298     Query14(xpNav, 3);
299     Query15(xpNav, "Курганков Георгий Михайлович");
300     Query16(xpNav, "2015-08-05");
301     Query17(xpNav, "a236");
302     Query18(xpNav, "c27789");
303
304     Console.Read();
305 }
306

```

Результаты выполнения запросов к XML-документу, выведенные в окне консоли, показаны на рис. 14.4.

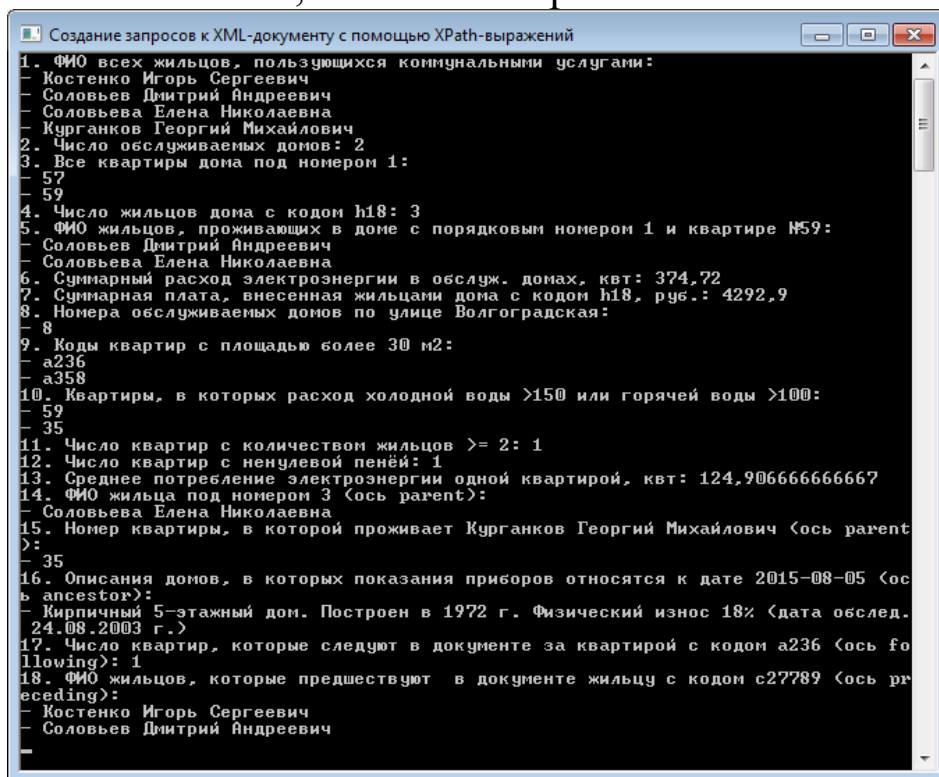


Рисунок 14.4. Результат выполнения консольного приложения

Модификация XML-документа с помощью приложения Windows Forms на языке C#.

Требуется разработать приложение Windows Forms, которое позволяет модифицировать XML-документ, добавляя в него новые узлы и удаляя выбранные узлы. Кроме того, приложение должно выводить данные по выбранному узлу XML-документа. Выбор узлов будет производиться с помощью выражений на языке XPath.

В XML-документе будут размещаться данные о продуктах питания (код, название, производитель, количество, вес, цена). Начальный код XML-документа представлен в листинге 14.3.

Создадим решение ModifXmlDoc с проектом приложения на основе экранной формы WindowsFormsApplication1.

Приложение должно выводить отдельные продукты в виде пунктов элемента управления listView1 (список). При выделении пунктов этого списка данные о конкретном продукте должны выводиться в текстовые поля textBoxId (код продукта), textBoxNum (количество), textBoxName (название) и др. Данные для создания новых элементов считываются из этих же текстовых полей.

Добавление новых элементов в XML-документ и соответствующих им пунктов в список listView1 будет производиться при нажатии кнопки buttonAdd. Удаление элементов должно выполняться с помощью кнопки buttonRemove. Код модифицируемого XML-документа при работе приложения будет выводиться в текстовое поле textBoxXML.

Интерфейс пользователя приложения Windows Forms, созданный в конструкторе форм, показан на рис.14.5.

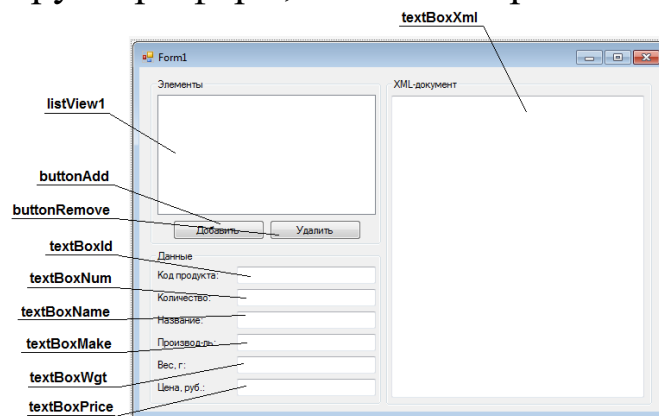


Рисунок 14.5. Интерфейс пользователя приложения WindowsFormsApplication1

## Листинг 14.3. Код XML-документа products.xml

```

<?xml version="1.0" encoding="utf-8"?>
<productStock>
  <product id="p1777" numberInStock="42">
    <name>Крупа гречневая</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">72,50</price>
  </product>
  <product id="p9794" numberInStock="67">
    <name>Кукуруза консервированная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">340</weight>
    <price unit="руб.">56</price>
  </product>
  <product id="p2079" numberInStock="53">
    <name>Крупа рисовая</name>
    <make>ЗАО Увелка</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">42,50</price>
  </product>
  <product id="p5681" numberInStock="30">
    <name>Говядина тушеная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">330</weight>
    <price unit="руб.">95</price>
  </product>
  <product id="p7678" numberInStock="68">
    <name>Макароны</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">500</weight>
    <price unit="руб.">56,35</price>
  </product>
</productStock>

```

Для обеспечения работы с XML-документом добавим в приложение класс XmlDataProvider, который будет размещён в библиотеке классов ClassLibrary1. В классе будут содержаться методы, которые возвращают код XML-документа и требуемые XML-узлы, а также позволяют добавить заданный узел в документ XML и удалить его из документа.

Исходный код класса XmlDataProvider представлен в листинге 14.4.

## Листинг 14.4. Исходный код класса XmlDataProvider

```

6  using System.Xml;
7
8  namespace ClassLibrary1
9  {
10     /// <summary>
11     /// Обеспечивает работу с XML-документом
12     /// </summary>
13     public class XmlDataProvider
14     {
15         XmlDocument doc; // XML-документ
16         string url;       // Указатель ресурса
17
18         public XmlDataProvider()
19         {
20             doc = new XmlDocument();
21             url = @"products.xml";
22         }
23
24         /// <summary>
25         /// Возвращает строку с кодом XML-документа
26         /// </summary>
27         /// <returns>Код XML-документа</returns>
28         public string GetXmlCode()
29         {
30             doc.Load(url);
31             return (doc.InnerXml);
32         }
33
34         /// <summary>
35         /// Возвращает список узлов product XML-документа
36         /// </summary>
37         /// <returns>Список узлов product XML-документа</returns>
38         public XmlNodeList GetXmlNodes()
39         {
40             doc.Load(url);
41             XmlNodeList xnl = doc.SelectNodes("productStock/product");
42             return xnl;
43         }
44     }
45 }

```



```

45  /// <summary>
46  /// Возвращает произвольный узел product XML-документа
47  /// </summary>
48  /// <returns>XML-узел product</returns>
49  public XmlNode GetXmlNode()
50  {
51      // Построение дерева для XML-элемента
52      XmlElement prodElem = doc.CreateElement("product");
53      prodElem.SetAttribute("id", "pXXXX");
54      prodElem.SetAttribute("numberInStock", "0");
55      XmlElement nameElem = doc.CreateElement("name");
56      nameElem.InnerText = "Название";
57      prodElem.AppendChild(nameElem);
58      XmlElement makeElem = doc.CreateElement("make");
59      makeElem.InnerText = "Производитель";
60      prodElem.AppendChild(makeElem);
61      XmlElement wgtElem = doc.CreateElement("weight");
62      wgtElem.SetAttribute("unit", "г.");
63      wgtElem.InnerText = "0";
64      prodElem.AppendChild(wgtElem);
65      XmlElement priceElem = doc.CreateElement("price");
66      priceElem.SetAttribute("unit", "руб.");
67      priceElem.InnerText = "0";
68      prodElem.AppendChild(priceElem);
69      return (prodElem);
70  }
71
72  /// <summary>
73  /// Удаляет заданный узел из XML-документа
74  /// </summary>
75  /// <param name="xn">Удаляемый XML-узел</param>
76  public void RemoveXmlNode(XmlNode xn)
77  {
78      string id = xn.SelectSingleNode("@id").InnerText;
79      XmlNodeList xnl = doc.SelectNodes("productStock/product");
80      foreach (XmlNode x in xnl)
81      {
82          if (x.SelectSingleNode("@id").InnerText == id)
83          {
84              doc.DocumentElement.RemoveChild(x);
85              break;
86          }
87      }
88      doc.Save(url);
89  }
90
91  /// <summary>
92  /// Добавляет узел в XML-документ
93  /// </summary>
94  /// <param name="xn">Добавляемый XML-узел</param>
95  public void AddXmlNode(XmlNode xn)
96  {
97      // Выбор корневого элемента XML-документа doc и добавление
98      // в него нового узла xn
99      doc.DocumentElement.AppendChild(xn);
100     doc.Save(url);
101 }
102 }

```

В классе **Form1** приложения **WindowsFormsApplication1** добавим вложенный класс **MyListViewItem**, который создан на основе класса **ListViewItem**, представляющего элемент списка **ListView**. В данном классе будет присутствовать автоматически



реализуемое свойство **XmlNode**, служащее для хранения заданного узла XML-документа.

#### Листинг 14.5. Исходный код класса **Form1**

```

10 using System.Xml;
11 using ClassLibrary1;
12
13 namespace WindowsFormsApplication1
14 {
15     public partial class Form1 : Form
16     {
17         XmlDataProvider xdp;
18
19         /// <summary>
20         /// Представляет элемент списка ListView (вложенный класс)
21         /// </summary>
22         class MyListViewItem : ListViewItem
23         {
24             // Узел XML-документа, связанный с элементом списка
25             public XmlNode XmlNode { get; set; }
26
27             public MyListViewItem(XmlNode xn)
28             {
29                 this.XmlNode = xn;
30                 this.Text = string.Format("- {0} - {1}",
31                     xn.SelectSingleNode("@id").InnerText,
32                     xn.SelectSingleNode("name").InnerText);
33             }
34         }
35
36         // Конструктор класса Form1
37         public Form1()
38         {
39             InitializeComponent();
40             xdp = new XmlDataProvider();
41         }
42
43         // Обработчик события "Загрузка формы Form1"
44         private void Form1_Load(object sender, EventArgs e)
45         {
46             this.Text = "Модификация XML-документа (Турчин Д.Е., каф. ИиАПС)";
47             this.Font = new Font("Arial", 10, FontStyle.Regular);
48             listView1.View = View.List;
49             textBoxXml.ScrollBars = ScrollBars.Vertical;
50             textBoxXml.Text = xdp.GetXmlCode().ToString();
51
52             MyListViewItem myLvI;
53             // Заполнение списка listView1
54             foreach (XmlNode xn in xdp.GetXmlNodes())
55             {
56                 myLvI = new MyListViewItem(xn);
57                 listView1.Items.Add(myLvI);
58             }
59         }

```

```

61 // Обработчик события "Выбор нового элемента списка listView1"
62 private void listView1_SelectedIndexChanged(object sender, EventArgs e)
63 {
64     // Коллекция выделенных элементов списка ListView
65     ListView.SelectedListViewItemCollection sLviC = listView1.SelectedItems;
66
67     // Выбор первого элемента коллекции sLviC и запись его данных в текстовые поля
68     foreach (MyListViewItem myLvi in sLviC)
69     {
70         textBoxId.Text = myLvi.XmlNode.SelectSingleNode("@id").InnerText;
71         textBoxNum.Text = myLvi.XmlNode.SelectSingleNode("@numberInStock").InnerText;
72         textBoxName.Text = myLvi.XmlNode.SelectSingleNode("name").InnerText;
73         textBoxMake.Text = myLvi.XmlNode.SelectSingleNode("make").InnerText;
74         textBoxWgt.Text = myLvi.XmlNode.SelectSingleNode("weight").InnerText;
75         textBoxPrice.Text = myLvi.XmlNode.SelectSingleNode("price").InnerText;
76         break;
77     }
78 }
79
80 // Обработчик события "Нажатие на кнопку buttonAdd"
81 private void buttonAdd_Click(object sender, EventArgs e)
82 {
83     XmlNode xn = xdp.GetXmlNode();
84
85     // Считывание данных из текстовых полей и их запись в XML-узел xn
86     xn.SelectSingleNode("@id").InnerText = textBoxId.Text;
87     xn.SelectSingleNode("@numberInStock").InnerText = textBoxNum.Text;
88     xn.SelectSingleNode("name").InnerText = textBoxName.Text;
89     xn.SelectSingleNode("make").InnerText = textBoxMake.Text;
90     xn.SelectSingleNode("weight").InnerText = textBoxWgt.Text;
91     xn.SelectSingleNode("price").InnerText = textBoxPrice.Text;
92
93     MyListViewItem myLvi = new MyListViewItem(xn);
94
95     xdp.AddXmlNode(myLvi.XmlNode);
96     listView1.Items.Add(myLvi);
97
98     textBoxXml.Text = xdp.GetXmlCode();
99 }
100
101 // Обработчик события "Нажатие на кнопку buttonRemove"
102 private void buttonRemove_Click(object sender, EventArgs e)
103 {
104     // Коллекция выделенных элементов списка ListView
105     ListView.SelectedListViewItemCollection sLviC = listView1.SelectedItems;
106
107     foreach (MyListViewItem myLvi in sLviC)
108     {
109         xdp.RemoveXmlNode(myLvi.XmlNode);
110         myLvi.Remove();
111     }
112     textBoxXml.Text = xdp.GetXmlCode();
113 }
114 }

```

Результат работы полученного приложения показан на рисунке 14.6.

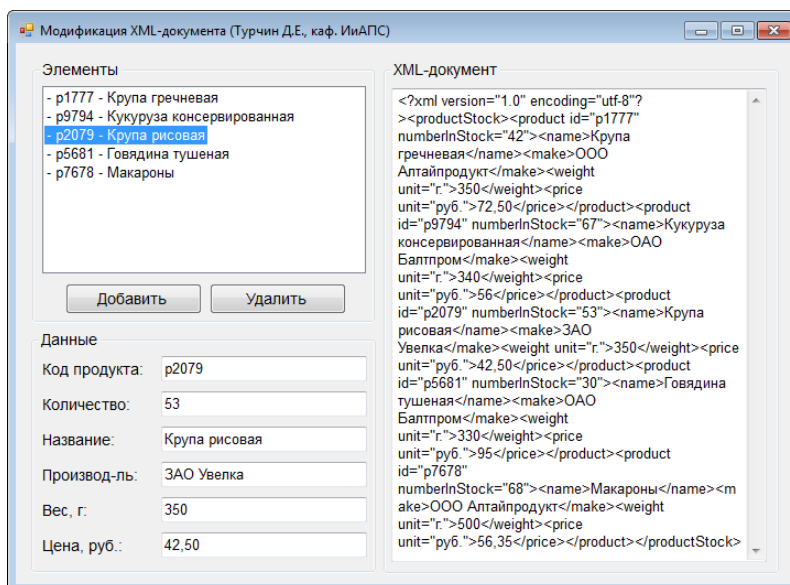


Рисунок 14.6. Работа приложения WindowsFormsApplication

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

### Основные этапы выполнения работы.

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. На основе XML-документа, полученного в работе №1, и предложенного задания (табл. 14.1) разработать необходимые XPath-выражения. При написании выражения №9 требуется использовать оси parent или ancestor, а для выражения №10 – оси preceding или following.
3. Создать консольное приложение на языке C# для обработки полученных XPath-выражений с помощью модели данных XPath и выполнения параметрических запросов к документу XML.
4. Разработать приложение Windows Forms, которое позволяет модифицировать XML-документ. Приложение должно добавлять заданные элементы (табл. 14.2) в XML-документ, а также удалять их из документа.
5. Оформить и защитить отчет по лабораторной работе.

### Индивидуальные варианты заданий.

Таблица 14.1

Варианты заданий для разработки XPath-выражений

Вариант	Данные для построения запросов к XML-документу
1,13	<p><b>Расписание рейсов междугородных автобусов.</b></p> <ol style="list-style-type: none"> <li>1. Все пункты отправления.</li> <li>2. Общее число пунктов прибытия.</li> <li>3. Все рейсы из пункта отправления «...».</li> <li>4. Число рейсов с ценой билета &gt; ... руб.</li> <li>5. Рейсы, осуществляемые ежедневно в пункт прибытия «...»</li> <li>6. Средняя стоимость билета.</li> <li>7. Пункты прибытия с числом рейсов &gt; ...</li> <li>8. Число пунктов отправления, с &gt; «...» числом пунктов прибытия.</li> <li>9. Пункт отправления, соответствующий рейсу с заданным кодом.</li> <li>10. Число пунктов прибытия, предшествующих в документе пункту прибытия с указанным названием.</li> </ol>
2,14	<p><b>Товарный склад.</b></p> <ol style="list-style-type: none"> <li>1. Перечень всех наименований товаров на складе.</li> <li>2. Общее число всех стеллажей.</li> <li>3. Все товары фирмы «Название».</li> <li>4. Число наименований товаров с весом более «Вес».</li> <li>5. Все товары с «Дата поступления» или сроком менее «Срок хран.».</li> <li>6. Общая стоимость всех товаров на складе.</li> <li>7. Фирмы, у которых на складе число единиц товаров более «Число».</li> <li>8. Число стеллажей, на которых суммарный вес товаров более «...».</li> <li>9. Коды стеллажей, на которых расположены товары выбранного производителя.</li> <li>10. Число товаров, следующих в документе за товаром с указанным кодом.</li> </ol>

Продолжение таблицы 14.1

3,15	<p><b>Ресторан.</b></p> <ol style="list-style-type: none"> <li>1. Все блюда в меню ресторана.</li> <li>2. Число всех сделанных заказов.</li> <li>3. Все ингредиенты блюда «Код».</li> <li>4. Число всех блюд в заказе «Код».</li> <li>5. Заказанные блюда, цена которых более «Цена».</li> <li>6. Суммарный объем ингредиента «Название» в заказанных блюдах.</li> <li>7. Заказы с количеством блюд 2 и более.</li> <li>8. Число заказов, сделанных «Дата» суммарной ценой более «Цена».</li> <li>9. Коды заказов, в которых был использован ингредиент с указанным названием.</li> <li>10. Число блюд, предшествующих в документе блюду с заданным кодом.</li> </ol>
4,16	<p><b>Расписание занятий.</b></p> <ol style="list-style-type: none"> <li>1. Все учебные дисциплины.</li> <li>2. Общее число занятий на четной и нечетной неделях.</li> <li>3. Все аудитории, в которых проводятся занятия в «День недели».</li> <li>4. Число занятий, которые ведут преподаватели с «Должность».</li> <li>5. Дни недели, по которым занятия в ауд. «Номер» первой парой.</li> <li>6. Среднее число занятий в один день.</li> <li>7. Дни недели, по которым проводится более «Число» занятий.</li> <li>8. Число дней, в которые количество лекций от «Число» до «Число».</li> <li>9. Дни недели, по которым занятия проводит преподаватель с заданным ФИО.</li> <li>10. Число занятий, следующих в документе за занятием с указанным кодом.</li> </ol>

Продолжение таблицы 14.1

5,17	<b>Книжный магазин.</b> <ol style="list-style-type: none"> <li>1. Все книги магазина.</li> <li>2. Число всех отделов.</li> <li>3. Все книги отдела «Название».</li> <li>4. Число всех журналов отдела «Название».</li> <li>5. Книги автора «Автор» с числом страниц более «Число страниц».</li> <li>6. Суммарная цена всех книг.</li> <li>7. Отделы, в которых число журналов менее «Число».</li> <li>8. Число книг с ценой от «...» до «...».</li> <li>9. Название отдела, в котором есть книга с указанным кодом.</li> <li>10. Число журналов, предшествующих в документе журналу с заданным названием.</li> </ol>
6,18	<b>Кинотеатр.</b> <ol style="list-style-type: none"> <li>1. Все фильмы, демонстрируемые в кинотеатре.</li> <li>2. Число залов в кинотеатре.</li> <li>3. Все актеры, играющие главные роли в фильме «Название».</li> <li>4. Число сеансов на фильм «...».</li> <li>5. Сеансы, проведенные «Дата» с ценой билета более «Цена».</li> <li>6. Средняя цена билета на сеансы фильма «Название».</li> <li>7. Фильмы с участием актера «...» или снятые режиссером «...».</li> <li>8. Число фильмов, на которые сеансов от «...» до «...».</li> <li>9. Код зала, в котором проводится показ фильма с указанным названием.</li> <li>10. Число сеансов, следующих в документе за сеансом с заданным кодом.</li> </ol>

Продолжение таблицы 14.1

7,19	<p><b>Проектно-строительная компания.</b></p> <ol style="list-style-type: none"> <li>1. Все сотрудники компании.</li> <li>2. Число всех отделов.</li> <li>3. Все проекты, выполняемые в отделе «Название».</li> <li>4. Число сотрудников в отделе «Название».</li> <li>5. Все сотрудники женского пола, имеющие оклад более «Оклад».</li> <li>6. Суммарная стоимость проектов в отделе «Название».</li> <li>7. Отделы, в которых число сотрудников муж. пола менее «Число».</li> <li>8. Число сотрудников, оклад которых составляет от «...» до «...».</li> <li>9. Название отдела, в котором работает сотрудник с заданным ФИО.</li> <li>10. Число проектов, предшествующих в документе проекту с указанным кодом.</li> </ol>
8,20	<p><b>Обувной магазин.</b></p> <ol style="list-style-type: none"> <li>1. Все отделы обувного магазина.</li> <li>2. Число всех единиц обуви.</li> <li>3. Все аксессуары отдела «Название».</li> <li>4. Число единиц обуви в отделе «Название».</li> <li>5. Вся обувь фирмы «Производитель» стоимостью более «Цена».</li> <li>6. Суммарная стоимость товаров в магазине.</li> <li>7. Отделы, в которых скидка на обувь составляет более «Скидка».</li> <li>8. Число единиц обуви с размером от «...» до «...».</li> <li>9. Название отдела, в котором присутствует обувь с заданным названием.</li> <li>10. Число аксессуаров, следующих в документе за аксессуаром с указанным кодом.</li> </ol>

Продолжение таблицы 14.1

9,21	<p><b>Перевозка грузов.</b></p> <ol style="list-style-type: none"> <li>1. Все водители автотранспортного предприятия.</li> <li>2. Число всех автомобилей на предприятии.</li> <li>3. Все водители, работающие на автомобиле «Код».</li> <li>4. Число автомобилей, задействованных в заказе «Код».</li> <li>5. Все автомобили марки «Марка» и типом кузова «Тип кузова».</li> <li>6. Средний вес перевезенного груза на один автомобиль.</li> <li>7. Заказы, в которых участвовали два и более водителя.</li> <li>8. Число заказов, в которых вес груза составил от «...» до «...».</li> <li>9. ФИО водителя, который выполнял заказ с указанным кодом.</li> <li>10. Число заказов, предшествующих в документе заказу с заданным кодом.</li> </ol>
10,22	<p><b>Поликлиника.</b></p> <ol style="list-style-type: none"> <li>1. Все врачи, работающие в поликлинике.</li> <li>2. Число всех пациентов, приписанных к поликлинике.</li> <li>3. Все пациенты, наблюдаемые врачом «ФИО».</li> <li>4. Число обращений, сделанных пациентом «ФИО».</li> <li>5. Все обращения, сделанные «Дата» с диагнозом «Болезнь».</li> <li>6. Средняя продолжительность болезни «Болезнь».</li> <li>7. Пациенты, которые сделали более «...» обращений.</li> <li>8. Число обращений с продолжительностью болезни от «...» до «...».</li> <li>9. ФИО врача, к которому относится обращение с заданным кодом.</li> <li>10. Число пациентов, следующих в документе за пациентом с указанным кодом.</li> </ol>



Продолжение таблицы 14.1

11,23	<p><b>Банковские услуги.</b></p> <ol style="list-style-type: none"> <li>1. Все кредиты, предоставленные банком.</li> <li>2. Число всех клиентов банка.</li> <li>3. Все вклады, сделанные клиентом «ФИО».</li> <li>4. Число кредитов, по которым процентная ставка менее «Ставка».</li> <li>5. Кредиты, взятые «Дата» на сумму более «Сумма».</li> <li>6. Средний размер вклада.</li> <li>7. Клиенты, взявшие более одного кредита.</li> <li>8. Число вкладов, размер которых составляет от «...» до «...».</li> <li>9. ФИО клиента, которому принадлежит вклад с указанным кодом.</li> <li>10. Число кредитов, предшествующих в документе кредиту с заданным кодом.</li> </ol>
12,24	<p><b>Прогноз погоды.</b></p> <ol style="list-style-type: none"> <li>1. Все записи об осадках.</li> <li>2. Число дней, по которым имеется прогноз погоды.</li> <li>3. Все данные по влажности во время суток «Тип».</li> <li>4. Число дней, в которые направление ветра будет «Направление».</li> <li>5. Все даты, когда во время суток «Тип» температура менее «...».</li> <li>6. Средняя скорость ветра для «Дата».</li> <li>7. Дни недели с прогнозом более, чем для «...» времён суток.</li> <li>8. Число дней, в которых давление составляет от «...» до «...».</li> <li>9. Дни недели, в которых была облачность указанного типа.</li> <li>10. Число времён суток, которое следуют за временем суток с указанным кодом.</li> </ol>

Таблица 14.2 Варианты заданий для модификации XML-документа

Вариант	Данные для добавления и удаления XML-элементов
1,13	<b>Квартира.</b> Адрес дома, номер, этаж, площадь, цена.
2,14	<b>Заказ на перевозку груза.</b> Номер, дата, адрес доставки, вес груза, стоимость перевозки.
3,15	<b>Спортсмен.</b> ФИО, вид спорта, дата рождения, пол, рост, вес.
4,16	<b>Студент.</b> ФИО, группа, пол, дата рождения, средний бал.
5,17	<b>Автомобиль.</b> Фирма, модель, год выпуска, цена, расход топлива.
6,18	<b>Сотрудник.</b> ФИО, пол, дата рождения, должность, зарплата.
7,19	<b>Книга.</b> Название, автор, цена, число страниц, год издания.
8,20	<b>Учебная дисциплина.</b> Название, ФИО преподавателя, форма контроля, семестр, число часов.
9,21	<b>Банковский вклад.</b> Номер счета, ФИО вкладчика, дата вклада, сумма, процент.
10,22	<b>Предмет обуви.</b> Наименование, производитель, число пар, размер, цена.
11,23	<b>Телевизор.</b> Фирма, модель, размер экрана, вес, цена.
12,24	<b>Билет на междугородный транспорт.</b> Рейс, пункт назнач., время отправления, длительность, номер места.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего предназначен язык XPath?
2. Что представляют собой XPath-выражения?
3. Какие основные узлы выделяют в дереве документа XML при использовании XPath?
4. Что понимают под контекстным узлом в XPath и что относят к контексту XPath-выражения?
5. Из каких частей в общем случае состоит шаг поиска в выражении XPath?
6. Что называют осью поиска, и на какие группы разделяют оси?
7. Что такое проверка узла и какие выделяют виды проверок в XPath?
8. Что называют предикатом в выражении XPath?
9. Какие выделяют логические операции и операции сравнения в XPath?
10. Для чего предназначен класс XPathNavigator?
11. Как в модели DOM осуществляется выбор узлов XML-документа с помощью XPath-выражений?
12. Каким образом можно удалить выбранные узлы XML-документа в модели DOM?

## ЛАБОРАТОРНАЯ РАБОТА 15. ОСНОВЫ СОЗДАНИЯ ЗАПРОСОВ К КОЛЛЕКЦИЯМ ОБЪЕКТОВ С ПОМОЩЬЮ LINQ

### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель лабораторной работы:* приобрести умение выполнять запросы к источникам данных в форме коллекций объектов с помощью технологии LINQ to Ob-jects.

Задачи лабораторной работы:

- ознакомиться с технологией LINQ;
- научиться создавать запросы LINQ к коллекциям объектов;
- освоить использование лямбда-выражений в запросах LINQ.

### ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

#### **Общие сведения о технологии LINQ. Запросы LINQ.**

Одной из наиболее важных технологий для доступа к данным на платформе .NET является LINQ (Language Integrated Query) – язык интегрированных запросов.

Под LINQ понимают набор средств, появившийся в .NET Framework 3.5, который предоставляет стандартные технологии для работы с различными типами источников данных. Для запросов и преобразований данных в LINQ используются одинаковые базовые шаблоны кодирования, напоминающие SQL.

В состав Visual Studio, начиная с версии 2008, входят средства для использования LINQ с коллекциями объектов, поддерживающих интерфейс IEnumerable, базами данных SQL Server, наборами данных ADO.NET и XML-файлами.

По типу источника данных выделяют следующие разновидности LINQ:

1. LINQ to Objects – позволяет применять запросы к коллекциям объектов;
2. LINQ to DataSet – позволяет применять запросы LINQ к объектам DataSet из ADO.NET;
3. LINQ to Entities – позволяет применять запросы LINQ внутри API-интерфейса ADO.NET Entity Framework (EF);

4. LINQ to XML – позволяет применять запросы LINQ к документам XML и манипулировать XML-данными;

5. Parallel LINQ (PLINQ) – позволяет выполнять параллельную обработку данных, возвращенных запросом LINQ.

Для применения средств LINQ to Objects в исходный код программы следует импортировать пространство имен System.Linq. Для этого в файле C# должна присутствовать следующая директива:

```
using System.Linq;
```

В основу LINQ положено понятие запроса, в котором определяется сведения, получаемые из источника данных. При необходимости, запрос также указывает способ сортировки и группировки этих сведений. В запросе LINQ работа всегда осуществляется с объектами.

Для сохранения результатов запросов LINQ удобно использовать неявно типизируемые локальные переменные, задаваемые ключевым словом var. Тип данных таких переменных определяется во время компиляции на основе первоначального значения.

Неявная типизация применима только для локальных переменных в контексте какого-то метода или свойства. Не допускается применять ключевое слово var для определения полей классов, возвращаемых значений и параметров методов.

Кроме того, локальным переменным, объявленным с помощью ключевого слова var, обязательно должно быть присвоено начальное значение в самом объявлении.

Все операции запроса LINQ состоят из трех основных действий:

1. Получение доступа к источнику данных. При этом в источнике должен быть реализован интерфейс IEnumerable.

2. Создание запроса, которое заключается в определении того, что именно следует извлечь из источника данных.

3. Выполнение запроса, в ходе которого выводятся результаты. Это может быть сделано в цикле foreach.

В общем виде запрос LINQ записывается следующим образом:

```
var переменная_запроса = выражение_запроса;
```

Запрос хранится в переменной запроса и инициализируется выражением запроса. Сама переменная запроса только хранит команды запроса. Фактическое выполнение запроса откладывается до выполнения итерации переменной запроса в операторе `foreach`. Выражение запроса состоит из набора предложений, включающих операции запросов LINQ и операнды.

### Основные операции запросов LINQ.

В языке C# определены различные операции запросов LINQ. Наиболее часто используемыми операциями являются:

- `from ... in` – используется для определения основы любого выражения, позволяющей извлечь подмножество данных из нужного источника;
- `select` – используется для выбора последовательности из источника данных;
- `where` – используется для определения ограничений на извлекаемые из источника данные;
- `orderby` – позволяет упорядочить результирующий набор;
- `group` – позволяет группировать данные по указанному ключу.

В простейшем виде каждый запрос LINQ строится из операций `from`, `in` и `select`, которые вместе с операндами образуют соответствующие предложения. Синтаксис простейшего запроса имеет вид:

```
var перем_запроса = from перем_диапаз in источник_данных
                    select результат_запроса;
```

Переменная диапазона, следующая за операцией `from`, принимает данные из источника данных, записанного после операции `in`. Тип переменной диапазона выводится из источника данных. Имя переменной диапазона задается в самом выражении запроса.

Запрос может содержать несколько предложений `from`, что требуется при получении данных из нескольких источников.

Операция `select` определяет, что именно должно быть получено по запросу (результат запроса). Когда предложение с операцией `select` создает что-либо отличное от копии переменной диапазона, то операция `select` называется проекцией. Использование

проекций для преобразования данных является мощной возможностью выражений запросов LINQ.

Часто требуется отсортировать возвращенные запросом данные по одному или нескольким критериям. Для этого используется операция `orderby`. Запрос LINQ при задании сортировки будет иметь следующий вид:

```
var перемен_запроса = from перемен_диапаз in источник_данных
    orderby выраж_сортировки [descending]
    select результат_запроса;
```

Выражение сортировки содержит элемент, по которому проводится сортировка. По умолчанию принята сортировка по возрастанию, поэтому упорядочивание строк производится в алфавитном порядке, числовых значений – от меньшего к большему, и т.д.

Операция `descending` используется для сортировки в обратном порядке (по убыванию). По умолчанию используется операция `ascending`, которую можно не записывать при сортировке по возрастанию.

Чтобы получить определенное подмножество из источника можно использовать операцию `where`. При этом запрос будет иметь следующий синтаксис:

```
var перемен_запроса = from перемен_диапаз in источник_данных
    where логич_выражение
    select результат_запроса;
```

Логическое выражение, задаваемое после операции `where`, задает определенное условие и возвращает булевское значение. Запрос возвращает только те элементы, для которых логическое выражение является истинным.

Для построения выражения фильтра в предложении `where` можно использовать логические операции C# AND (`&&`) и OR (`||`).

Операция `group` служит для группирования полученных данных по ключам. Данная операция, как и операция `select`, может завершать выражение запроса. Запрос с операцией `group` в простейшем случае может быть записан следующим образом:

```
var перемен_запроса = from перемен_диапаз in источник_данных
    group результат_запроса by ключ;
```

Когда запрос завершается предложением `group`, его результаты представляют группу списков. При итерации таких результатов запроса, необходимо использовать вложенный цикл `foreach`. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл – итерацию элементов каждой группы.

Для результата выполнения операции `group` определено доступное только для чтения свойство `Key`, которое возвращает ключ.

Операция `join` служит для объединения двух последовательностей данных в одну. При использовании операции `join` каждый источник должен содержать данные, которые можно сравнивать. Сравниваемые элементы данных указываются после ключевого слова `on`. При этом операция `join` отбирает только те элементы данных, которые имеют общее значение. Запрос LINQ при использовании операции `join` имеет следующий синтаксис:

```
var переменная_запроса = from переменная_диапазона_A in источник_данных_A
                        join переменная_диапазона_B in источник_данных_B
                        on переменная_диапазона_A.свойство equals переменная_диапазона_B.свойство
                        select результат_запроса;
```

### **Анонимные типы и расширяющие методы.**

В языке C# одним из средств, непосредственно связанных с LINQ, являются анонимные типы (англ. anonymous types). Как следует из названия, анонимный тип представляет собой класс, не имеющий имени. Его основное назначение состоит в создании объекта, возвращаемого оператором `select`.

Анонимный тип генерируется компилятором на основе инициализации создаваемого объекта.

Благодаря анонимным типам в ряде случаев отпадает необходимость объявлять класс, который предназначен только для хранения результата запроса.

Анонимный тип объявляется с помощью следующей общей формы:

```
new { имя_A = значение_A, имя_B = значение_B, ... }
```

Расширяющий метод представляет собой статический метод, который может быть связан с классом так, что он может быть вызван, как метод экземпляра этого класса. Расширяющие



методы дают возможность использовать метод с классом, который не предоставляет его изначально.

Для того чтобы поменять порядок элементов в результирующем наборе на обратный используется расширяющий метод `Reverse<T>()` класса `Enumerable`.

Для вычисления одного значения из коллекции значений используются статистические операции. К основным статистическим операциям LINQ относятся:

- `Average` – вычисляет среднее арифметическое значение в коллекции;
- `Count` – подсчитывает число элементов в коллекции (при необходимости только те элементы, которые удовлетворяют заданному условию);
- `Max` – определяет максимальное значение в коллекции;
- `Min` – определяет минимальное значение в коллекции;
- `Sum` – вычисляет сумму значений в коллекции.

### **Понятие и виды анонимных функций C#. Особенности за-писи лямбда-выражений.**

Для упрощения работы с делегатами в C# используются анонимные функции.

Анонимная функция – это оператор или выражение, которое можно использовать каждый раз, когда ожидается тип делегата.

В языке C# существует два типа анонимных функций:

- Анонимные методы. Начиная с C# 2.0. Применение анонимных методов является громоздким и трудно читаемым.
- Лямбда-выражения. Начиная с C# 3.0.

Лямбда-выражение (англ. *lambda-expression*) – это анонимная функция, с помощью которой можно создавать типы делегатов или деревьев выражений.

С помощью лямбда-выражений можно написать локальные функции, которые затем можно передавать в другие функции в качестве аргументов или возвращать из них в качестве значения. Лямбда-выражения особенно полезны при написании запросов LINQ.

Лямбда-выражение в C# имеет следующий синтаксис:

`(параметр1, параметр2, ... , параметрN) => выражение`

В начале лямбда-выражения записывается список входных параметров (если таковые имеются), разделённых запятыми. Если параметров более одного, то параметры заключаются в круглые скобки. За списком параметров следует лямбда-оператор, обозначаемый через `=>`. После лямбда-оператора записывается выражение или блок операторов.

Простейшее лямбда-выражение может иметь следующий вид:

```
x => x
```

Данное лямбда-выражение может быть прочитано, как «х идёт к х». Это означает, что при входном параметре `x` выражение вернёт `x`, то есть выражение просто возвращает, то, что оно получило.

Следующее лямбда-выражение возвращает длину строкового входного параметра `s`:

```
s => s.Length
```

Соответствующий этому лямбда-выражению делегат должен принимать единственный параметр типа `string` и иметь тип возврата `int`.

Если лямбда-выражению передаётся несколько параметров, то их отделяют запятыми и помещают в скобки. Например, лямбда-выражение, возвращающее результат сравнения (тип `bool`) двух параметров `a` и `b`, будет иметь следующий вид:

```
(a, b) => a == b
```

Сложные лямбда-выражения могут включать в себя блок операторов. Например, следующее лямбда-выражение возвращает большее из значений двух параметров `x` и `y`:

```
(x, y) =>
{
    if (x > y) return (x);
    else return (y);
}
```

Деревья выражений являются эффективным представлением в форме дерева лямбда-выражений в операциях запросов.

## Использование лямбда-выражений в запросах LINQ.

Лямбда-выражения могут быть переданы в качестве параметров в методы расширения LINQ-запросов.

Метод расширения Where используется для фильтрации элементов в последовательность и имеет следующий синтаксис:

```
public static IEnumerable<T> Where<T>(
    this IEnumerable<T> source,
    Func<T, bool> predicate);
```

Метод расширения Select.

Операция GroupBy используется для группировки элементов входной последовательности по заданному ключу и имеет следующий синтаксис:

```
public static IEnumerable<IGrouping<K, T>> GroupBy<T, K>(
    IEnumerable<T> source,
    Func<T, K> keySelector);
```

Интерфейс IGrouping<K, T> представляет последовательность типа T с ключом типа K и определяется следующим образом:

```
public interface IGrouping<K, T> : IEnumerable<T>
{
    K Key { get; }
}
```

Метод расширения Join выполняет соединение двух последовательностей по эквивалентности двух ключей, извлечённых из элементов этих последовательностей. Данный метод имеет следующий синтаксис:

```
public static IEnumerable<V> Join<T, U, K, V>(
    IEnumerable<T> outer,
    IEnumerable<U> inner,
    Func<T, K> outerKeySelector,
    Func<U, K> innerKeySelector,
    Func<T, U, V> resultSelector);
```

Операции разбиения (partitioning) позволяют вернуть выходную последовательность, которая является подмножеством входной последовательности. К основным операциям разбиения относятся Take, TakeWhile, Skip, SkipWhile.

Операция конкатенации `Concat` позволяет объединить несколько однотипных входных последовательностей в одну выходную.

## **МЕТОДИКА И ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

**Выполнение запросов LINQ к списку объектов в консольном приложении на языке C#.**

Требуется разработать набор запросов LINQ к массиву объектов, каждый из которых содержат данные о продуктивном товаре (наименование, производитель, количество единиц, вес одной единицы, цена, код стеллажа), хранимом на складе. Группы товаров размещаются на стеллажах (код, число ярусов, максимальная грузонесущая способность).

Исходный код на языке C# для класса `Product`, описывающего продуктивный товар, представлен в листинге 15.1, а код класса `Store` (стеллаж) – в листинге 15.2.

Листинг 15.1. Исходный код класса `Product`

```

9  |  /// <summary>
10 |  /// Представляет продуктовые товары
11 |  /// </summary>
12 |  public class Product
13 |  {
14 |      /// <summary>
15 |      /// Наименование
16 |      /// </summary>
17 |      public string Name { get; set; }
18 |
19 |      /// <summary>
20 |      /// Производитель
21 |      /// </summary>
22 |      public string ProducedBy { get; set; }
23 |
24 |      /// <summary>
25 |      /// Число единиц
26 |      /// </summary>
27 |      public int NumberInStok { get; set; }
28 |
29 |      /// <summary>
30 |      /// Вес, г
31 |      /// </summary>
32 |      public int Weight { get; set; }
33 |
34 |      /// <summary>
35 |      /// Цена, руб
36 |      /// </summary>
37 |      public int Price { get; set; }
38 |
39 |      /// <summary>
40 |      /// Код стеллажа
41 |      /// </summary>
42 |      public string StoreID { get; set; }
43 |
44 |      /// <summary>
45 |      /// Возвращает строку с данными о продукте
46 |      /// </summary>
47 |      /// <returns>Строка с данными о продукте</returns>
48 |      public override string ToString()
49 |      {
50 |          return string.Format("Наименование: {0}\n" +
51 |                                "- Производитель: {1}\n" +
52 |                                "- Количество: {2} шт\n" +
53 |                                "- Вес: {3} г\n" +
54 |                                "- Цена: {4} руб\n" +
55 |                                "- Код стеллажа: {5}", Name, ProducedBy, NumberInStok,
56 |                                Weight, Price, StoreID);
57 |      }
58 |  }

```

Листинг 15.2. Исходный код класса Store

```

9  /// <summary>
10  /// Представляет стеллажи
11  /// </summary>
12  public class Store
13  {
14      /// <summary>
15      /// Код стеллажа
16      /// </summary>
17      public string StoreID { get; set; }
18
19      /// <summary>
20      /// Число ярусов
21      /// </summary>
22      public int NumberOfFlour { get; set; }
23
24      /// <summary>
25      /// Макс. грузонесущая способность, кг
26      /// </summary>
27      public int MaxWeight { get; set; }
28  }

```

Требуется разработать следующие запросы:

1. Данные о всех товарах.
2. Наименования всех товаров в алфавитном порядке.
3. Товары с количеством более 50 шт.
4. Товары фирмы «Алтайпродукт» с ценой менее 80 руб.
5. Число наименований товаров весом от 250 до 500 г.
6. Наименования товаров и их количество в порядке убывания количества.
7. Средняя, наибольшая и наименьшая цены товаров фирмы «Алтайпродукт».
8. Суммарный вес всех товаров на складе.
9. Общая стоимость товаров каждого наименования.
10. Наименования товаров, сгруппированные по производителям (используется операция group).
11. Наименование и количество товара с указанием данных о стеллаже, на котором он хранится (используется операция join).

Данные о товарах организуем в виде массива `itemsInStock`, состоящего из объектов класса `Product`. Массив `itemsInStock` будет объявлен в методе `Main()` (листинг 15.3) консольного приложения. Также в методе `Main()` производится вызов методов, выполняющих запросы LINQ.

Листинг 15.3. Исходный код метода Main консольного приложения

```

10 class Program
11 {
12     static void Main(string[] args)
13     {
14         Console.Title = "Выполнение запросов LINQ к массиву объектов";
15
16         // Коллекция объектов класса Product с инициализацией элементов
17         List<Product> products = new List<Product> {
18             new Product { Name = "Крупа гречневая",
19                           ProducedBy = "ООО Алтайпродукт",
20                           NumberInStok = 42,
21                           Weight = 350,
22                           Price = 72,
23                           StoreID = "024" },
24             new Product { Name = "Кукуруза консервированная",
25                           ProducedBy = "ОАО Балтпром",
26                           NumberInStok = 67,
27                           Weight = 340,
28                           Price = 56,
29                           StoreID = "017" },
30             new Product { Name = "Крупа рисовая",
31                           ProducedBy = "ЗАО Увелка",
32                           NumberInStok = 53,
33                           Weight = 350,
34                           Price = 42,
35                           StoreID = "024" },
36             new Product { Name = "Говядина тушеная",
37                           ProducedBy = "ОАО Балтпром",
38                           NumberInStok = 30,
39                           Weight = 330,
40                           Price = 96,
41                           StoreID = "017" },
42             new Product { Name = "Макароны",
43                           ProducedBy = "ООО Алтайпродукт",
44                           NumberInStok = 48,
45                           Weight = 500,
46                           Price = 56,
47                           StoreID = "028" }};
48
49         // Коллекция объектов класса Store
50         List<Store> stores = new List<Store> {
51             new Store { StoreID = "024",
52                           NumberOfFlour = 4,
53                           MaxWeight = 10000 },
54             new Store { StoreID = "028",
55                           NumberOfFlour = 4,
56                           MaxWeight = 10000 },
57             new Store { StoreID = "017",
58                           NumberOfFlour = 6,
59                           MaxWeight = 12000 }};

```

Исходный код методов, которые содержат выражения запроса LINQ и выполняют их обработку, представлен в листинге 15.4.

# Листинг 15.4. Исходный код методов, выполняющих запросы LINQ

```

61         Console.WriteLine("***** Результаты запросов LINQ *****");
62         // Вызовы методов выполнения запросов LINQ
63         GetAllProds(products);
64         GetAllNames(products);
65         GetProdOver(products, 25);
66         GetAltPrNam(products, "ООО Алтайпродукт", 80);
67         GetNumProds(products, 250, 500);
68         GetNameNumb(products);
69         GetAvgPrice(products, "ООО Алтайпродукт");
70         GetSumWeigt(products);
71         GetSumPrice(products);
72         GetNameMake(products);
73         GetProdBalt(products, stores, "ОАО Балтпром", "ЗАО Увелка");
74         Console.ReadLine();
75     }
76
77     /// Получить все данные о товарах
78     /// </summary>
79     /// <param name="products">Список продуктов</param>
80     static void GetAllProds(List<Product> products)
81     {
82         Console.WriteLine("\n1. Все данные о товарах на складе:");
83         var all = from p in products select p;
84         foreach (var a in all) Console.WriteLine(a.ToString());
85     }
86
87     /// <summary>
88     /// Получить наименования всех товаров в алфавитном порядке
89     /// </summary>
90     /// <param name="products">Список продуктов</param>
91     static void GetAllNames(List<Product> products)
92     {
93         Console.WriteLine("\n2. Все наименования товаров на складе (по алфавиту):");
94         var names = from p in products orderby p.Name select p.Name;
95         foreach (var n in names) Console.WriteLine("- " + n);
96     }
97
98     /// <summary>
99     /// Получить данные о товарах с количеством больше заданного
100    /// </summary>
101    /// <param name="products">Список продуктов</param>
102    /// <param name="amount">Число единиц</param>
103    static void GetProdOver(List<Product> products, int amount)
104    {
105        Console.WriteLine("\n3. Все товары с количеством более {0}:", amount);
106        var overStock = from p in products where p.NumberInStok > amount select p;
107        foreach (var os in overStock) Console.WriteLine(os.ToString());
108    }

```



```

110 /// <summary>
111 /// Получить наименования товаров выбранного производителя с ценой
112 /// меньше заданной
113 /// </summary>
114 /// <param name="products">Список продуктов</param>
115 /// <param name="make">Производитель</param>
116 /// <param name="price">Цена, руб</param>
117 static void GetAltPrNam(List<Product> products, string make, int price)
118 {
119     Console.WriteLine("\n4. Все товары фирмы {0} с ценой менее {1} руб.:",
120         make, price);
121     var altNames = from p in products
122                     where p.ProducedBy == make && p.Price < price
123                     select p.Name;
124     foreach (var an in altNames) Console.WriteLine("- {0}", an);
125 }
126
127 /// <summary>
128 /// Найти число наименований товаров с весом в указанном диапазоне
129 /// </summary>
130 /// <param name="products">Список продуктов</param>
131 /// <param name="minW">Наибольший вес, г</param>
132 /// <param name="maxW">Наименьший вес, г</param>
133 static void GetNumProds(List<Product> products, int minW, int maxW)
134 {
135     Console.WriteLine("\n5. Число наименований товаров весом от {0} до {1} г:",
136         minW, maxW);
137     var nameProd = from p in products
138                     where p.Weight > minW && p.Weight < maxW
139                     select p.Name;
140     Console.WriteLine("Всего {0} наименов.", nameProd.Count());
141 }
142
143 /// <summary>
144 /// Получить наименования и количество продуктов (по убыванию)
145 /// </summary>
146 /// <param name="products">Список продуктов</param>
147 static void GetNameNumb(List<Product> products)
148 {
149     Console.WriteLine("\n6. Наименование и количество товара (по убыванию):");
150     var nameNumb = from p in products
151                     orderby p.NumberInStok descending
152                     select p;
153     foreach (var nn in nameNumb)
154     { Console.WriteLine("- {0}, {1} шт.", nn.Name, nn.NumberInStok); }
155 }

```

```

157 /// <summary>
158 /// Найти наибольшую, наименьшую и среднюю цены товаров выбранной фирмы
159 /// </summary>
160 /// <param name="products">Список продуктов</param>
161 /// <param name="make">Производитель</param>
162 static void GetAvgPrice(List<Product> products, string make)
163 {
164     Console.WriteLine("\n7. Средняя, наибольшая и наименьшая цены товаров фирмы {0}",
165         make);
166     var prAltPrice = from p in products
167                     where p.ProducedBy == make
168                     select p.Price;
169     Console.WriteLine("- средняя цена: {0:f2} руб.\n" +
170         "- наибольшая цена: {1} руб.\n" +
171         "- наименьшая цена: {2} руб.",
172         prAltPrice.Average(), prAltPrice.Max(), prAltPrice.Min());
173 }
174
175 /// <summary>
176 /// Найти суммарный вес товаров на складе
177 /// </summary>
178 /// <param name="products">Список продуктов</param>
179 static void GetSumWeigt(List<Product> products)
180 {
181     Console.WriteLine("\n8. Суммарный вес всех товаров:");
182     var all = from p in products select p;
183     double sumWeight = 0;
184     foreach (var a in all) sumWeight += a.NumberInStok * a.Weight;
185     Console.WriteLine("Всего {0:f2} кг.", sumWeight / 1000);
186 }
187
188 /// <summary>
189 /// Найти общую стоимость товаров каждого наименования
190 /// </summary>
191 /// <param name="products">Список продуктов</param>
192 static void GetSumPrice(List<Product> products)
193 {
194     Console.WriteLine("\n9. Общая стоимость товаров каждого наименования:");
195     var all = from p in products orderby p.Name select p;
196     foreach (var a in all)
197     {
198         Console.WriteLine("- {0}, {1} руб.", a.Name,
199             a.NumberInStok * a.Price);
200     }
201 }

```

```

203 /// <summary>
204 /// Получить наименования товаров, сгруппированные по производителям
205 /// </summary>
206 /// <param name="products">Список продуктов</param>
207 static void GetNameMake(List<Product> products)
208 {
209     Console.WriteLine("\n10. Наименования товаров, сгруппир. по производителям:");
210     var groups = from p in products
211                  group p by p.ProducedBy;
212     // Цикл для выбора каждой группы group из списка групп groups
213     foreach (var group in groups)
214     {
215         Console.WriteLine("Товары фирмы {0}:", group.Key);
216         // Цикл для выбора каждого элемента elem группы group
217         foreach (var elem in group)
218         { Console.WriteLine("- " + elem.Name); }
219     }
220 }
221
222
223 /// <summary>
224 /// Получить наименования и количество товаров выбранных фирм с указанием
225 /// данных по стеллажам, на которых они хранятся (исп. join)
226 /// </summary>
227 /// <param name="products">Список продуктов</param>
228 /// <param name="stores">Список стеллажей</param>
229 /// <param name="make1">Производитель 1</param>
230 /// <param name="make2">Производитель 2</param>
231 static void GetProdBalt(List<Product> products, List<Store> stores,
232 string make1, string make2)
233 {
234     Console.WriteLine("\n11. Наименования и количество товаров фирм {0}\n" +
235 "или {1} с указанием данных по стеллажам, на которых они хранятся",
236 make1, make2);
237     var result = from p in products
238                  where p.ProducedBy == make1 || p.ProducedBy == make2
239                  join s in stores on p.StoreID equals s.StoreID
240                  select new
241                  {
242                      name = p.Name,
243                      numb = p.NumberInStok,
244                      stID = p.StoreID,
245                      numF = s.NumberOfFlour,
246                      maxW = s.MaxWeight
247                  };
248     Console.WriteLine("|{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
249 "Наименование", "Количество", "Код стел.", "Число ярусов",
250 "Макс. груз, кг");
251     foreach (var r in result)
252     {
253         Console.WriteLine("|{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
254 r.name, r.numb, r.stID, r.numF, r.maxW);
255     }
256 }

```

Результат работы полученного консольного приложения показан на рис. 15.1.

```

Выполнение запросов LINQ к массиву объектов
- Крупа гречневая, 42 шт.;
- Говядина тушеная, 30 шт.;

7. Средняя, наибольшая и наименьшая цены товаров Алтайпродукт:
- средняя цена: 64,50 руб.;
- наибольшая цена: 72,50 руб.;
- наименьшая цена: 56,50 руб.

8. Суммарный вес всех товаров:
Всего 89,93 кг.

9. Общая стоимость товаров каждого наименования:
- Говядина тушеная, 2880,00 руб.;
- Крупа гречневая, 3045,00 руб.;
- Крупа рисовая, 2244,55 руб.;
- Кукуруза консервированная, 3805,60 руб.;
- Макароны, 2712,00 руб.;

10. Наименования товаров, сгруппир. по производителям:
Товары фирмы ООО Алтайпродукт:
- Крупа гречневая
- Макароны
Товары фирмы ОАО Балтпром:
- Кукуруза консервированная
- Говядина тушеная
Товары фирмы ЗАО Увелка:
- Крупа рисовая

11. Наименования и количество товаров фирм ОАО Балтпром
или ЗАО Увелка с указанием данных по стеллажам, на которых они хранятся
:
: Наименование!Количество! Код стел!Число ярусов! Макс. груз, кг!
: Кукуруза консервированная! 67! 017! 6! 12000!
: Крупа рисовая! 53! 024! 4! 10000!
: Говядина тушеная! 30! 017! 6! 12000!

```

Рисунок 15.1. Результат работы консольного приложения

Выполнение запросов к коллекции объектов с помощью лямбда-выражений.

Требуется разработать следующие запросы LINQ к коллекциям объектов, полученных в примере 2.1:

1. Наименования и производители товаров с весом больше «вес», отсортированные по цене товара.
2. Наименования товаров с количеством от «мин. количество» до «макс. количество», сгруппированные по производителям.
3. Наименования товаров с весом больше «вес» и коды стеллажей, на которых хранятся эти товары, с числом ярусов меньше «число».

Указанные LINQ-запросы должны быть реализованы с помощью лямбда-выражений.

Исходный код методов, реализующих заданные запросы, представлен в листинге 15.5

Исходный код метода Main консольного приложения, в котором выполняется вызов методов-запросов, приведен в листинге 15.6.

# Листинг 15.5. Исходный код методов, выполняющих запросы LINQ с помощью лямбда-выражений

```

271 /// <summary>
272 /// Выводит наименования и производителей товаров с весом больше заданного
273 /// отсортированные по цене товара
274 /// </summary>
275 /// <param name="products">Список товаров</param>
276 /// <param name="weight">Вес, гр.</param>
277 static void Query1Lambda(List<Product> products, int weight)
278 {
279     Console.WriteLine("\n1. Наименования и производители товаров с весом > {0} гр.,\n"
280         "отсортированные по цене:", weight);
281     var result = products
282         .Where(p => p.Weight > weight)
283         .OrderBy(p => p.Price);
284     foreach (var elem in result)
285     {
286         Console.WriteLine("- {0}, {1}", elem.Name, elem.ProducedBy);
287     }
288 }
289
290 /// <summary>
291 /// Выводит число единиц товаров указанной фирмы
292 /// </summary>
293 /// <param name="products">Список товаров</param>
294 /// <param name="firm">Фирма-производитель</param>
295 static void Query2Lambda(List<Product> products, string firm)
296 {
297     Console.WriteLine("\n2. Число единиц товаров фирмы {0}:", firm);
298     var result = products
299         .Where(p => p.ProducedBy == firm)
300         .Select(p => p.NumberInStok);
301     Console.WriteLine("{0} шт.", result.Sum());
302 }
303
304 /// <summary>
305 /// Выводит наименования товаров с количеством в заданных пределах,
306 /// сгруппированные по производителям
307 /// </summary>
308 /// <param name="products">Список товаров</param>
309 /// <param name="minAmount">Максимальное количество</param>
310 /// <param name="maxAmount">Минимальное количество</param>
311 static void Query3Lambda(List<Product> products, int minAmount, int maxAmount)
312 {
313     Console.WriteLine("\n3. Наименования товаров с количеством от {0} до {1} шт.,\n"
314         "сгруппированные по производителям:", minAmount, maxAmount);
315     var groups = products
316         .Where(p => p.NumberInStok > minAmount &&
317             p.NumberInStok < maxAmount)
318         .GroupBy(p => p.ProducedBy);
319     foreach (var group in groups)
320     {
321         Console.WriteLine("{0}", group.Key);
322         foreach (var elem in group)
323         {
324             Console.WriteLine("- {0} ({1} шт.)", elem.Name, elem.NumberInStok);
325         }
326     }
327 }

```

```

329  /// <summary>
330  /// Выводит наименования товаров с весом больше заданного и коды стеллажей,
331  /// на которых хранятся эти товары, с числом ярусов меньше заданного
332  /// </summary>
333  /// <param name="products">Список товаров</param>
334  /// <param name="stores">Список стеллажей</param>
335  /// <param name="prodWeight">Вес товара, гр.</param>
336  /// <param name="numFlour">Число ярусов стеллажа</param>
337  static void Query4Lambda(List<Product> products, List<Store> stores,
338  int prodWeight, int numFlour)
339  {
340  Console.WriteLine("\n4. Наименования товаров с весом > {0} гр. и " +
341  "коды стеллажей, на которых хранятся эти товары, с числом ярусов < {1} шт.:",
342  prodWeight, numFlour);
343  var result = products
344  .Where(p => p.Weight > prodWeight)
345  .Join(
346  // последовательность inner
347  stores.Where(s => s.NumberOfFlour < numFlour),
348  p => p.StoreID, // outerKeySelector
349  s => s.StoreID, // innerKeySelector
350  (p, s) => new // resultSelector (анонимный тип)
351  {
352  name = p.Name,
353  weight = p.Weight,
354  stId = s.StoreID,
355  numF = s.NumberOfFlour
356  }
357  );
358  string rowFormat = "|{0,16}|{1,10}|{2,10}|{3,14}|";
359  Console.WriteLine(rowFormat, "Наименование", "Вес, гр.", "Код стел.",
360  "Число ярусов");
361  foreach (var elem in result)
362  {
363  Console.WriteLine(rowFormat, elem.name, elem.weight, elem.stId, elem.numF);
364  }
365  }

```

Листинг 15.6. Исходный код метода Main консольного приложения

```

77  Console.Clear();
78  Console.Title = "Выполнение запросов LINQ с помощью лямбда-выражений";
79
80  Console.WriteLine("***** Результаты запросов LINQ *****");
81  Query1Lambda(products, 340);
82  Query2Lambda(products, "000 Алтайпродукт");
83  Query3Lambda(products, 10, 50);
84  Query4Lambda(products, stores, 100, 5);
85
86  Console.ReadLine();
87  }

```

Результаты выполнения LINQ-запросов к коллекциям объектов показаны на рис. 15.2.

```

Выполнение запросов LINQ с помощью лямбда-выражений
***** Результаты запросов LINQ *****
1. Наименования и производители товаров с весом > 340 гр.,
отсортированные по цене>:
- Крупа рисовая, ЗАО Увелка
- Макароны, ООО Алтайпродукт
- Крупа гречневая, ООО Алтайпродукт
2. Число единиц товаров фирмы ООО Алтайпродукт: 90 шт.
3. Наименования товаров с количеством от 10 до 50 шт.,
сгруппированные по производителям:
ООО Алтайпродукт
- Крупа гречневая <42 шт.>
- Макароны <48 шт.>
ООО Балтпрон
- Говядина тушеная <30 шт.>
4. Наименования товаров с весом > 100 гр. и коды стеллажей, на которых хранятся
эти товары, с числом ярусов < 5 шт.:
Наименование! Вес, гр.! Код стелл.! Число ярусов!
Крупа гречневая! 350! 024! 4!
Крупа рисовая! 350! 024! 4!
Макароны! 500! 028! 4!

```

Рисунок 15.2. Результаты LINQ-запросов в окне консоли

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать библиотеку классов на языке C#, содержащую классы с указанными свойствами (табл. 15.1).
3. Добавить в решение консольное приложение на языке C#, содержащее коллекцию объектов (массив, список и др.) полученных классов. Число элементов коллекции должно быть не менее пяти.
4. Разработать и выполнить заданные запросы LINQ к коллекции объектов (табл. 15.2). Для запроса №6 следует использовать операцию group, а для запроса №7 – операцию join.
5. Создать заданные запросы LINQ к коллекциям объектов (табл. 15.3), используя лямбда-выражения. В запросе №2 требуется использовать операцию Group, а в запросе №3 – операцию Join.
6. Оформить и защитить отчет по лабораторной работе.



## Индивидуальные варианты заданий.

Таблица 15.1

Варианты заданий для разработки классов

Вариант	Данные для разработки классов
1,13	<b>Студент.</b> Код, ФИО, группа, пол, дата рождения, средний бал, код научного руководителя. <b>Научный руководитель.</b> Код, ФИО, должность.
2,14	<b>Автомобиль.</b> Код, фирма, модель, год выпуска, цена, расход топлива, код магазина. <b>Магазин.</b> Код, название, телефон.
3,15	<b>Сотрудник.</b> Код, ФИО, пол, дата рождения, должность, зарплата, код отдела. <b>Отдел.</b> Код, название, телефон.
4,16	<b>Книга.</b> Код, название, автор, цена, число страниц, год издания, код стеллажа. <b>Стеллаж.</b> Код, число полок, название отдела.
5,17	<b>Учебная дисциплина.</b> Код, название, ФИО преподавателя, форма контроля, семестр, число часов, код специальности. <b>Специальность.</b> Код, название, институт.
6,18	<b>Банковский вклад.</b> Код, номер счета, ФИО вкладчика, дата вклада, сумма, процент, код банка. <b>Банк.</b> Код, название, адрес центрального офиса.
7,19	<b>Предмет обуви.</b> Код, наименование, производитель, число пар, размер, цена, код отдела. <b>Отдел.</b> Код, название, число сотрудников.
8,20	<b>Ноутбук.</b> Код, фирма, модель, процессор, объем памяти, цена, код магазина. <b>Магазин.</b> Код, название, адрес.
9,21	<b>Билет на междугородный автобус.</b> Код, рейс, пункт назначения, время отправления, длительность, номер места, код автобуса. <b>Автобус.</b> Код, модель, число посадочных мест.
10,22	<b>Квартира.</b> Код, дом, номер, этаж, площадь, цена, код агентства. <b>Агентство недвижимости.</b> Код, название, телефон.



Продолжение таблицы 15.1

11,23	<b>Заказ на перевозку груза.</b> Код, номер, дата, адрес доставки, вес груза, стоимость перевозки, код водителя. <b>Водитель.</b> Код, ФИО, дата рождения.
12,24	<b>Спортсмен.</b> Код, ФИО, вид спорта, дата рождения, пол, рост, вес, код тренера. <b>Тренер.</b> Код, ФИО, звание.

Таблица 15.2

Варианты заданий для создания LINQ-запросов к коллекции объектов

Вариант	Данные для создания запросов к коллекции объектов
1,13	<ol style="list-style-type: none"> <li>1. Данные по студентам мужского пола.</li> <li>2. ФИО студентов с датой рождения «дата».</li> <li>3. Число студентов, у которых средний бал более «бал».</li> <li>4. ФИО и даты рождения студентов группы «группа».</li> <li>5. Общий средний бал для всех студентов группы «группа».</li> <li>6. ФИО студентов, сгруппированные по студенческим группам.</li> <li>7. ФИО студента и его группа с указанием ФИО и должности научного руководителя.</li> </ol>
2,14	<ol style="list-style-type: none"> <li>1. Данные по автомобилям фирмы «фирма».</li> <li>2. Модели автомобилей с годом выпуска «год выпуска».</li> <li>3. Число автомобилей с ценой от «цена» до «цена».</li> <li>4. Фирмы и модели автомобилей с расходом топлива менее «...».</li> <li>5. Средняя цена автомобиля фирмы «фирма».</li> <li>6. Модели автомобилей, сгруппированные по коду магазина.</li> <li>7. Фирмы и модели автомобилей с указанием названия и телефона магазина.</li> </ol>

## Продолжение таблицы 15.2

3,15	<ol style="list-style-type: none"> <li>1. Данные по сотрудникам, занимающим должность «должность».</li> <li>2. ФИО сотрудников женского пола.</li> <li>3. Число сотрудников с датой рождения «дата».</li> <li>4. ФИО и должности сотрудников с зарплатой от «...» до «...».</li> <li>5. Суммарная зарплата сотрудников с должностью «должность».</li> <li>6. ФИО сотрудников, сгруппированные по коду отдела.</li> <li>7. ФИО и даты рождения сотрудников с указанием названия и телефона отдела.</li> </ol>
4,16	<ol style="list-style-type: none"> <li>1. Данные по книгам автора «автор».</li> <li>2. Названия книг, изданных в «год издания».</li> <li>3. Число книг с ценой от «цена» до «цена».</li> <li>4. Авторы и названия книг с числом страниц более «число страниц».</li> <li>5. Средняя цена одной книги автора «автор».</li> <li>6. Названия книг, сгруппированные по коду стеллажа.</li> <li>7. Названия и число страниц книг с указанием числа полок стеллажа и отдела, в котором он расположен.</li> </ol>
5,17	<ol style="list-style-type: none"> <li>1. Данные по дисциплинам за семестр «семестр».</li> <li>2. Названия дисциплин с формой итогового контроля «...».</li> <li>3. Число дисциплин с количеством часов от «...» до «...».</li> <li>4. ФИО преподавателей и названия дисциплин за семестр «...».</li> <li>5. Общее число часов по дисциплинам преподавателя «фио».</li> <li>6. Названия дисциплин, сгруппированные по коду специальности.</li> <li>7. Названия и семестры дисциплин с указанием специальности и института.</li> </ol>

## Продолжение таблицы 15.2

6,18	<ol style="list-style-type: none"> <li>1. Данные по всем вкладам, сделанным «дата».</li> <li>2. Счета, на которых размещены вклады размером более «...».</li> <li>3. Число вкладов, по которым процент составляет от «...» до «...».</li> <li>4. Номер счета и ФИО вкладчика для вкладов с суммой менее «...».</li> <li>5. Средний процент по вкладам вкладчика «фио».</li> <li>6. Коды вкладов, сгруппированные по ФИО вкладчиков.</li> <li>7. Номера счетов и суммы вкладов с указанием названия и адреса банка.</li> </ol>
7,19	<ol style="list-style-type: none"> <li>1. Данные по всей обуви фирмы «...».</li> <li>2. Наименования обуви с ценой более «цена».</li> <li>3. Число наименований обуви с размером от «...» до «...».</li> <li>4. Производитель и наименование обуви с количеством менее «...».</li> <li>5. Суммарная стоимость обуви по каждому наименованию.</li> <li>6. Наименования обуви, сгруппированные по кодам отделов.</li> <li>7. Наименования и цены обуви с указанием названия отдела и числа сотрудников.</li> </ol>
8,20	<ol style="list-style-type: none"> <li>1. Данные по всем ноутбукам с процессором «...».</li> <li>2. Модели ноутбуков фирмы «...».</li> <li>3. Число моделей с ценой от «...» до «...».</li> <li>4. Модель и производитель ноутбуков с объемом памяти более «...».</li> <li>5. Средняя цена ноутбуков фирмы «фирма».</li> <li>6. Модели ноутбуков, сгруппированные по коду магазина.</li> <li>7. Модель и цена ноутбуков с указанием названия и адреса магазина.</li> </ol>

## Продолжение таблицы 15.2

9,21	<ol style="list-style-type: none"> <li>1. Данные по всем билетам с пунктом назначения «...».</li> <li>2. Рейсы с временем отправления «...».</li> <li>3. Число билетов с номерами мест от «...» до «...».</li> <li>4. Пункты назначения и рейсы с длительностью более «...».</li> <li>5. Средняя длительность рейсов в пункт назначения «...».</li> <li>6. Коды билетов, сгруппированные по пункту назначения.</li> <li>7. Рейсы и места с указанием модели автобуса и числа мест в нем.</li> </ol>
10,22	<ol style="list-style-type: none"> <li>1. Данные по всем продаваемым квартирам.</li> <li>2. Номера квартир, продаваемых в доме «дом».</li> <li>3. Число квартир с ценой менее «цена».</li> <li>4. Дома и номера квартир, расположенных на этажах от «...» до «...».</li> <li>5. Средняя стоимость квартир с площадью менее «площадь».</li> <li>6. Все квартиры, сгруппированные по коду агентства.</li> <li>7. Код и цена квартиры с указанием названия агентства и его телефона.</li> </ol>
11,23	<ol style="list-style-type: none"> <li>1. Данные по всем заказам на перевозку.</li> <li>2. Номера заказов, сделанных «дата».</li> <li>3. Число заказов со стоимостью перевозки более «стоимость».</li> <li>4. Даты и адреса доставки грузов весом от «вес» до «вес».</li> <li>5. Общий вес груза, переведенный «дата».</li> <li>6. Все заказы, сгруппированные по коду водителя.</li> <li>7. Номера и стоимости заказов с указанием ФИО и даты рождения водителя.</li> </ol>

Продолжение таблицы 15.2

12,24	<ol style="list-style-type: none"> <li>1. Данные по всем спортсменам.</li> <li>2. ФИО спортсменов мужского пола.</li> <li>3. Число спортсменов с весом от «вес» до «вес».</li> <li>4. Даты рождения и ФИО спортсменов, имеющих рост более «...».</li> <li>5. Средний рост спортсменов по «вид спорта».</li> <li>6. Все спортсмены, сгруппированные по видам спорта.</li> <li>7. ФИО и дата рождения спортсмена с указанием ФИО и звание тренера.</li> </ol>
-------	---

Таблица 15.3

Варианты заданий для создания LINQ-запросов к коллекции объектов с помощью лямбда-выражений

Вариант	Данные для создания запросов к коллекции объектов
1,13	<ol style="list-style-type: none"> <li>1. ФИО и группа студента со средним балом менее «бал».</li> <li>2. ФИО студентов, сгруппированные по полу.</li> <li>3. Код и ФИО студента с указанием кода и ФИО научного руководителя.</li> </ol>
2,14	<ol style="list-style-type: none"> <li>1. Фирмы и модели автомобилей с ценой менее «цена».</li> <li>2. Модели и цены автомобилей, сгруппированные по фирме.</li> <li>3. Коды и модели автомобилей с указанием кода и названия магазина.</li> </ol>
3,15	<ol style="list-style-type: none"> <li>1. Коды и ФИО сотрудников с полом «пол».</li> <li>2. ФИО и даты рождения сотрудников, сгруппированные по должности.</li> <li>3. ФИО и заработные платы сотрудников с указанием кода и названия отдела.</li> </ol>

## Продолжение таблицы 15.3

4,16	<ol style="list-style-type: none"> <li>1. Авторы и названия книг с ценой более «цена».</li> <li>2. Названия и число страниц книг, сгруппированные по автору.</li> <li>3. Названия и годы издания книг с указанием кода стеллажа и отдела, в котором расположен стеллаж.</li> </ol>
5,17	<ol style="list-style-type: none"> <li>1. Названия дисциплин с числом часов более «число» и формы контроля по дисциплинам.</li> <li>2. Названия дисциплин и ФИО преподавателей, сгруппированные по семестрам.</li> <li>3. Коды и названия дисциплин с указанием кодов и названий специальностей.</li> </ol>
6,18	<ol style="list-style-type: none"> <li>1. Номера счетов и суммы вкладов с процентом более «процент».</li> <li>2. Коды вкладов и номера счетов, сгруппированные по дате вклада.</li> <li>3. Номера счетов и ФИО вкладчиков с указанием кода и названия банка.</li> </ol>
7,19	<ol style="list-style-type: none"> <li>1. Наименование и производитель обуви с размером менее «размер».</li> <li>2. Наименования и цены обуви, сгруппированные по фирме.</li> <li>3. Коды и наименования обуви с указанием кодов названий отделов.</li> </ol>
8,20	<ol style="list-style-type: none"> <li>1. Модели и производители ноутбуков с ценой менее «цена».</li> <li>2. Модели и процессоры ноутбуков, сгруппированные по производителю.</li> <li>3. Коды и модели ноутбуков с указанием кода и названия магазина.</li> </ol>

Продолжение таблицы 15.3

9,21	1. Рейсы и времена отправления в пункт назначения «название». 2. Номера рейсов и цены билетов, сгруппированные по времени отправления. 3. Коды билетов и рейсы с указанием кода и модели автобуса.
10,22	1. Номера квартир и домов, в которых расположены квартиры, с ценой более «цена». 2. Номера и площади квартир, сгруппированные по этажам. 3. Коды и номера квартир с указанием кодов и названий агентств недвижимости.
11,23	1. Номера квартир и домов, в которых расположены квартиры, с ценой более «цена». 2. Номера и площади квартир, сгруппированные по этажам. 3. Коды и номера квартир с указанием кодов и названий агентств недвижимости.
12,24	1. ФИО и рост спортсменов с весом более «вес». 2. ФИО и даты рождения спортсменов, сгруппированные по полу. 3. Виды спорта и ФИО спортсменов с указанием кодов и ФИО тренеров.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего предназначена технология LINQ?
2. Какие выделяют разновидности LINQ по источнику данных?
3. Что такое неявно типизированные переменные и как они используются в LINQ?
4. Из каких этапов состоит выполнение запроса LINQ?
5. Каков синтаксис для простейшего выражения запроса LINQ?
6. Как задается фильтрация данных в выражении запроса LINQ?

7. Каким образом задается сортировка результатов запроса LINQ?
8. Как можно сгруппировать данные запроса LINQ по определенному ключу?
9. Для чего предназначена операция join в выражении запроса LINQ?
10. Что называют анонимным типом и как он записывается в выражении запроса LINQ?
11. Какие средства языка C# называют анонимными функциями?
12. Что понимают под лямбда-выражениями и каков их синтаксис в языке C#?



## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 1. МЕТОДЫ СОРТИРОВКИ

### ЦЕЛЬ И СОДЕРЖАНИЕ

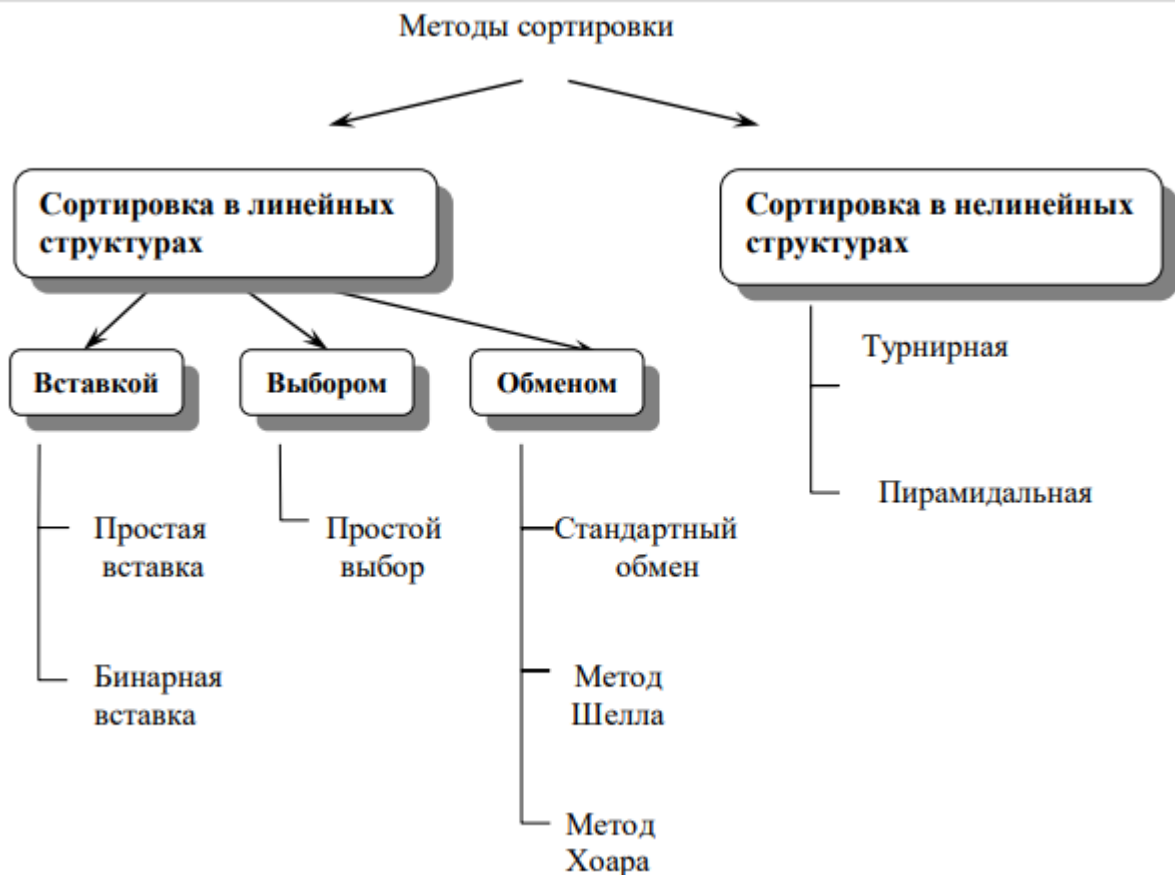
*Цель практической работы:* ознакомление с алгоритмами сортировки линейных и нелинейных структур и методикой оценки эффективности алгоритмов

### ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Упорядочение элементов множества в возрастающем или убывающем порядке называется сортировкой.

С упорядоченными элементами проще работать, чем с произвольно расположенными: легче найти необходимые элементы, исключить, вставить новые. Сортировка применяется при трансляции программ, при организации наборов данных на внешних носителях, при создании библиотек, каталогов, баз данных и т.д.

Алгоритмы сортировки можно разбить на следующие группы:



Обычно сортируемые элементы множества называют записями и обозначают через  $k_1, k_2, \dots, k_n$ .

## Сортировка выбором

Сортировка выбором состоит в том, что сначала в неупорядоченном списке выбирается и отделяется от остальных наименьший элемент. После этого исходный список оказывается изменённым. Изменённый список принимается за исходный и процесс продолжается до тех пор, пока все элементы не будут выбраны. Очевидно, что выбранные элементы образуют упорядоченный список.

Например, требуется найти минимальный элемент списка:

{5, 11, 6, 4, 9, 2, 15, 7}

Процесс выбора показан на рис.1, где в каждой строчке выписаны сравниваемые пары. Выбираемые элементы с меньшим весом обведены кружком. Нетрудно видеть, что число сравнений соответствует на рисунке числу строк, а число перемещений – количеству изменений выбранного элемента.

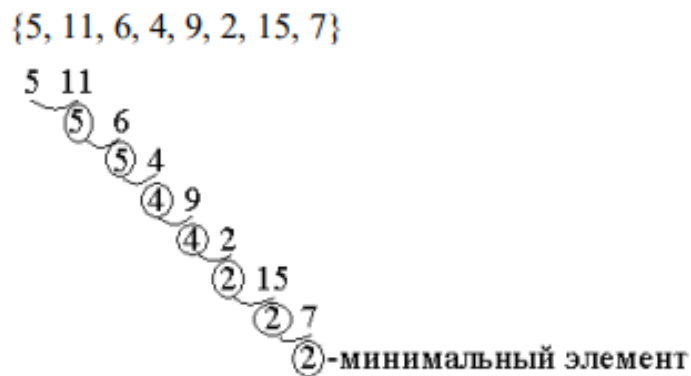


Рисунок 1. Сортировка выбором

Выбранный в исходном списке минимальный элемент размещается на предназначенном ему месте несколькими способами:

- Минимальный элемент после  $i$ -го просмотра перемещается на  $i$ -ое место нового списка ( $i = 1, 2, \dots, n$ ), а в исходном списке на место выбранного элемента записывается какое-то очень большое число, превосходящее по величине любой элемент списка, при этом длина заданного списка остаётся постоянной. Изменённый таким образом список можно принимать за исходный.
- Минимальный элемент записывается на  $i$ -ое место исходного списка ( $i = 1, 2, \dots, n$ ), а элемент с  $i$ -го места - на место

выбранного. При этом очевидно, что уже упорядоченные элементы (а они будут расположены, начиная с первого места ) исключаются из дальнейшей сортировки, поэтому длина каждого последующего списка (списка, участвующего в каждом последующем просмотре) должна быть на 1 элемент меньше предыдущего.

- Выбранный минимальный элемент, как и в предыдущем случае, перемещается на  $i$ -ое место заданного списка, а чтобы это  $i$ -ое место освободилось для записи очередного минимального элемента, левая от выбранного элемента часть списка перемещается вправо на одну позицию так, чтобы заполнилось место, занимаемое до этого выбранным элементом.

Сложность метода сортировки выбором порядка  $O(n^2)$

### **Сортировка вставкой**

В этом методе из неупорядоченной последовательности элементов выбирается поочередно каждый элемент, сравнивается с предыдущим, уже упорядоченным, и помещается на соответствующее место. Сортировку вставкой рассмотрим на примере заданной неупорядоченной последовательности элементов: {40, 11, 83, 57, 32, 21, 75, 64}

Процедура сортировки отражена на рис.2, где кружком на каждом этапе обведён анализируемый элемент, стрелкой сверху отмечено место перемещения анализируемого элемента, в рамку заключены упорядоченные части последовательности.

Этапы:

1-й	$\overline{40, 11}, 83, 57, 32, 21, 75, 64$ $\boxed{11, 40}, 83, 57, 32, 21, 75, 64$
2-й	$11, 40, \overline{83}, 57, 32, 21, 75, 64$ $\boxed{11, 40, 83}, 57, 32, 21, 75, 64$
3-й	$11, 40, \overline{57}, 83, 32, 21, 75, 64$ $\boxed{11, 40, 57, 83}, 32, 21, 75, 64$
4-й	$11, \overline{40, 57, 83}, \overline{32}, 21, 75, 64$ $\boxed{11, 32, 40, 57, 83}, 21, 75, 64$
5-й	$11, \overline{32, 40, 57, 83}, \overline{21}, 75, 64$ $\boxed{11, 21, 32, 40, 57, 83}, 75, 64$
6-й	$11, 21, 32, 40, 57, \overline{75}, \overline{83}, 64$ $\boxed{11, 21, 32, 40, 57, 75, 83}, 64$
7-й	$11, 21, 32, 40, 57, \overline{64}, \overline{75}, 83$ $\boxed{11, 21, 32, 40, 57, 64, 75, 83}$

Рисунок 2. Сортировка вставкой

На первом этапе сравниваются два начальных элемента. Поскольку второй элемент меньше первого, он перемещается на место первого элемента, который сдвигается вправо на одну позицию. Остальная часть последовательности остаётся без изменения. На втором этапе из неупорядоченной последовательности выбирается элемент и сравнивается с двумя упорядоченными ранее элементами. Так как он больше предыдущих, то остаётся на месте. Затем анализируются четвёртый, пятый и последующие элементы – до тех пор, пока весь список не будет упорядоченным, что имеет место на последнем (седьмом) этапе.

Разновидностью сортировки вставкой является метод фон Неймана. Пусть заданы два упорядоченных по возрастанию элементов одномерных массива:  $a$  размерности  $n$  и  $b$  размерности  $m$ . Требуется получить третий массив с размерности  $n+m$ , который содержал бы все элементы исходных массивов, упорядоченных по возрастанию.

Алгоритм решения этой задачи, известный как «сортировка фон Неймана» или сортировка слиянием, состоит в следующем:

сначала анализируются первые элементы обоих массивов. Меньший элемент переписывается в новый массив. Оставшийся элемент последовательно сравнивается с элементами из другого массива. В новый массив после каждого сравнения попадает меньший элемент. Процесс продолжается до исчерпания элементов одного из массивов. Затем остаток другого массива дописывается в новый массив. Полученный новый массив упорядочен таким же образом, как исходные.

Сложность метода сортировки вставкой порядка  $O(n^2)$ .

### **Сортировка обменом**

Сортировка обменом – метод, в котором элементы списка последовательно сравниваются между собой и меняются местами в том случае, если предшествующий элемент больше последующего. Требуется, например, провести сортировку списка методом стандартного обмена или методом «пузырька»: {40, 11, 83, 57, 32, 21, 75, 64}

Обозначим квадратными скобками со стрелками обмениваемые элементы, а - сравниваемые элементы. Первый этап сортировки показан на рис.3, а второй этап – на рис.4.

Нетрудно видеть, что после каждого просмотра списка все элементы, начиная с последнего, занимают свои окончательные позиции, поэтому их не следует проверять при следующих просмотрах. Каждый последующий просмотр исключает очередную позицию с найденным максимальным элементом, тем самым укорачивая список. После первого просмотра в последней позиции оказался больший элемент, равный 83 (исключаем его из дальнейшего рассмотрения).

Второй просмотр выявляет максимальный элемент, равный 75 (рис.4).

Процесс сортировки продолжается до тех пор, пока не будут сформированы все элементы конечного списка, либо не выполнится условие Айверсона.

Исходный список	40	11	83	57	32	21	75	64
Первый просмотр	11	40	83	57	32	21	75	64
Полученный список	11	40	57	32	21	75	64	83

Рисунок 3. Сортировка обменом (первый просмотр )

Исходный список	11	40	57	32	21	75	64
Второй просмотр	11	40	57	32	21	75	64
Полученный список	11	40	32	21	57	64	75

Рисунок 4. Сортировка обменом (второй просмотр)

**Условие Айверсона:** если в ходе сортировки при сравнении элементов не было сделано ни одной перестановки, то множество считается упорядоченным (условие Айверсона выполняется только при шаге  $d=1$ ).

Модификацией сортировки стандартным обменом является шейкерная или челночная сортировка . Здесь, как и в методе пузырька проводится по парное сравнение элементов . При этом первый проход осуществляется слева направо, второй – справа налево и т.д. Иными словами меняется направление просмотра элементов списка.

Сложность метода стандартного обмена  $O(n^2)$ .

### Сортировка Шелла

В методе Шелла сравниваются не соседние элементы, а элементы, расположенные на расстоянии  $d$  (где  $d$  – шаг между сравниваемыми элементами)  $d = \lfloor n/2 \rfloor$ . После каждого просмотра шаг



Исходный список	32	11	40	21	75	57	83	64
d = 1	↑ 11	↑ 32 32	↓ 40 21	↑ 40 40	↓ 75 57	↑ 75 75	↓ 83 64	↑ 83
Полученный список	11	32	21	40	57	75	64	83

Рисунок 7. Метод Шелла (d=1 )

### Быстрая сортировка (сортировка Хоара)

В методе быстрой сортировки фиксируется какой-либо ключ (базовый), относительно которого все элементы с большим весом перебрасываются вправо, а с меньшим – влево. При этом весь список элементов делится относительно базового ключа на две части. Для каждой части процесс повторяется.

Поясним метод на примере.

На рис.8 представлен первый этап быстрой сортировки. В первой строке указана исходная последовательность.

Примем первый элемент последовательности за базовый ключ, выделим его квадратом и обозначим  $k_0 = 40$ . Установим два указателя :  $i$  и  $j$ , из которых  $i$  начинает отсчёт слева ( $i=1$ ), а  $j$  – справа ( $j=n$ ).

Сравниваем базовый ключ  $k_0$  и текущий ключ  $k_j$ . Если  $k_0 \leq k_j$ , то устанавливаем  $j=j-1$  и проводим следующее сравнение  $k_0$  и  $k_j$ . Продолжаем уменьшать  $j$  до тех пор, пока не достигнем условия  $k_0 > k_j$ . После этого меняем местами ключи  $k_0$  и  $k_j$  (шаг 3 на рис.8 ).



Номер шага	<div style="display: flex; justify-content: space-between; align-items: center;"> <span><math>i \longrightarrow</math></span> <span><math>\longleftarrow j</math></span> </div>								Примечание
	40	11	83	57	32	21	75	64	Исходный список
1	40							64	$k_0 < k_j$
2	40						75		$k_0 < k_j$
3	40						21		Обмен; $k_0 > k_j$
	21						40		
4		11					40		$k_i < k_0$
5			83				40		Обмен; $k_i > k_0$
			40				83		
6			40				32		Обмен; $k_0 > k_j$
			32				40		
7				57			40		Обмен; $k_i > k_0$
				40			57		
	21	11	32	40	57	83	75	64	Полученный список

Рисунок 8. Метод Хоара

Теперь начинаем изменять индекс  $i = i + 1$  и сравнивать элементы  $k_i$  и  $k_0$ . Продолжаем увеличение  $i$  до тех пор, пока не получим условие  $k_i > k_0$ , после чего следует обмен  $k_i$  и  $k_0$  (см. шаг 5). Снова возвращаемся к индексу  $j$ , уменьшаем его. Чередую уменьшение  $j$  и увеличение  $i$ , продолжаем этот процесс с обоих концов к середине до тех пор, пока не получим  $i = j$  (см. шаг 7).

В отличие от предыдущих рассмотренных сортировок уже на первом этапе имеют место два факта: во-первых, базовый ключ  $k_0 = 40$  занял своё постоянное место в сортируемой последовательности; во-вторых, все элементы слева от  $k_0$  будут меньше него, а справа больше него. Таким образом, по окончании первого этапа имеем:

$$\begin{array}{ccccccc}
 \underline{21, \quad 11, \quad 32} & \boxed{40} & \underline{57, \quad 83, \quad 75, \quad 64} \\
 \text{левая часть} & & \text{правая часть}
 \end{array}$$

Указанная процедура сортировки применяется независимо к левой и правой частям.

Сложность метода Хоара  $O(n \log_2 n)$ .

### Сортировка в нелинейных структурах

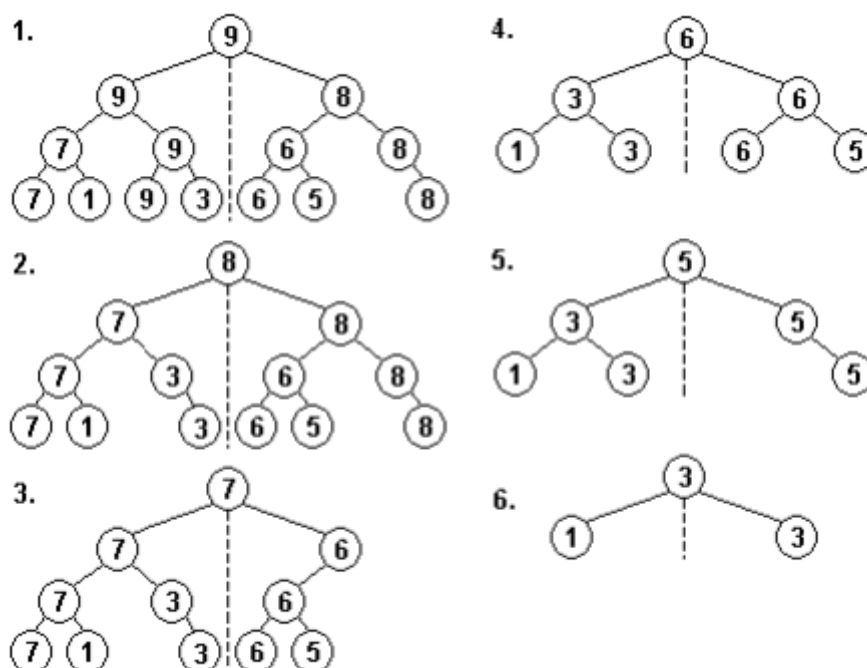
Сортировка в нелинейных структурах осуществляется только на бинарных деревьях (деревьях, из каждой вершины которого выходит по два ребра).

### Турнирная сортировка

Свое название эта сортировка получила, потому что она используется при проведении соревнований, турниров и олимпиад. Элементы исходного множества представляются листьями дерева. Их по парное сравнение позволяет определить максимальный элемент.

Пример: Дано исходное множество  $\{7, 1, 9, 3, 6, 5, 8\}$

Производится по парное сравнение элементов снизу- вверх. Найденный максимальный элемент помещается в результирующее множество.



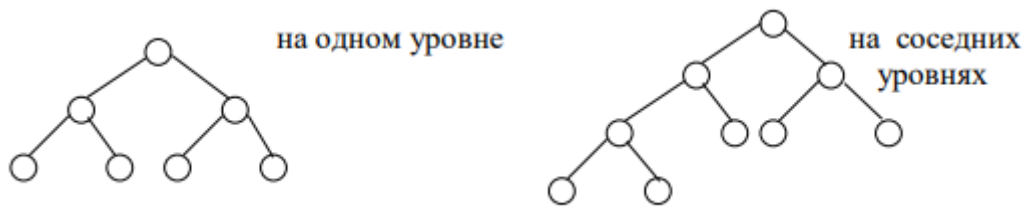
В результате будет получено упорядоченное множество  $\{9, 8, 7, 6, 5, 3\}$

### Пирамидальная сортировка

Данный тип сортировки заключается в построение пирамидального дерева.

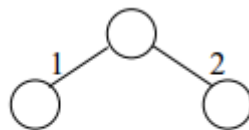
Пирамидальное дерево – это бинарное дерево обладающее тремя свойствами:

- В вершине каждой триады располагается элемент с большим весом.
- Листья бинарного дерева располагаются либо в одном уровне либо в двух соседних



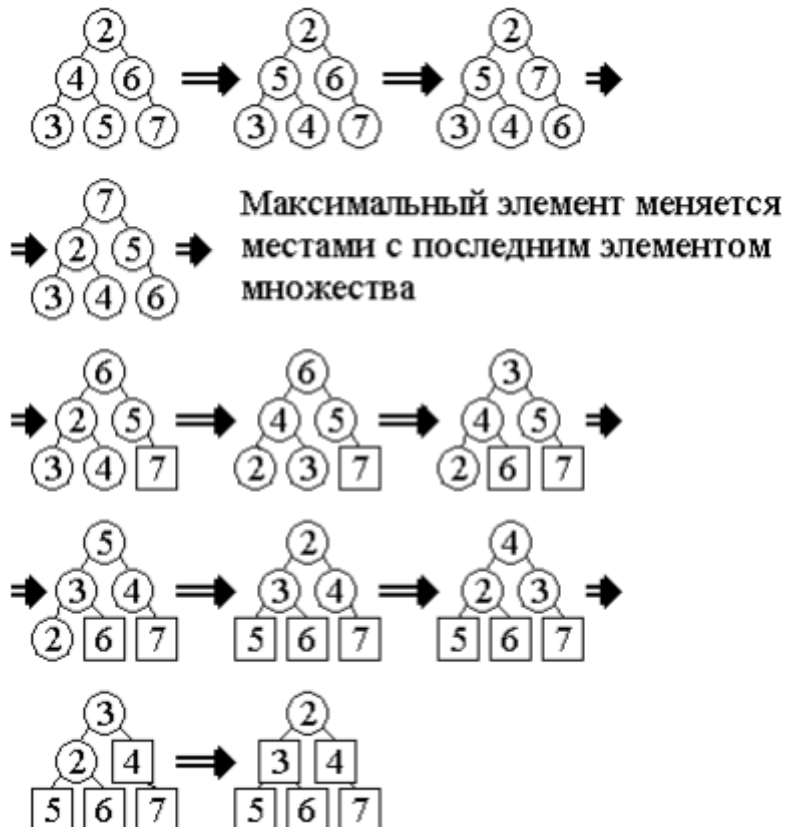
• Листья нижнего уровня располагаются левее листьев более высокого уровня.

В ходе преобразования элементы триад сравниваются дважды, при этом элемент с большим весом перейдет вверх, а с меньшим – вниз.



1 - первое сравнение  
2 - второе сравнение

Пример: Дано исходное множество  $\{ 2, 4, 6, 3, 5, 7 \}$

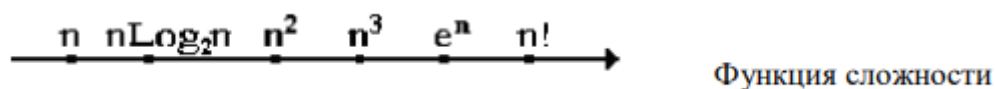


В результате будет получено упорядоченное множество  $\{ 2, 3, 4, 5, 6, 7 \}$

### Функция сложности алгоритма

Для оценки эффективности алгоритмов используется функция сложности алгоритма, которая обозначается заглавной буквой “О”, в круглых скобках записывается аргумент. Например, функция сложности  $O(n^2)$  читается как функция сложности порядка  $n^2$ . Функция сложности алгоритма – это функция, которая определяет количество сравнений, перестановок а так временные и ресурсные затраты на реализацию алгоритма.

Функция сложности принимает следующий ряд значений:



Чем правее на оси расположена функция сложности, тем сложнее алгоритм.

### ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Для каждого из перечисленных методов сортировки провести анализ временных затрат для списков различной размерности.

Для указанных в практической работе методов сортировки провести следующие исследования:

Выполнить сортировку массива из  $N$  чисел (С помощью написанной программы на С#).

Результаты занести в таблицу.

Метод			
Количество элементов сортируемого массива $N$	$N_1$	$N_2$	$N_3$
Время сортировки $t$ , с	$t_1$	$t_2$	$t_3$

- Оценить сложность рассмотренных методов сортировки;
- провести анализ отклонения полученной в результате эксперимента сложности алгоритма от теоретической;
- построить графические зависимости времени сортировки от количества элементов сортируемого массива.

Вариант	n1	n2	n3	Составить программу
1,15	1000	4000	6000	Простая вставка
2,16	800	3000	7000	Сортировка слиянием
3,17	2000	5000	6800	Метод Шелла
4,18	1500	3500	6000	Простой выбор
5,19	1000	2800	8500	Метод Хоара
6,20	500	2500	6500	Бинарная вставка
7,21	900	3000	8500	Шейкерная сортировка
8,22	1200	2000	7500	Простая вставка
9,23	2500	3500	6500	Шейкерная сортировка
10,24	1600	2800	8800	Метод Шелла
11,25	2200	3400	7200	Простой выбор
12,26	1900	2700	8000	Метод Хоара
13,27	1300	3500	6000	Бинарная вставка
14,28	1700	2800	7500	Сортировка слиянием

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под сортировкой?
2. Каковы особенности сортировки: вставкой, выбором, обменом, Шелла, Хоара, турнирной, пирамидой?
3. Что включает в себя понятие сложности алгоритма?
4. В чём состоит методика анализа сложности алгоритмов сортировки?

## ПРАКТИЧЕСКОЕ ЗАНЯТИЕ 2. МЕТОДЫ ПОИСКА

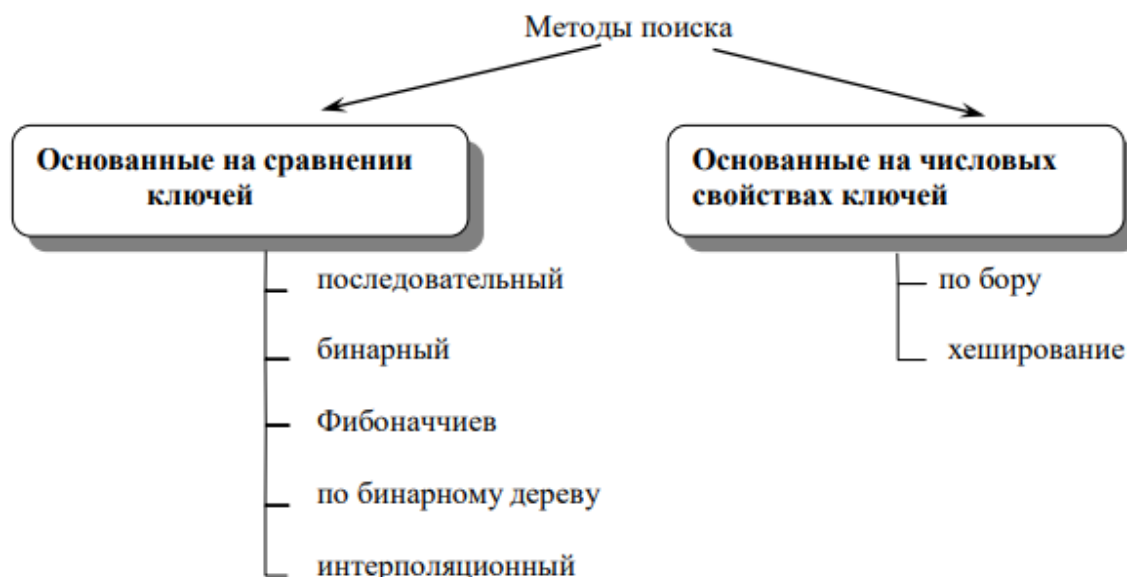
### ЦЕЛЬ И СОДЕРЖАНИЕ

*Цель практической работы:* ознакомление с алгоритмами поиска в линейных и нелинейных структурах и оценкой эффективности алгоритмов.

### ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Предметы (объекты), составляющие множество, называются его элементами. Элемент множества будет называться ключом, и обозначаться латинской буквой “ $k$ ” с индексом, указывающим номер элемента.

Алгоритмы поиска можно разбить на следующие группы:



Задача поиска:

Пусть дано множество ключей  $\{k_1, k_2, k_3 \dots k_n\}$ .

Необходимо отыскать во множестве ключ  $k_i$ . Поиск может быть завершён в двух случаях:

1. Ключ во множестве отсутствует;
2. Ключ найден во множестве.

#### Последовательный поиск.

В последовательном поиске исходное множество не упорядоченно, т.е. имеется произвольный набор ключей  $\{k_1, k_2, k_3 \dots k_n\}$ . Метод заключается в том, что отыскиваемый ключ  $k_i$  последовательно сравнивается со всеми элементами множества. При этом поиск заканчивается досрочно, если ключ найден.

### **Бинарный поиск.**

В бинарном поиске исходящее множество должно быть упорядоченно по возрастанию. Иными словами каждый последующий ключ больше предыдущего  $\{k_1 \leq k_2 \leq k_3 \leq k_4 \dots k_{n-1} \leq k_n\}$ .

Отыскиваемый ключ сравнивается с центральным элементом множества, если он меньше центрального, то поиск продолжается в левом подмножестве, в противном случае в правом. Центральный элемент находится по формуле  $N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1$ , где квадратные скобки обозначают, что от деления берётся только целая часть (всегда округляется в меньшую сторону). В методе бинарного поиска анализируются только центральные элементы.

Пример. Дано множество  $\{7, 8, 12, 16, 18, 20, 30, 38, 49, 50, 54, 60, 61, 69, 75, 79, 80, 81, 95, 101, 123, 198\}$ . Найти во множестве ключ  $K=61$ .

#### Шаг 1

$$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 22/2 \rfloor + 1 = 12$$

$$K \sim k_{12}$$

$61 > 60$  Дальнейший поиск в правом подмножестве  $\{61, 69, 75, 79, 80, 81, 95, 101, 123, 198\}$ . Значок " $\sim$ " обозначает сравнение элементов (чисел, значений).

#### Шаг 2

$$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 12/2 \rfloor + 1 = 7$$

$$K \sim k_{19}$$

$61 < 95$  Дальнейший поиск в левом подмножестве  $\{61, 69, 75, 79, 80, 81\}$  (относительно предыдущего подмножества).

#### Шаг 3

$$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 6/2 \rfloor + 1 = 4$$

$$K \sim k_{16}$$

$61 < 79$  Дальнейший поиск в левом подмножестве  $\{61, 69, 75, 79\}$ .

#### Шаг 4

$$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 4/2 \rfloor + 1 = 3$$

$$K \sim k_{15}$$

$61 < 75$  Дальнейший поиск в левом подмножестве  $\{61, 69\}$ .

#### Шаг 5

$$N_{\text{эл-та}} = \lfloor n/2 \rfloor + 1 = \lfloor 2/2 \rfloor + 1 = 2$$

$$K \sim k_{14}$$

$61 < 69$  Дальнейший поиск в левом подмножестве.

Шаг 6 $K \sim k_{13}$ 

61=61.

Вывод: искомый ключ найден под номером 13.**Фибоначчиев поиск.**

В этом поиске анализируются элементы, находящиеся в позициях, равных числам Фибоначчи. Числа Фибоначчи получаются по следующему правилу: каждое последующее число равно сумме двух предыдущих чисел, например:  $\{1, 2, 3, 5, 8, 13, 21, 34, 55, \dots\}$ .

Поиск продолжается до тех пор, пока не будет найден интервал между двумя ключами, где может располагаться отыскиваемый ключ.

Пример. Дано исходное множество ключей

$\{3, 5, 8, 9, 11, 14, 15, 19, 21, 22, 28, 33, 35, 37, 42, 45, 48, 52\}$

Пусть отыскиваемый ключ равен 42. ( $K=42$ ).

Последовательное сравнение отыскиваемого ключа будет проводиться в позициях, равных числам Фибоначчи:  $\{1, 2, 3, 5, 8, 13, 21, \dots\}$ .

Шаг 1.  $K \sim K_1$   $42 > 3 \Rightarrow$  отыскиваемый ключ сравнивается с ключом, стоящим в позиции равной числу Фибоначчи.

Шаг 2.  $K \sim K_2$   $42 > 5 \Rightarrow$  сравнение продолжается с ключом, стоящим в позиции равной следующему числу Фибоначчи.

Шаг 3.  $K \sim K_3$   $42 > 8 \Rightarrow$  сравнение продолжается

Шаг 4.  $K \sim K_5$   $42 > 11 \Rightarrow$  сравнение продолжается

Шаг 5.  $K \sim K_8$   $42 > 19 \Rightarrow$  сравнение продолжается

Шаг 6.  $K \sim K_{13}$   $42 > 35 \Rightarrow$  сравнение продолжается

Шаг 7.  $K \sim K_{18}$   $42 < 52 \Rightarrow$  найден интервал, в котором находится отыскиваемый ключ: от 13 до 18 позиции, т.е.  $\{35, 37, 42, 45, 48, 52\}$

В найденном интервале поиск вновь ведётся в позициях равных числам Фибоначчи.

**Интерполяционный поиск.**

Исходное множество должно быть упорядочено по возрастанию весов.

Первоначальное сравнение осуществляется на расстоянии шага  $d$ , который определяется по формуле:



$$d = \left\lceil \frac{(j-i)(K - K_i)}{K_j - K_i} \right\rceil$$

Где:

$i$  – номер первого рассматриваемого элемента

$j$  – номер последнего рассматриваемого элемента

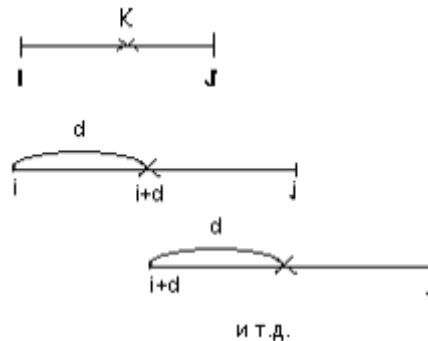
$K$  – отыскиваемый ключ

$K_i, K_j$  –

значения ключей в  $i$  и  $j$  позициях

$\lceil \rceil$  – целая часть от числа.

Идея метода заключается в следующем: шаг  $d$  меняется после каждого этапа, по формуле приведённой выше. Алгоритм заканчивает работу при  $d=0$ , при этом анализируются соседние элементы, после чего делается окончательное решение.



Этот метод прекрасно работает, если исходное множество представляет собой арифметическую прогрессию или множество, приближенное к ней.

Пример . Дано множество ключей: {2, 9, 10, 12, 20, 24, 28, 30, 37, 40, 45, 50, 51, 60, 65, 70, 74, 76}

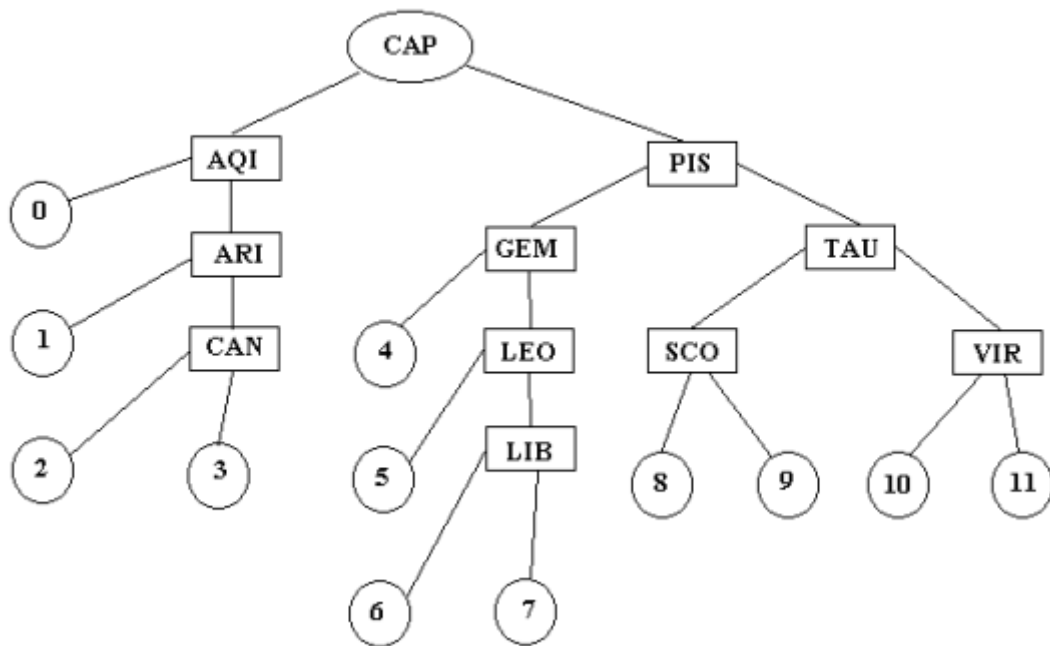
Шаг 1. Определим шаг  $d$  для исходного множества ключей:  
 $d = \lceil ((18-1)(70-2)/(76-2)) \rceil = 15$

Сравниваем ключ, стоящий под шестнадцатым порядковым номером в данном множестве с искомым ключом:  $K_{16} \sim K \Rightarrow 70 = 70$  ключ найден.

### Поиск по бинарному дереву.

Использование структуры бинарного дерева позволяет быстро вставлять и удалять записи и производить эффективный поиск по таблице. Такая гибкость достигается добавлением в каждую запись двух полей для хранения ссылок.

Пусть дано бинарное дерево:



Требуется по бинарному дереву отыскать ключ SAG.

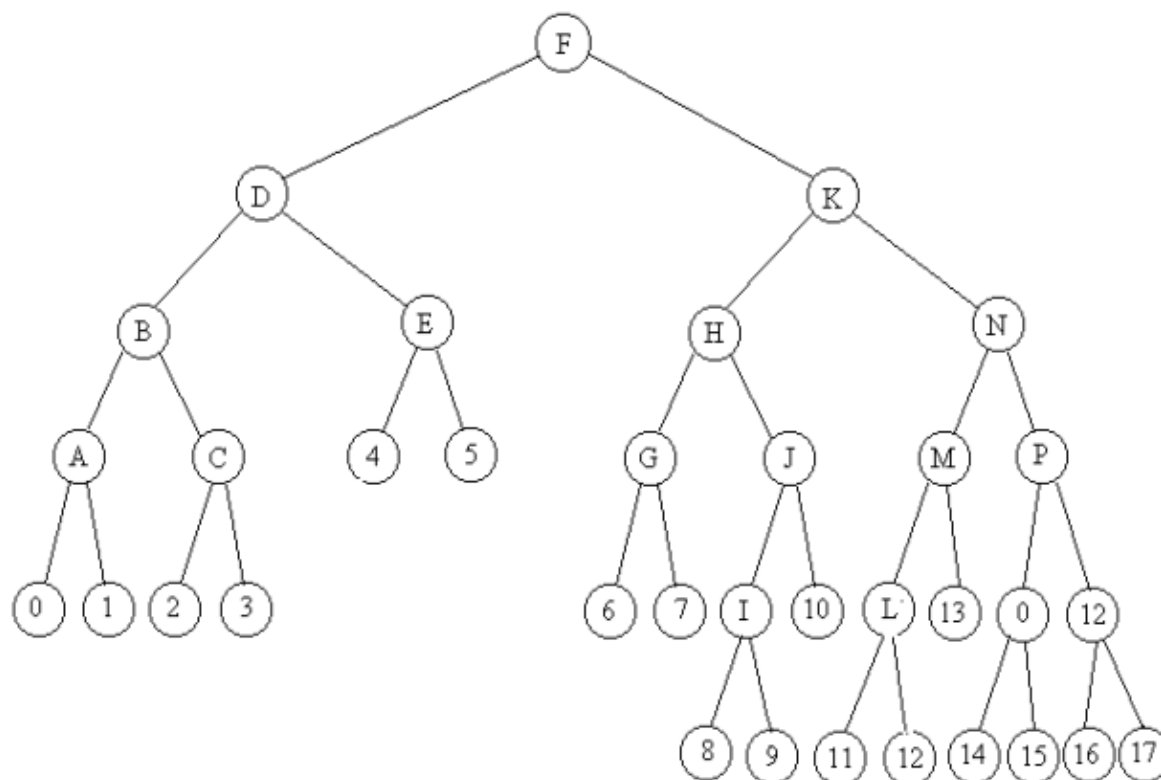
При просмотре от корня дерева видно, что по первой букве латинского алфавита, название SAG больше чем CAP. Следовательно, дальнейший поиск будем осуществлять в правой ветви. Это слово больше чем PIS - снова идем вправо; оно меньше чем TAU - идем влево; оно меньше чем SCO и попадаем в узел 8. Таким образом, название SAG должно находиться в узле 8.

При этом узлы дерева имеют следующую структуру:

Ключ	Информационная часть	Указатель на левое поддерево	Указатель на правое поддерево
	/может отсутствовать/		
	KEY	LLINK	RLINK

### Поиск по сбалансированному дереву.

Бинарное дерево называется сбалансированным, если высота левого поддерева каждого узла отличается от высоты правого не более чем на (+-1).



Сбалансированные бинарные деревья занимают промежуточное положение между оптимальными бинарными деревьями (все внешние узлы, которых расположены на двух смежных уровнях) и произвольными бинарными деревьями.

Бинарное дерево называется сбалансированным, если высота левого поддерева каждого узла отличается от высоты правого не более чем на  $(\pm 1)$ .

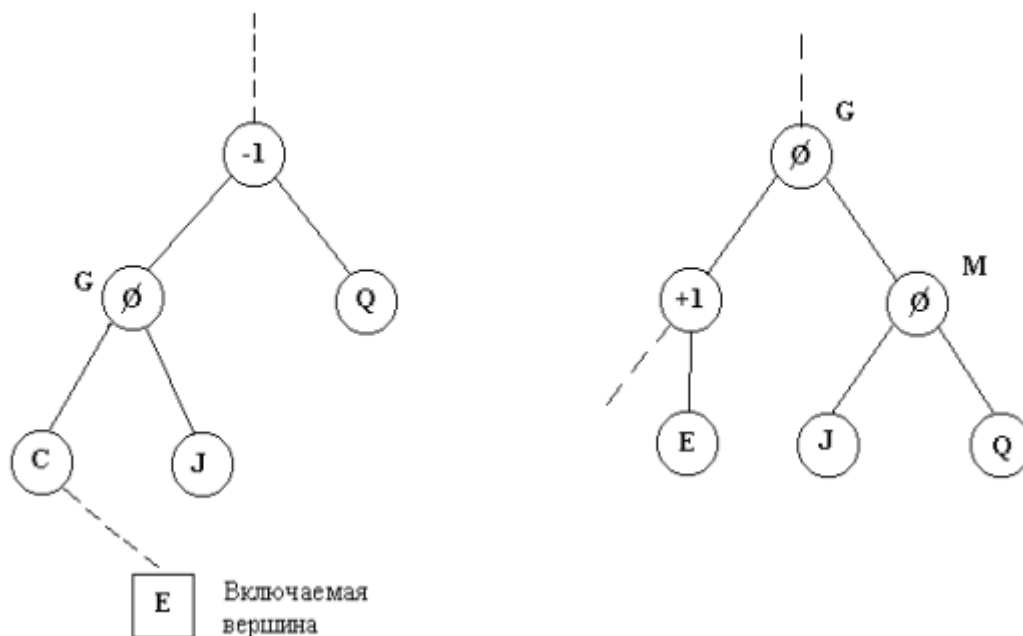
Рассмотрим следующую структуру узлов сбалансированного бинарного дерева:

Ключ	Указатель на левое поддерево	Указатель на правое поддерево	Показатель сбалансированности узла
KEY	LLINK	RLINK	B

где

B – показатель сбалансированности узла, то есть разность высот правого и левого поддерева ( $B = +1; 0; -1$ ).

При восстановлении баланса дерева по высоте учитывается показатель B.



Символы  $+1$ ,  $\emptyset$ ,  $-1$  указывают, что левое поддерево выше правого, поддеревья равны по высоте, правое поддерево выше левого.

### Поиск по бору

Особую группу методов поиска образует представление ключей в виде последовательности цифр и букв. Рассмотрим, например, имеющиеся во многих словарях буквенные высечки. Тогда по первой букве данного слова можно отыскать страницы, содержащие все слова, начинающиеся с этой буквы. Развивая идею побуквенных высечек, получим схему поиска, основанную на индексации в структуре бора (термин использует часть слова выборка).

Бор представляет собой  $m$ -арное дерево. Каждый узел уровня  $h$  представляет множество всех ключей, начинающихся с определенной последовательности из  $h$  литер. Узел определяет  $m$  – путевое разветвление в зависимости от  $(h + 1)$  –ой литеры.

Бор представляет собой таблицу, следующего вида:

Символы \ Узлы	Узлы				
	1	2	3	...	N
Пробел( )					

Пробел ( )- обязательный символ таблицы.

В первом узле записывается первая буква или цифра ключа. Во втором узле к ней добавляется ещё один символ и т.д. Если слово, начинающееся с определенной буквы (цифры) единственное, то оно сразу записывается в первом узле

Пример. Дано множество

{A, AA, AB, ABC, ABCD, ABCA, ABCC, BOR, C, CC, CCC, CCCD, CCCB, CCCA}.

От исходного множества перейдём к построению бора.

Исходный алфавит = {A,B,C,D}.

BOR- единственное слово на букву В и оно побуквенно не разбивается.

Узлы Символы	1	2	3	4	5	6	7
_		A_	AB_	ABC_	C_	CC_	CCC_
A	2	AA		ABCA			CCCA
B	BOR	3					CCCB
C	5		4	ABCC	6	7	
D				ABCD			CCCD

Узлы бора представляют собой векторы, каждая компонента которых представляет собой либо ключ, либо ссылку (возможно пустую).

Узел 1 – корень, и первую букву следует искать здесь. Если первой сказала, например, буква В, то из таблицы видно, что ему соответствует слово BOR. Если же первая буква А, то первый узел передает управление к узлу 2, где аналогичным образом отыскивается вторая буква. Узел 2 указывает, что вторыми буквами будут \_, А, В и т. д.

### Поиск хешированием

В основе поиска лежит переход от исходного множества к множеству хеш-функций  $h(k)$ .

Хеш-функция имеет следующий вид:  $h(k)=k \bmod (m)$ , где  $k$ -ключ,  $m$ - целое число,  $\bmod$ -целочисленный остаток от деления.

Например, пусть дано множество {9,1,4,10,8,5}.

Определим для него хеш-функцию  $h(k)=k \bmod(m)$ ;

Пусть  $m=1$ , тогда  $h(k)=\{0, 0, 0, 0, 0, 0\}$ ;

Пусть  $m=20$ , тогда  $h(k)=\{9, 1, 4, 10, 8, 5\}$ ;

Пусть  $m$  равно половине максимального ключа, тогда  $m=[10/2]=5$   $h(k)=\{4, 1, 4, 0, 3, 0\}$ ;

Хеш-функция указывает адрес, по которому следует отыскивать ключ. Для разных ключей хеш-функция может принимать одинаковые значения, такая ситуация называется коллизией. Таким образом, поиск хешированием заключается в устранении коллизий.

Пример. Дано множество  $\{7, 13, 6, 3, 9, 4, 8, 5\}$ . Найти ключ  $K=27$ .

Хеш-функция равна  $h(k)=K \bmod(m)$ ;  $m=[13/2]=6$  (т.к. 13-максимальный ключ).

$h(k)=\{1, 1, 0, 3, 3, 4, 2, 5\}$

Для устранения коллизий построим таблицу.

$h(k)$	Цепочки ключей
0	6
1	7, 12
2	8
3	3, 9
4	4
5	5

По парным сравнением множества хеш-функций и множества исходных ключей заполняем таблицу. Напоминаем, что хеш-функция указывает адрес, по которому следует отыскивать ключ.

Например, если отыскивается ключ  $K=27$ , тогда  $h(k)=27 \bmod 6=3$ -это значит, что ключ  $K=27$  может быть только в 3-й строке. Так как его там нет, то данный ключ отсутствует в исходном множестве.

## ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Для проведения практической работы необходимо выполнить следующие действия.

1. Выполнить тестовый пример поиска: выполнить поиск в массиве из  $N$  чисел (3 разных алгоритма поиска реализовать на C#).
2. Результаты занести в таблицу.

<b>Метод</b>			
<b>Количество элементов массива N</b>	$N_1$	$N_2$	$N_3$
<b>Время поиска t, с</b>	$t_1$	$t_2$	$t_3$

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под поиском?
2. Каковы особенности поиска: последовательного, бинарного, интерполяционного, Фибоначиевого, по бинарному дереву, по бору, хешированием?
3. В чём состоит методика анализа сложности алгоритмов поиска?

## СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Цель самостоятельной работы обучающихся – получить новые знания по дисциплине «Разработка кода информационных систем».

Самостоятельная работа необходима для формирования у обучающихся способности самостоятельно решать задачи профессиональной деятельности, формирования умения и навыков планирования времени, формирования стремления развиваться и совершенствоваться.

Виды самостоятельной работы обучающихся указаны в табл.1.

Таблица 1

Виды самостоятельной работы

№п/п	Вид СРС
1	Изучение литературы на тему «CASE-средства»
2	Изучение литературы на тему «Кроссплатформенность информационной системы»
3	Изучение литературы на тему «Репозиторий проекта»
4	Изучение литературы на тему «Типовые алгоритмы»



## **СПИСОК ЛИТЕРАТУРЫ**

### **ОСНОВНАЯ ЛИТЕРАТУРА**

1. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : Учебное пособие / Г. Н. Федорова. – Москва : НИЦ ИНФРА-М, 2023. – 336 с. – ISBN 978-5-906818-41-6. – URL: <https://znanium.com/catalog/document?id=416358> (дата обращения: 13.02.2024). – Текст : электронный.

### **ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА**

1. Казанский, А. А. Программирование на visual c#: учебное пособие для СПО / Казанский А. А.. – 2-е изд., пер. и доп. – Москва : Юрайт, 2020. – 192 с. – ISBN 978-5-534-14130-6. – URL: <https://urait.ru/book/programmirovanie-na-visual-c-2013-467844> (дата обращения: 13.02.2024). – Текст : электронный.

### **ПЕРЕЧЕНЬ ИНТЕРНЕТ-РЕСУРСОВ**

1. <http://metanit.com> - сайт посвящен различным языкам и технологиям программирования, компьютерам, мобильным платформам и ИТ-технологиям.

2. <https://msdn.microsoft.com/magazine/> - Интернет-журнал о технологиях разработки Microsoft. 3

3. <https://professorweb.ru> - информационный ресурс, посвященный разработке приложений на основе технологии .NET.