

ЧАСТЬ 2

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА

Основные понятия ООП

В этой Части мы рассмотрим важнейшее понятие — концепцию объектно-ориентированного программирования (ООП). Во введении уже отмечалось, что язык C# представляет собой средство объектно-ориентированного программирования.

И действительно, в языке C# все программы и данные представляют собой объекты, а все обрабатывающие их алгоритмы являются методами. Оба этих понятия имеют самое непосредственное отношение к ООП и будут предметом изучения в этой Части. Мы рассмотрим

- основные понятия объектно-ориентированного программирования (инкапсуляция, наследование, полиморфизм и др.);
- пользовательские типы (структуры и классы), взаимодействие между создаваемыми объектами при решении сложных задач;
- использование с внешних устройств (файлы данных);
- визуализацию выполнения программы (экранные формы) и др.

Владение методикой ООП абсолютно необходимо для успешного программирования на языке C#. Фактически, не разобравшись в этом, невозможно создавать на C# хоть сколько-нибудь сложные программы и системы.

Можно выделить пять основных черт ООП:

1. **Все является объектом.** Объект как хранит информацию, так и способен ее преобразовывать.
2. **Программа — совокупность объектов, указывающих друг другу что делать.** Для обращения к одному объекту другой объект «посылает ему сообщение».
3. **Каждый объект имеет свою собственную «память» состоящую из других объектов.** Таким образом программист может скрыть сложность программы за довольно простыми объектами.
4. **У каждого объекта есть тип.** Иногда тип называют еще и классом. Класс (тип) определяет, какие сообщения объекты могут посылать друг другу.
5. **Все объекты одного типа могут получать одинаковые сообщения.**

Объектно-ориентированный подход — это метод построения модульных систем с простой и децентрализованной структурой (даже для больших систем).

Основными принципами ООП являются:

- инкапсуляция
- наследование
- полиморфизм

Основной единицей в ООП является **программный объект**, который объединяет в себе как описывающие его данные (свойства), так и средства обработки этих данных (методы).

Программные объекты обладают *свойствами*, могут использовать *методы* и реагируют на *события*.

Классы объектов. *Классы объектов* являются «шаблонами», определяющими наборы свойств, методов и событий. По этим шаблонам создаются объекты. Таким образом, класс является моделью ещё не существующей сущности, т.н. объекта.

Каждый из классов обладает специфическим набором свойств, методов и событий.

Экземпляры класса. Объект, созданный по шаблону класса объектов, является *экземпляром класса* и наследует весь набор свойств, методов и событий данного класса. Каждый экземпляр класса имеет уникальное для данного класса имя. Различные экземпляры класса обладают одинаковым набором свойств, однако значения этих свойств у них могут отличаться. Таким образом, объект — это сущность, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции исходного кода на выполнение).

Свойства объектов. Каждый *объект* обладает определенным набором свойств, первоначальные значения которых можно задать, а затем при необходимости изменять в программном коде.

Методы объектов. Для того чтобы объект выполнил какую-либо операцию, необходимо применить метод, которым он обладает. Многие методы имеют аргументы, которые позволяют задать параметры выполняемых действий. Таким образом, взаимодействие объектов в абсолютном большинстве случаев обеспечивается вызовом ими методов друг друга.

События. *Событие* представляет собой действие, распознаваемое объектом. Событие может создаваться пользователем (например, щелчок мышью или нажатие клавиши) или быть результатом воздействия других программных объектов. В качестве

реакции на событие вызывается определенная процедура, которая может изменять значения свойств объекта, вызывать его методы и т.д.

Инкапсуляция. *Инкапсуляция* - это механизм, который объединяет данные и программный код, манипулирующий этими данными, а также скрывает реализацию от пользователя, тем самым защищая и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе; в этом случае говорят, что создаётся так называемый "чёрный ящик". Когда коды и данные объединяются таким способом, создаётся класс. Таким образом, класс – это единица инкапсуляции при объектно-ориентированном программировании.

Классы и объекты

Чтобы обеспечить расширяемость и повторное использование, необходима создавать систему, состоящую из автономных модулей. **Единицей модульности при ООП является класс.** *Класс* – это пользовательский тип данных. *Экземпляр класса* (объект) – это структура данных, размещаемая в памяти компьютера и обрабатываемая программой. Можно сказать что класс - это модель, а объект - экземпляр такой модели. В качестве примера можно представить автомобильный завод, где конструкторы разрабатывают чертеж автомобиля (это и есть класс), а в сборочном цехе собирают по этому чертежу сами машины (экземпляры класса).

Ранее, до существования объектно-ориентированных подходов, концепция модуля и типа данных существовали независимо друг от друга. Под модулем в разных языках понимались различные подпрограммы, пакеты или библиотеки, состоящие в основном из процедур и функций. Наиболее замечательным свойством класса является одновременное использование обеих этих концепций в рамках единой конструкции. С одной стороны класс это модуль, а с другой класс это новый тип данных.

Наследование. *Наследование* - это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов. Применение иерархии классов делает управляемыми большие потоки информации. Например, подумайте об описании жилого дома. Дом - это часть общего класса, называемого строением. С другой стороны, строение - это часть более общего класса - конструкции, который является частью ещё более общего класса объектов, который можно назвать созданием рук человека. В каждом случае порождённый класс наследует все связанные с родителем качества и добавляет к ним свои собственные

определяющие характеристики. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающе его определяли. Однако при использовании наследования можно описать объект путём определения того общего класса (или классов), к которому он относится, с теми специальными чертами, которые делают объект уникальным. Наследование играет очень важную роль в ООП.

Наследование является наиболее значимой после классов возможностью ООП. Этот процесс создания новых классов, называемых *наследниками*, или *производными классами*, из уже существующих или базовых классов. Производный класс получает все возможности базового класса, но может также быть усовершенствован за счет добавления собственных. Базовый класс при этом остается неизменным (рис. 7).

- Возможно, что стрелка на рисунке показывает совершенно иное направление, чем вы предполагали. Если она показывает вниз, то это называется наследованием. Однако обычно она указывает вверх, от производного класса к базовому, и это называется *производный от*. Также это указывает на то, что класс-наследник «знает» о существовании класса-родителя, в то время как базовому классу ничего не известно о его потомках.
- Наследование — важная часть ООП. Выигрыш от него состоит в том, что наследование позволяет использовать существующий программный код несколько раз. Имея написанный и отлаженный базовый класс, мы можем его больше не модифицировать, при этом механизм наследования позволит нам приспособить его для работы в различных ситуациях. Используя уже написанный код, мы экономим ресурсы, а так же увеличиваем надежность программы. Наследование может помочь и при начальной постановке задачи программирования, разработке общей структуры программы.

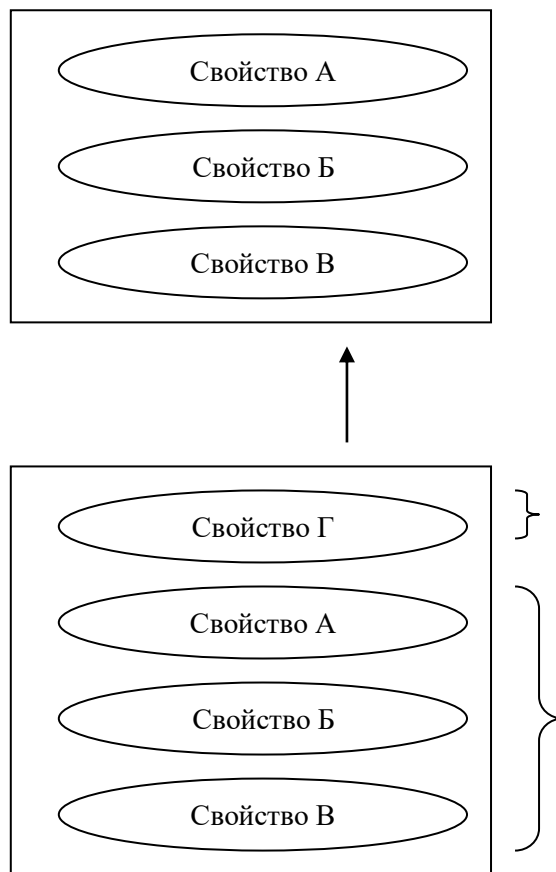


Рис. 7. Взаимосвязь классов при наследовании

Полиморфизм. *Полиморфизм* - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма применительно к ООП является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор. При этом программисту не нужно делать этот выбор самому. Нужно только помнить и использовать общий интерфейс.

Ключевым в понимании полиморфизма является то, что он позволяет манипулировать объектами различной степени сложности путём создания общего для них стандартного интерфейса для реализации похожих действий.

Лабораторная работа №5

Методы

Теоретическое введение. Метод – это последовательность инструкций (операторов) для решения какой-либо более или менее самостоятельной задачи (подзадачи), оформленная специальным образом. Первое знакомство со стандартными методами у вас уже состоялось при использовании метода `Main`, а также некоторых методов класса `Console` (`ReadLine()`, `WriteLine()` и др.), класса `Math` (`Abs()`, `Sin()` и др.) и др.

Сейчас мы рассмотрим методы более последовательно и подробно и научимся создавать их самостоятельно.

Фактически, в языке `C#` все алгоритмы обработки данных являются методами (например, все предыдущие программы создавались в рамках метода `Main`).

Методы в некотором смысле можно рассматривать в качестве аналогов подпрограмм (процедур и функций) в языках процедурного программирования, однако там данные и подпрограммы их обработки формально не связаны. В ООП же методы непосредственно связаны с объектом и определяют действия, которые можно выполнять над объектом или которые сам объект может выполнять.

Использование методов:

- улучшает структуру программы и облегчает ее восприятие;
- облегчает отладку, так как каждый метод может отлаживаться отдельно;
- при решении некоторой подзадачи несколько раз в рамках одной программы позволяет оформлять алгоритм один раз, а обращаться к нему многократно;
- уменьшает объем программы, если один и тот же метод выполняется несколько раз. Время выполнения при этом практически не изменяется.

Метод обычно реализует алгоритм в достаточно общем виде с использованием *формальных параметров* – абстрактных обозначений величин, участвующих в описании алгоритма. Каждый метод имеет свое имя, по которому осуществляется обращение к нему. При вызове метода формальные параметры заменяются на *фактические* (конкретизирующие задачу), с которыми метод и выполняется.

Формальным параметром может быть: переменная, массив, а также другой метод (типа делегат). Перед именем формального параметра указывается его тип (`int`, `double`, `string` и т. д.). Если формальным параметром является массив, то за типом

массива указывается квадратные скобки (для одномерного массива), либо в квадратных скобках ставится , (запятая) (для двумерного массива). Формальные параметры показывают, какие операции и над какими данными необходимо выполнять для реализации алгоритма. Фактическим параметром может быть литерал, переменная, выражение, массив, метод. Фактические параметры содержат данные, над которыми выполняются операции. Фактические параметры должны соответствовать формальным по количеству, типу и порядку следования.

Формальные параметры бывают входными и выходными. Через входные параметры передаются данные для вычисления алгоритма, выходные параметры содержат результат выполнения алгоритма (метода). Ключевое слово `out` или `ref` перед выходным параметром означает передачу параметра по ссылке, т.е. при обращении к методу на место выходного параметра передается адрес аргумента, фигурирующего в обращении к методу.

Переменные, не являющиеся ни входными, ни выходными аргументами, являются внутренними переменными метода и в списке параметров не указываются.

Алгоритм выполняется лишь при вызове метода.

З а м е ч а н и е . Взаимное расположение методов в пределах одного класса не имеет значения. Управление всегда вначале передается методу `Main`.

Общие положения. Способы передачи параметров

Методы могут возвращать значение (аналогично подпрограммам-функциям) и не возвращать значение (аналогично подпрограммам-процедурам).

Рассмотрим сначала методы, возвращающие значение.

Пример 5.1. Рассмотрим простейший пример.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 2, c = 3;
            Console.WriteLine(p(a, c));
        }
        static int p(int a1, int c1)
        {
            int s = a1 + c1;
```

```

        return s;
    }
}

```

В примере 5.1 в классе `Program` определены методы `Main` и `p`. В методе `p` вычисляется сумма двух переменных целого типа. Метод возвращает одно значение `s`, которое вычисляется в этом методе. Если метод возвращает значение, то имя переменной, в которую помещается возвращаемое значение, указывается после ключевого слова `return`, присутствие которого в данном случае обязательно. Если метод возвращает значение, то необходимо указывать тип возвращаемого значения, в данном примере `int`. Модификатор `static` обсудим позже¹. Заголовок метода включает также тип возвращаемого значения (если метод возвращает значение), имя метода и в скобках формальные параметры метода с указанием их типов. Все вместе эти элементы образуют *подпись (сигнатуру) метода*.

Вызов метода, возвращающего значение, осуществляется указанием имени метода и в скобках фактических параметров метода, которые заменяют формальные параметры метода перед его выполнением.

Обращение к методу записывается в том месте кода, где требуется получить значение, возвращаемое методом. В нашем примере обращение к методу `p`: `p(a, c)` записано в операторе вывода. Аргументы, указываемые при вызове метода, должны иметь тот же тип, что и параметры метода в описании метода и должны получить значения к моменту обращения к методу.

Как уже отмечалось, `C#` имеет две разновидности типов: типы значений и ссылочные типы (см. Часть 1). Переменные, основанные на типах значения, содержат непосредственно значения. В приведенном примере методу `p` передаются переменные `a` и `c`. Обе переменные являются экземплярами структуры `int` и относятся к типу значения. В данном примере переменные типа значения передаются методу с помощью значения. Использована так называемая *передача параметров по значению*. Любые изменения параметра, выполняемые внутри метода, не влияют на исходные данные, хранимые в переменных.

Метод получает собственные копии объектов значимого типа, не доступные никому кроме него, а исходные экземпляры объектов остаются неизменными. Вызывающий

¹ OO языки обычно разрешают создавать методы (и данные), относящиеся к классу целиком, а не к отдельным объектам. Методы (и данные) класса отмечаются ключевым словом `static`.

код должен выделить память, а вызванный метод управляет выделенной памятью. Создаются переменные для параметров метода, в которые пересылаются (копируются) соответствующие аргументы.

Так в данном примере при обращении к методу *p* в ячейку для параметра *a1* пересылается значение *a* т.е. 2, в ячейку для параметра *c1* – значение *c*, т.е. 3, и переменная *s* получает значение 5, которое и выводится в окно экрана.

Рассмотрим второй вариант решения той же задачи.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main()
        {
            int a = 2, c = 3, x;
            ps(a, c, out x);
            Console.WriteLine(x);
        }
        static void ps(int a1, int c1, out int s)
        {
            s = a1 + c1;
        }
    }
}
```

Те же вычисления выполняются в методе, не возвращающем значения. Если метод не возвращает значения, он имеет тип `void`. В списке параметров метода перечислены входные (*a1*, *c1*) и выходной (*s*) параметры. Ключевое слово `out` перед выходным параметром, имеющим тип значения, позволяет методу манипулировать единственным экземпляром типа значения (*передача параметра по ссылке*). Генерируется код, передающий вместо параметра его адрес.

В примере передается не значение переменной *x*, а адрес переменной *x*. Параметр *s* не является типом `int`; он является ссылкой на тип `int`, в данном случае ссылкой на переменную *x*. Поэтому после вызова метода значение переменной *x* изменяется.

Если использовать в этом примере передачу по значению, то результат вычисления суммы не будет передан в вызывающий метод и значение *x* не изменится. *Выходные*

параметры метода, имеющие тип значения, всегда должны передаваться по ссылке. В нашем примере в этом случае будет вообще выдано сообщение об ошибке, т. к. переменная x не инициализирована.

Для передачи по ссылке можно использовать также ключевое слово `ref`, но в этом случае аргумент, передаваемый по ссылке, должен быть инициализирован до обращения к методу (при использовании ключевого слова `out` это необязательно). Ниже приводится вариант программы с использованием `ref`.

Третий вариант

```
using System;
class Program
{
    static void Main()
    {
        int a = 2, c = 3, x = 0;
        //x присвоено фиктивное значение 0
        p(a, c, ref x);
        Console.WriteLine(x);
    }
    static void p(int a1, int c1, ref int s)
    {
        s = a1 + c1;
    }
}
```

Пример 5.2. Вычислить следующие суммы:

$$S_1 = 2^1 + 2^2 + \dots + 2^n,$$

$$S_2 = 3^1 + 3^2 + \dots + 3^m,$$

$$S_3 = 5^1 + 5^2 + \dots + 5^k,$$

$$S_4 = (\ln x)^1 + (\ln x)^2 + \dots + (\ln x)^k.$$

Суммирование ряда оформим в виде метода.

```
using System;
namespace ConsoleApplication1
{
```

```

class Program

static void Main(string[] args)
{
    double S1, S2, S3, S4, x, z;
    int n, m, k;
    Console.WriteLine("Введите количество членов ряда");
    n = int.Parse(Console.ReadLine());
    m = int.Parse(Console.ReadLine());
    k = int.Parse(Console.ReadLine());
    x = double.Parse(Console.ReadLine());
    Sum(2.0, n, out S1);
    Sum(3.0, m, out S2);
    Sum(5.0, k, out S3);
    z = Math.Log(x);
    Sum(z, k, out S4);
    Console.WriteLine(" 2+2^2+...+2^n= {0:f3}", S1);
    Console.WriteLine(" 3+3^2+...+3^m= {0:f3}", S2);
    Console.WriteLine(" 5+5^2+...+5^k= {0:f3}", S3);
    Console.WriteLine("lnx+(lnx)^2+...+(lnx)^k= {0:f3}", S4);
}

// метод суммирования
static void Sum(double a, int t, out double s)
{
    int i;
    double p;
    s = 0; p = 1;
    for (i = 1; i <= t; i++)
    {
        p = p * a; s = s + p;
    }
}
}

```

Пример 5.3. Произведено n выстрелов. Вероятность попадания в цель одного выстрела 0,6.

Составить таблицу для вычисления вероятности для 1, 2, ..., n выстрелов.

Вероятность $P_n(m)$ того, что при n испытаниях событие A наступит m раз, вычисляется по формуле Бернулли:

$$P_n(m) = \frac{n!}{m!(n-m)!} p^m (1-p)^{n-m}$$

Вычисление $k!$ оформим в виде метода.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int n, m;
            double Ver, Vn, Vm, Vnm, p;
            Console.WriteLine("Введите число выстрелов");
            n = int.Parse(Console.ReadLine());
            p = 0.6;
            Console.WriteLine("Выстрел Вероятность");
            for(m = 1; m <= n; m++)
            {
                Fact(n, out Vn);
                Fact(m, out Vm);
                Fact(n - m, out Vnm)
                Ver = Vn * Math.Pow(p, m) * Math.Pow(1 - p, n - m) / (Vm * Vnm);
                Console.WriteLine("    {0:d2}        {1:f4}", m, Ver);
            }
        }
    }
}

// Вычисление факториала
static void Fact(int k, out double fk)
{
    int i;
```

```

        fk = 1;
        for(i =1 ; i <= k; i++)
            fk = fk * i;
    }
}
}

```

Пример 5.4. Траектория снаряда, вылетающего из орудия под углом α с начальной скоростью v_0 , описывается уравнением:

$$y(x) = x \cdot \operatorname{tg} \alpha - \frac{g \cdot x^2}{2v_0^2 \cos^2 \alpha}$$

С точностью $\Delta x = 0,5$ м определить точку, в которой снаряд упадет на землю.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double v0, alfa, x, yx;
            Console.WriteLine("Введите скорость и угол наклона
орудия");
            v0 = double.Parse(Console.ReadLine());
            alfa = double.Parse(Console.ReadLine());
            alfa = alfa * Math.PI / 180;
            x = 0;
            do
            {
                x = x + 0.5;
                T(alfa, v0, x, out yx);
            } while (yx > 0);
            Console.WriteLine(" {0:f1}", x);
        }
    }
}

// Вычисление высоты снаряда
static void T (double alfa, double v0, double x, out double yx)

```

```

        {
            yx = x * Math.Tan(alfa) - 9.81 * x * x / (Math.Cos(alfa) *
v0 * Math.Cos(alfa) * v0);
        }
    }
}

```

Пример 5.5. Разработать (определить) метод для решения квадратного уравнения $ax^2 + bx + c = 0$. Вызвать метод для решения уравнения $3,2t^2 + 4,6t - 5 = 0$ и, если уравнение имеет решение, вывести на печать больший из корней.

Вариант 1. Метод `root` статический, методы `root` и `Main` в одном классе. Передача параметров с помощью `ref`.

```

using System;
class Program
{
    static void root(double a, double b, double c, ref double
x1, ref double x2, ref bool l)
    {
        double d;
        d = b * b - 4 * a * c;
        if (d >= 0)
        {
            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            l = !l;
        }
    }
    static void Main()
    {
        bool q = false;
        double t1 = 0, t2 = 0, u = 0;
        root(3.2, 4.6, -5.0, ref t1, ref t2, ref q);
        if (q)
        {
            u = t1;

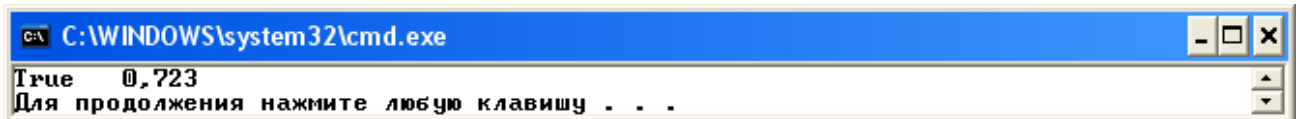
```

```

        if (t2 > u) u = t2;
        Console.WriteLine("{0}    {1:f3}", q, u);
    }
else
    Console.WriteLine("решения нет");

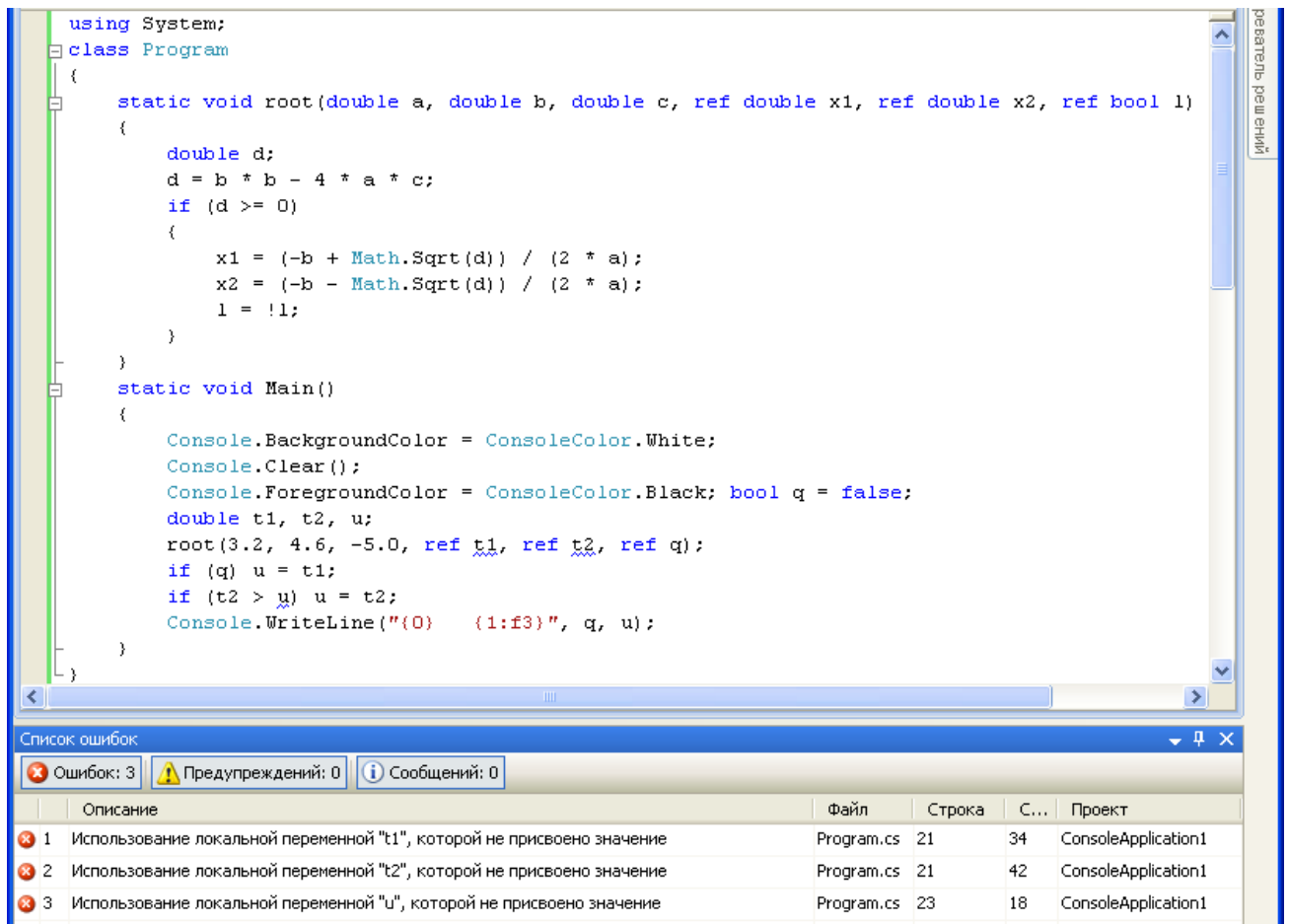
}
}

```



З а м е ч а н и я . 1. Если два метода находятся в одном и том же классе и один из них статический (метод Main статический по определению) и из него вызывается другой метод, то вызываемый метод должен быть статическим. Существует только одна копия статического метода. Ниже приводятся варианты (вариант 3 и вариант 4) решения этой же задачи, где использован нестатический метод (instance – экземплярный). Метод связан с экземпляром типа, а не с типом как таковым. Метод может обращаться к экземплярным полям и методам, а также к статическим полям и методам. *(Статический метод не применяется к конкретному объекту структуры, а относится ко всей структуре, и доступ к нему осуществляется по имени структуры, а не экземпляра (объекта) структуры.)*

2. В случае использования ref переменные должны быть инициализированы до вызова. Ниже приводится листинг кода с выведенными сообщениями об ошибках, вызванных отсутствием инициализации аргументов метода, передаваемых по ссылке.



В случае использования `out` можно не инициализировать переменные в основной программе, а инициализировать в вызываемом методе. В некоторых случаях инициализация необязательна, см. например второй вариант первого примера. В данном случае инициализация обязательна, т. к. эти переменные используются в условном операторе. То же касается использования таких переменных в циклах, конструкторах и т. п.

Вариант 2. Метод `root` статический, методы `root` и `Main` в одном классе. Передача параметров с помощью `out`. В вариантах 2, 3, 4 при отсутствии решения уравнения соответствующее сообщение не выводится.

```

using System;
class Program
{
    static void root(double a, double b, double c, out double
x1,out double x2, out bool l)
    {

```



```

        double d;
        l = false;
        d = b * b - 4 * a * c;
        x1 = 0;
        x2 = 0;
        if (d >= 0)
        {
            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            l = !l;
        }
    }
    static void Main()
    {
        bool q;
        double t1, t2, u = 0;
        root(3.2, 4.6, -5.0, out t1, out t2, out q);
        if (q) u = t1;
        if (t2 > u) u = t2;
        Console.WriteLine("{0}    {1:f3}", q, u);
    }
}

```

Вариант 3. Метод `root` не статический и метод `Main` (статический) в одном классе `Program`.

```

using System;
class Program
{
    void root(double a, double b, double c, ref double x1, ref
double x2, ref bool l)
    {
        double d;
        d = b * b - 4 * a * c;
        if (d >= 0)

```

```

        {
            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            l = !l;
        }
    }

    static void Main()
    {
        bool q = false;
        double t1 = 0, t2 = 0, u = 0;
        //создается экземпляр класса, т.е. pr - переменная типа Program
        Program pr = new Program();
        //метод root экземпляра pr класса Program
        pr.root(3.2, 4.6, -5.0, ref t1, ref t2, ref q);
        if (q) u = t1;
        if (t2 > u) u = t2;
        Console.WriteLine("{0}    {1:f3}", q, u);
    }
}

```

Вариант 4. Методы root и Main в разных классах

```

using System;

class test
{
    public void root (double a, double b, double c, ref double
x1, ref double x2, ref bool l)
    {
        double d;
        d = b * b - 4 * a * c;
        if (d >= 0)
        {
            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            l = !l;
        }
    }
}

```

```

        }
    }
}
class Program
{
    static void Main ()
    {
        bool q = false;
        double t1 = 0, t2 = 0, u = 0;
        test pr = new test(); // pr - экземпляр класса test
        //вызывается метод root экземпляра pr класса test
        pr.root(3.2, 4.6, -5.0, ref t1, ref t2, ref q);
        if (q) u = t1;
        if (t2 > u) u = t2;
        Console.WriteLine("{0}    {1:f3}", q, u);
    }
}

```

В последнем варианте метод `root` имеет модификатор доступа `public` (открытый), обеспечивающий доступ к этому методу из другого класса.

Пример 5.6. Вычислить число сочетаний из n по m : $C = n!/(m!(n-m)!)$.

Вычисление факториала оформить в виде метода, возвращающего значение (ср. с примером 5.3).

Вариант 1. Методы `Main` и `fact` для вычисления факториала в разных классах.

```

using System;
class test
{
    public int fact (int n)
    {
        int f = 1;
        for (int i = 2; i <= n; i++)
        {
            f = f * i;
        }
        return f;
    }
}

```

```

}
class Program
{
    static void Main()
    {
        int n = 4, m = 3 ;
        int cnm;
        test pr = new test();
        cnm = pr.fact(n) / (pr.fact(m)*pr.fact(n - m));
        Console.WriteLine("{0}",cnm);
    }
}

```

Вариант 2. Методы Main и fact для вычисления факториала в одном классе.

```

using System;
class Program
{
    static int fact(int n)
    {
        int f = 1;
        for (int i = 2; i <= n; i++)
        {
            f = f * i;
        }
        return f;
    }
    static void Main()
    {
        int n = 4, m = 3 ;
        int cnm;
        cnm = fact(n)/ (fact(m) * fact(n - m));
        Console.WriteLine("{0}",cnm);
    }
}

```

Использование массивов в качестве параметров

Массив является ссылочным типом и если массив является параметром метода, то передача его всегда осуществляется по ссылке, независимо от наличия ключевого слова `ref`. Поскольку массивы являются ссылочными типами, метод может изменять значения элементов. Наличие квадратных скобок после типа в подписи метода означает, что параметром является массив.

Пример 5.7. Вычислить угол φ между двумя векторами: $X=\{x_1, x_2, \dots, x_n\}$ и $Y=\{y_1, y_2, \dots, y_n\}$.

Косинус угла между векторами вычисляется по формуле:

$$z = \cos \varphi = \frac{x_1 \cdot y_1 + x_2 \cdot y_2 + \dots + x_n \cdot y_n}{\sqrt{(x_1^2 + x_2^2 + \dots + x_n^2)(y_1^2 + y_2^2 + \dots + y_n^2)}},$$
$$\varphi = \arctg \frac{z}{\sqrt{1 - z^2}}.$$

```
// Вычисление скалярного произведения
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Skalr(double[] x, double[] y, out double sk)
        {
            int i;
            sk = 0;
            for (i = 0; i < x.Length; i++)
                sk += x[i] * y[i];
        }

        // Вычисление угла между векторами
        static void Main(string[] args)
        {
            int i, n;
            double Fi, ab, a2, b2;
            string sb;
```

```

        Console.WriteLine("Введите число элементов массивов");
        n = int.Parse(Console.ReadLine());
        double[] a = new double[n];
        double[] b = new double[n];
        Console.WriteLine("Введите элементы массивов");
        for (i = 0; i < n; i++)
        {
            sb = Console.ReadLine();
            a[i] = double.Parse(sb);
        }
        for (i = 0; i < n; i++)
        {
            sb = Console.ReadLine();
            b[i] = double.Parse(sb);
        }
        Skalr(a, b, out ab);
        Skalr(a, a, out a2);
        Skalr(b, b, out b2);
        Fi = Math.Acos(ab/Math.Sqrt(a2 * b2));
        Console.WriteLine("Arccos = {0:f4}", Fi);
    }
}

```

Пример 5.8. В массив B размера 4×5 вставить после строки, содержащей максимальное количество положительных элементов, столбец массива C размера 5×6 , содержащий максимальное количество положительных элементов. Определение количества положительных элементов в заданной строке (или столбце) матрицы осуществить в методе.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {

```

```

int i, j, k = 0, f=0, f1=0, max, max1;
double[,] B = new double[5, 5]
{{5,-4,-3, 1,-7},
 {-2,-8, 7,-5,-9},
 {-1,-3,-5, 6,-7},
 {8,-9,-5,-4,-2},
 {0, 0, 0, 0, 0}};
double[,] C = new double[5, 6]
{{4,-4,-7, 1,2,0},
 {-2, 8, 7,-5, 9, 1},
 {-6,-3,-5, 6,-7,-4},
 {8, 9,-5,-4, 2, 3},
 {10, 19,-5,-5, 2,-1}};
Console.WriteLine("Матрица B");
for (i = 0; i < 4; i++)
{
    for (j = 0; j < 5; j++)
        Console.Write("{0,4:f0}", B[i, j]);
    Console.WriteLine();
}
Console.WriteLine("Матрица C");
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 6; j++)
        Console.Write("{0,4:f0}", C[i, j]);
    Console.WriteLine();
}
max = 0;
for (j = 0; j < 4; j++)
{
    Count(B, 5, 0, j, out k);
    if (max < k)
    {
        max = k;
        f = j;
    }
}

```

```

    }

    for (i = 4; i > f; i--)
        for (j = 0; j < 5; j++)
            B[i, j] = B[i - 1, j];
    max1 = 0;
    for (j = 0; j < 6; j++)
    {
        Count(C, 5, 1, j, out k);
        if (max1 < k)
        {
            max1 = k;
            f1 = j;
        }
    }
    for (j = 0; j < 5; j++)
        B[f+1, j] = C[j, f1];

    Console.WriteLine("Матрица B с включенным столбцом
матрицы C");
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 5; j++)
            Console.Write("{0,4:f0}", B[i, j]);
        Console.WriteLine();
    }
}

static void Count(double[,] a, int n, int prs, int j,
out int k)
{
    int i;
    double l;
    k = 0;
    for (i = 0; i < n; i++)
    {
        if (prs == 0) l = a[j, i]; else l = a[i, j];
    }
}

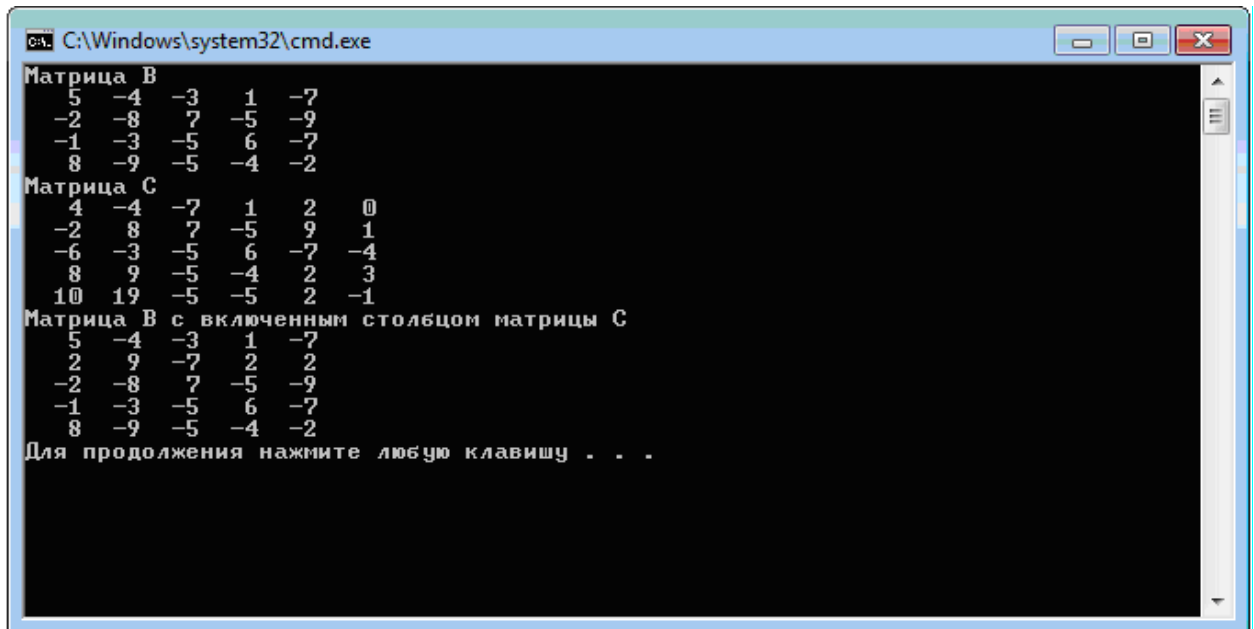
```



```

        if (l > 0) k++;
    }
}
}
}

```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the following text:

```

Матрица В
5 -4 -3 1 -7
-2 -8 7 -5 -9
-1 -3 -5 6 -7
8 -9 -5 -4 -2
Матрица С
4 -4 -7 1 2 0
-2 8 7 -5 9 1
-6 -3 -5 6 -7 -4
8 9 -5 -4 2 3
10 19 -5 -5 2 -1
Матрица В с включенным столбцом матрицы С
5 -4 -3 1 -7
2 9 -7 2 2
-2 -8 7 -5 -9
-1 -3 -5 6 -7
8 -9 -5 -4 -2
Для продолжения нажмите любую клавишу . . .

```

Пример 5.9. Элементы первого столбца заданной матрицы a размера $n \times m$ упорядочены по убыванию. Включить заданный массив b размера m в качестве новой строки в массив a с сохранением упорядоченности первого столбца. Включение массива оформить в виде метода.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j, n,m;
            Console.WriteLine("Введите размеры матрицы");
            n = int.Parse(Console.ReadLine());
            m = int.Parse(Console.ReadLine());
            double[,] a = new double[n+1, m] ;
            double[] b = new double[m] ;
            Console.WriteLine("Введите элементы матрицы");
            for (i = 0; i < n; i++)
                for (j = 0; j < m; j++)
                    a[i, j] = double.Parse(Console.ReadLine());
            Console.WriteLine("Введите элементы вектора");
            for (i = 0; i < m; i++)
                b[i] = double.Parse(Console.ReadLine());
            for (i = 0; i < n; i++)
            {
                for (j = 0; j < m; j++)
                    Console.Write(" {0,5:f1} ", a[i, j]);
                Console.WriteLine();
            }
            Console.WriteLine();
            for (j = 0; j < m; j++)
                Console.Write(" {0,5:f1} ", b[j]);
```

```

        Console.WriteLine();
        Console.WriteLine();
        MatVec(a, b, n, m); n++;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < m; j++)
                Console.Write(" {0,5:f1} ", a[i, j]);
            Console.WriteLine();
        }
    }

    // Включение вектора в качестве строки матрицы
    static void MatVec(double[,] a, double[] b, int n, int m)
    {
        int i, j, k = n;
        for (i = 0; i < n; i++)
        {
            if (a[i, 0] < b[0])
            {
                k = i; break;
            }
        }
        for (i = n-1; i >= k; i--)
            for (j = 0; j < m; j++)
                a[i + 1, j] = a[i, j];
        for (i = 0; i < m; i++)
            a[k, i] = b[i];
    }
}

```

Пример 5.10. Максимальный элемент массива *a* размера 6 поменять местами с максимальным элементом массива *b* размера 8. Оформить метод для поиска максимального элемента одномерного массива. Результатом метода являются значение и индекс максимального элемента.

```

using System;
class Program
{
    static void maxx(int[] x, ref int xmax, ref int imax)
    {
        xmax = x[0];
        imax = 0;
        for (int i = 0; i < x.Length; i++)
        {
            if (x[i] >= xmax)
            {
                xmax = x[i];
                imax = i;
            }
        }
    }
    static void Main()
    {
        int[] a = new int[] {1, 7, 3, 5, 6, 2};
        int[] b = new int[] {4, 2, 1, 8, 9, 3, 5, 6};
        for (int i = 0; i < a.Length; i++)
        {
            Console.Write("{0:d} ", a[i]);
        }
        Console.WriteLine();
        for (int i = 0; i < b.Length; i++)
        {
            Console.Write("{0:d} ", b[i]);
        }
        Console.WriteLine();
        Console.WriteLine();
        int amax = 0, bmax = 0, iamax = 0, ibmax = 0;
        maxx(a, ref amax, ref iamax);
        maxx(b, ref bmax, ref ibmax);
        a[iamax] = bmax; b[ibmax] = amax;
        for (int i = 0; i < a.Length; i++)
    }
}

```

```

    {
        Console.Write("{0:d}  ", a[i]);
    }
    Console.WriteLine();
    for (int i = 0; i < b.Length; i++)
    {
        Console.Write("{0:d}  ", b[i]);
    }
    Console.WriteLine();
}
}

```

В методе `maxx` предусмотрена возможность обработки массивов разного размера. Для определения размера массива-аргумента, обрабатываемого в каждом конкретном случае, используется свойство массива `Length`.

В методе `Main` для указания верхней границы индексов в циклах также используется свойство `Length`, (хотя можно было бы использовать и различные константы).

```

C:\WINDOWS\system32\cmd.exe
1 7 3 5 6 2
4 2 1 8 9 3 5 6
1 7 3 5 6 2
4 2 1 8 9 3 5 6
Для продолжения нажмите любую клавишу . . .

```

Пример 5.11. Просуммировать элементы строки матрицы d размера $n \times m$ (программа реализована для $n = 3$, $m = 4$), содержащей максимальный элемент матрицы. Поиск максимального элемента осуществлять в методе.

Исходный массив d представим в виде одномерной последовательности с нумерацией индексов от 0 до $n \cdot m - 1$ (в данном примере $n \cdot m - 1 = 11$). Возвращаемым значением метода является номер строки, содержащей максимальный элемент матрицы. После нахождения максимального элемента матрицы необходимо просуммировать следующие подряд m элементов, начиная с первого элемента в строке (это элемент с номером $ns \cdot m$ (ns - номер строки, содержащей максимальный элемент матрицы, строки нумеруются от 0 до $n - 1$)).

Результатом работы метода является одно значение – номер строки, содержащей максимальный элемент матрицы. Возможны два варианта организации метода: получить результат как возвращаемое методом значение (вариант 1) или использовать для этого параметр метода (вариант 2). Далее приводятся оба варианта кода.

Вариант 1. Метод возвращает значение.

```
using System;
class Program
{
    const int m = 4;
    static int maxx(int[] x)
    {
        int xmax = x[0];
        int imax = 0;
        for (int i = 0; i < x.Length; i++)
        {
            if (x[i] > xmax)
            {
                xmax = x[i];
                imax = i;
            }
        }
        return imax / m;
    }
    static void Main()
    {
        int[] d = new int[] {1, 7, 3, 5, 6, 2, 9, 3, 8, 3, 6, 1};

        for (int i = 0; i < d.Length; i++)
        {
            Console.Write("{0:d}  ", d[i]);
        }
        Console.WriteLine();
        int s = 0;
        int ns = maxx(d) * m;
        for (int k = ns; k < ns + m; k++)
            s = s + d[k];
        Console.WriteLine(s);
    }
}
```

Вариант 2. Для получения результата используется параметр метода.

```
using System;
class Program
{
    const int m = 4;
    static void maxx(int[] x, ref int di)
    {
        int xmax = x[0];
        int imax = 0;
        for (int i = 0; i < x.Length; i++)
        {
            if (x[i] > xmax)
            {
                xmax = x[i];
                imax = i;
            }
        }
        di = imax / m;
    }
    static void Main()
    {
        int[] d = new int[] {1, 7, 3, 5, 6, 2, 9, 3, 8, 35, 6, 1};
        int di = 0;
        for (int i = 0; i < d.Length; i++)
        {
            Console.Write("{0:d}  ", d[i]);
        }
        Console.WriteLine();
        int s = 0;
        maxx(d, ref di);
        int ns = di * m;
        for (int k = ns; k < ns + m; k++)
            s = s + d[k];
        Console.WriteLine(s);
    }
}
```

}

Использование делегата для передачи метода в качестве параметра в другой метод

До сих пор в качестве параметров методов использовались простые переменные или массивы. Но параметром метода может быть также и другой метод. В качестве наглядного примера рассмотрим следующую задачу. Пусть требуется вычислить две суммы

$$S1 = \sum_{i=1}^{12} i^2,$$

$$S2 = \sum_{i=1}^7 i^3.$$

Вычисление суммы в методе оформим в общем виде, не конкретизируя зависимость члена суммы от его номера, т.е. оформим функцию для вычисления суммы

$$S = \sum_{i=1}^n f(i).$$

Функция f должна быть включена в список формальных параметров как *переменная типа делегат*, которая при каждом обращении должна быть заменена фактическим параметром, т.е. именем метода, описывающего вычисление конкретного члена суммы.

Делегат могут вызывать только такие методы, у которых тип возвращаемого значения и список параметров совпадают с соответствующими элементами объявления делегата.

Ключевое слово `delegate` используется для объявления ссылочного типа. Делегат - это тип, который определяет подпись метода и его можно связать с любым методом с совместимой подписью.

Пример 5.12. В качестве примера приводится программа для решения рассмотренной выше задачи вычисления двух сумм.

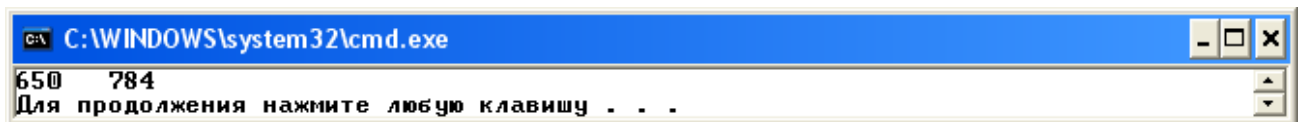
```
using System;
class Program
{
    delegate int fi(int i);
    static int f1(int i)
    {
        return i * i;
    }
}
```



```

static int f2(int i)
{
    return i * i * i;
}
static int si(fi f, int n)
{
    int s = 0;
    for (int i = 0; i <= n; i++)
    {
        s = s + f(i);
    }
    return s;
}
static void Main()
{
    int s1 = si(f1,12);
    int s2 = si(f2,7);
    Console.WriteLine("{0}    {1}", s1, s2);
}
}

```



Следующие примеры демонстрирует использование метода в качестве параметра, а также использование перечисления для вычисления определенного интеграла двумя различными методами.

Пример 5.13. Оформить функцию для вычисления $S =$. Используя эту функцию,

вычислить $S1 = \int_0^{\pi/2} \sin^2 x dx$ и $S2 = \int_1^2 \frac{1}{\sqrt{9+x^2}} dx$. Для вычисления определенного интеграла

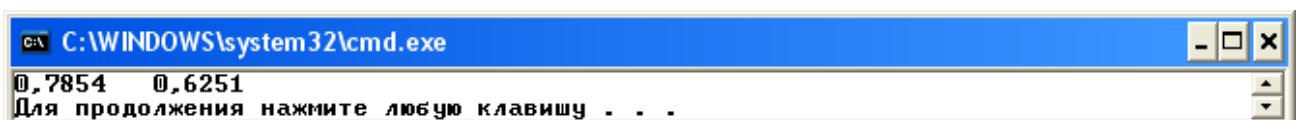
использовать метод трапеций. Разделим отрезок $[a, b]$ на n отрезков длиной $h = (b - a)/n$.

Формула трапеций для вычисления интеграла в этом случае имеет вид

$$\frac{h}{2}(f(a)+f(b)+2 \quad),$$

где $x_2 = a + h$, $x_3 = a + 2h$,

```
using System;
class Program
{
    delegate double fx(double i);
    static double f1(double x)
    {
        return Math.Sin(x)*Math.Sin(x);
    }
    static double f2(double x)
    {
        return 1/Math.Sqrt(9 + x * x);
    }
    static double sw(fx f, double a, double b, int n)
    {
        double c = 0, x = a, h = (b - a)/n;
        for (int i = 2; i <= n; i++)
        {
            x += h; c += f(x);
        }
        return (2 * c + f(a) + f(b)) * h / 2;
    }
    static void Main()
    {
        double s1 = sw(f1, 0.0, Math.PI/2, 40);
        double s2 = sw(f2, 0.0, 2.0, 30);
        Console.WriteLine("{0:f4}    {1:f4}", s1, s2);
    }
}
```



Пример 5.14. Пусть требуется найти решение трех уравнений:

$$\cos \frac{1}{x} - 2 \sin \frac{1}{x} + \frac{1}{x} = 0 \text{ на отрезке } [1, 2],$$

$$\sin(\ln x) - \cos(\ln x) + 2 \ln x = 0 \text{ на отрезке } [1, 3],$$

$$1 - x + \sin x - \ln(1 + x) = 0 \text{ на отрезке } [0, 2].$$

Нахождение корня уравнения $f(x) = 0$ на отрезке $[a, b]$ выполним методом *половинного деления*.

Делим отрезок пополам, т.е. выбираем $x = (a + b)/2$.

Если $f(x) = 0$, то x является корнем уравнения. Если $f(x) \neq 0$, то выбираем тот из отрезков $[a, x]$ или $[x, b]$, на концах которого функция $f(x)$ имеет противоположные знаки. Полученный отрезок снова делим пополам и проводим то же рассмотрение и т.д.

Процесс деления отрезков пополам продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше наперед заданного числа $\varepsilon = 10^{-5}$.

Функция f должна быть включена в список формальных параметров как *переменная типа делегат*, которая при каждом обращении должна быть заменена фактическим параметром, т.е. именем метода, описывающего решение конкретного уравнения.

Далее приводится программа для вычисления корней трех рассмотренных уравнений.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        delegate double fx(double i);
        static double f1(double x)
        {
            return Math.Cos(1 / x) - 2 * Math.Sin(1 / x) + 1 / x;
        }
        static double f2(double x)
        {

```

```

        return Math.Sin(Math.Log(x)) - Math.Cos(Math.Log(x))
+ 2 * Math.Log(x);
    }
    static double f3(double x)
    {
        return 1 - x + Math.Sin(x) - Math.Log(1 + x);
    }
    static void Poldel(fx f, double a, double b, out double
xs)
    {
        double xt;
        do
        {
            xt = (a + b) / 2;
            if (f(a) * f(xt) < 0)
                b = xt;
            else
                a = xt;
        } while (b - a > 1e-5);
        xs = xt;
    }
    static void Main(string[] args)
    {
        double x1, x2, x3;
        Poldel(f1, 1, 2, out x1);
        Poldel(f2, 1, 3, out x2);
        Poldel(f3, 0, 2, out x3);
        Console.WriteLine("Корень уравнения  $\cos(1/x) - 2 * \sin(1/x) + 1/x$  {0:f4} Проверка {1:f6}", x1, f1(x1));
        Console.WriteLine("Корень уравнения  $\sin(\log(x)) - \cos(\log(x)) + 2*\log(x)$  {0:f4} Проверка {1:f6}", x2, f2(x2));
        Console.WriteLine("Корень уравнения  $1 - x + \sin(x) - \log(1 + x)$  {0:f4} Проверка {1:f6}", x3, f3(x3));
    }
}

```

Вопросы для самопроверки

1. Что такое метод. Разновидности методов: метод, возвращающий значение, и метод, не возвращающий значения. Особенности их оформления.
2. Что такое подпись метода.
3. Вызов метода. Способы передачи параметров: по значению, по ссылке. Правила согласования формальных и фактических параметров при вызове метода.
4. Различные возможности взаимного расположения вызываемого и вызывающего методов: в одном классе, в разных классах.
5. Особенности вызова нестатического метода из статического метода.
6. Использование массивов в качестве параметров.
7. Методы как параметры: Использование делегата для передачи метода как параметра другого метода.

Задания I уровня

Задание I уровня предназначено для приобретения навыков разработки методов, возвращающих значение. Сформулировать, если нужно, задачу математически. Определить входные параметры метода и возвращаемое значение. Разработать метод таким образом, чтобы не портились входные параметры. Составить программу. Разработать тесты и проверить работу программы на компьютере.

1. Определить, сколькими способами можно отобрать команду в составе пяти человек из восьми кандидатов; из 10 кандидатов; из 11 кандидатов. Использовать метод для подсчета количества способов отбора по формуле

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

2. Два треугольника заданы длинами своих сторон a , b и c . Определить треугольник с большей площадью, вычисляя площади треугольников по формуле Герона

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где $p = (a + b + c)/2$.

3. Два велосипедиста одновременно начинают движение из одной точки. Первый начинает движение со скоростью 10 км/ч и равномерно увеличивает скорость на 1

км/ч. Второй начинает движение со скоростью 9 км/ч и равномерно увеличивает скорость на 1,6 км/ч. Определить:

а) какой спортсмен преодолеет большее расстояние через 1 ч, через 4 ч;

б) когда второй спортсмен догонит первого.

Использовать метод для вычисления пути в зависимости от времени по формуле

$$S = vt + at^2/2,$$

где v – начальная скорость;

a – ускорение.

Задания II уровня

Задание II уровня предназначено для приобретения навыков разработки методов с использованием массивов в качестве параметров. Определить параметры методов. Разработать методы. Составить программу. Проверить ее работу на компьютере.

1. Поменять местами максимальные элементы матриц A размера 5×6 и B размера 3×5 . Поиск максимального элемента матрицы осуществить в методе.
2. В массивах A размера 9 и B размера 7 заменить максимальные элементы на среднее арифметическое значение элементов, расположенных после максимального, в том массиве, для которого максимальный элемент расположен дальше от конца массива. Поиск максимального элемента осуществить в методе.
3. В матрицах B размера 5×5 и C размера 6×6 удалить строку, содержащую максимальный элемент на диагонали. Поиск максимального элемента диагонали осуществить в методе.
4. Поменять местами строку матрицы A размера 5×5 и столбец матрицы B размера 5×5 , содержащие максимальные элементы на диагоналях. Поиск максимального элемента на диагонали осуществить в методе.
5. Поменять местами строки матриц A размера 4×6 и B размера 6×6 , содержащие максимальные элементы в 1-м столбце. Поиск максимального элемента в заданном столбце матрицы осуществить в методе.
6. Объединить массивы A размера 7 и B размера 8, предварительно удалив максимальные элементы этих массивов. Результат получить в массиве A . Удаление элемента массива с заданным индексом осуществить в методе.

7. В массив B размера 4×5 вставить после строки, содержащей максимальное количество положительных элементов, столбец массива C размера 5×6 , содержащий максимальное количество положительных элементов. Определение количества положительных элементов в заданной строке (или столбце) матрицы осуществить в методе.
8. Упорядочить по возрастанию элементы массивов A размера 9 и B размера 11, расположенные после максимального элемента. Упорядочение части массива, начинающейся элементом с заданным индексом, осуществить в методе.
9. Даны матрицы A размера 6×5 и C размера 7×4 . Объединить массивы, сформированные из сумм положительных элементов столбцов матриц A и C . Суммирование положительных элементов столбцов с получением результата в виде массива осуществить в методе.
10. В заданной квадратной матрице найти максимальный элемент среди элементов, расположенных ниже главной диагонали (включая диагональ), и минимальный элемент среди элементов, расположенных выше главной диагонали. Удалить столбцы, в которых они находятся. Удаление столбца оформить в виде метода.
11. В двух заданных матрицах найти максимальные элементы и поменять их местами. Поиск максимального элемента матрицы оформить в виде метода.
12. Заданы две матрицы одинакового размера. Столбец первой матрицы, содержащий максимальный элемент матрицы, поменять местами со столбцом второй матрицы, содержащим максимальный элемент. Поиск столбца, содержащего максимальный элемент матрицы, оформить в виде метода.
13. В заданной матрице удалить строки, содержащие максимальный и минимальный элементы матрицы. Удаление строки оформить в виде метода.
14. В заданной матрице элементы каждой строки расположить в порядке возрастания. Упорядочение элементов строки оформить в виде метода.
15. Для каждой из трех заданных матриц найти среднее значение ее элементов без учета максимального и минимального элементов. Полученные значения занести в одномерный массив. Определить, образовали ли полученные значения убывающую или возрастающую последовательность. Нахождение среднего значения элементов матрицы оформить в виде метода.
16. В двух заданных одномерных массивах расположить отрицательные элементы в порядке убывания (оставив положительные элементы на прежних местах). Упорядочение отрицательных элементов массива оформить в виде метода.

17. В двух заданных матрицах расположить строки в порядке убывания их максимальных элементов. Упорядочение строк матрицы оформить в виде метода.
18. В двух заданных квадратных матрицах упорядочить элементы главной диагонали по возрастанию. Упорядочение диагональных элементов оформить в виде метода.
19. В заданной матрице удалить все строки, содержащие нулевые элементы. Удаление строки оформить в виде метода.
20. В двух заданных матрицах удалить все столбцы, не содержащие нулевых элементов. Удаление столбца матрицы оформить в виде метода.
21. Для двух заданных квадратных матриц составить одномерные массивы из минимальных элементов строк, расположенных правее элементов главной диагонали (включая диагональ). Формирование одномерного массива оформить в виде метода.
22. Для заданной матрицы сформировать два одномерных массива, состоящих из количества отрицательных элементов строк и максимальных среди отрицательных элементов столбцов. Использовать методы.
23. В двух заданных матрицах по пять наибольших элементов увеличить вдвое, остальные вдвое уменьшить. Преобразование матрицы оформить в виде метода.
24. В двух заданных квадратных матрицах столбец, содержащий максимальный элемент матрицы, поменять местами с главной диагональю. Поиск максимального элемента и дальнейшее преобразование матрицы оформить в виде методов.
25. В двух заданных матрицах найти строку, содержащую максимальное количество отрицательных элементов. Нахождение количества отрицательных элементов строк матрицы и поиск среди них максимального оформить в виде методов.
26. В двух заданных матрицах одинакового размера поменять строки, содержащие максимальное количество отрицательных элементов. Нахождение количества отрицательных элементов заданной строки матрицы осуществлять в методе. Определение номера строки, содержащей максимальное количество отрицательных элементов, осуществлять в методе.
27. В двух заданных матрицах максимальные элементы четных строк заменить нулями, нечетных — умножить на номер столбца, в котором они находятся. Использовать методы.
28. Две функции $y = f_1(x)$ и $y = f_2(x)$ заданы таблично на отрезке $[A, B]$.
 - а) определить, является ли каждая из этих функций монотонно убывающей или монотонно возрастающей на заданном отрезке. Проверку монотонности функции оформить в виде метода;

- б) найти все интервалы монотонности. Использовать метод;
- в) найти самый длинный интервал монотонности. Использовать метод.

Задания III уровня

Задание III уровня предназначено для приобретения навыков использования делегата в качестве параметра метода. Выполнить также все пункты задания II уровня

1. Вычислить суммы

$$s = 1 + \sum_{i=1}^{\infty} \frac{\cos(ix)}{i!}, y = e^{\cos x} \cos(\sin x); a = 0,1; b = 1; h = 0,1;$$

$$s = \sum_{i=1}^{\infty} (-1)^i \frac{\cos(ix)}{i^2}, y = \frac{x^2 - \pi^2/3}{4}; a = \pi/5; b = \pi; h = \pi/25.$$

Вычисление суммы осуществлять в методе. Для вычисления члена суммы использовать делегата.

2. В заданной матрице расположить элементы четных строк в порядке возрастания, а элементы нечетных строк – в порядке убывания. Обработку матрицы по строкам осуществлять в методе. Для упорядочения строки использовать делегата.
3. Вычислить сумму имеющих заданного массива, в котором предварительно производится попарная перестановка соседних элементов, начиная либо с первого (если значение первого элемента больше среднего арифметического элементов массива), либо с последнего элемента (в противном случае). Для нахождения суммы элементов с нечетными индексами использовать метод. Для перестановки элементов массива – делегат.
4. Вычислить сумму квадратов элементов вектора, полученного пересылкой в него либо верхнего, либо нижнего треугольника заданной квадратной матрицы (в обоих случаях включая главную диагональ). Для нахождения суммы использовать метод, для пересылки – делегата.
5. Разработать метод определения количества интервалов смены знака функции, заданной таблично на отрезке с постоянным шагом аргумента. Решить задачу для функции $y = x^2 - \sin x$ на отрезке $[0, 2]$ с шагом 0,1 и функции $y = e^x - 1$ на отрезке $[-1, 1]$ с шагом 0,2. Для вычисления очередного значения функции использовать делегата.

6. Поменять местами столбец, содержащий максимальный элемент на главной диагонали заданной квадратной матрицы, со столбцом, содержащим максимальный элемент в первой строке матрицы. Для замены столбцов использовать метод. Для поиска соответствующих максимальных элементов использовать делегата.

Решить с использованием делегата задачи №№ 7, 10, 13, 22, 27 и 28 II уровня.

Лабораторная работа №6

Структуры

Теоретическое введение. Структуры (классы) являются двумя основными типами в C#. Структура (класс) представляющая собой объектный тип данных, внешне похожа на типы данных процедурно-ориентированных языков, такие как *структура* в языке Си или *запись* в языке Паскаль. При этом элементы такой структуры (класса) могут сами быть не только данными, но и *методами* (т.е. процедурами или функциями). Такое объединение называется инкапсуляцией (см. выше).

Структуры (классы) являются типами, создаваемыми (определяемыми) пользователем, и представляют собой составной тип данных, имеющий в составе:

- *Поля данных* - параметры объекта (конечно, не все, а только необходимые в программе), задающие его состояние (свойства объекта предметной области). Иногда поля данных объекта называют свойствами объекта, из-за чего возможна путаница. Физически поля представляют собой значения (переменные, константы), объявленные как принадлежащие структуре (классу);
- *Методы* - процедуры и функции, связанные с классом. Они определяют действия, которые можно выполнять над объектом такого типа, и которые сам объект может выполнять.

Структура является типом значения. Класс является ссылочным типом (см. выше). Далее рассматривается работа со структурами и с классами (определение, создание объектов соответствующего типа, различные способы инициализации полей и т. п.). Для решения многих задач можно использовать как классы, так и структуры. Однако классы имеют более широкое применение. В частности, классы допускают наследование, что позволяет на базе одного (базового) класса без особых затрат создавать различные производные классы (см. ниже).

Использование структур

Массивы состоят из элементов одного типа. В тех случаях, когда единообразно нужно обрабатывать наборы данных, представляющих совокупность величин различных типов, рассматривая их как единое целое, целесообразно использовать структуры.

Структуры определяются с помощью ключевого слова `struct`. Далее указывается имя структуры и в фигурных скобках определяются члены структуры. Структуры могут содержать произвольное число различных видов членов: полей, методов и др.

Классы и структуры являются двумя основными конструкциями системы общих типов. Каждая из них по сути является структурой данных, инкапсулирующей набор данных (поля) и поведение (методы). Данные и поведение являются членами класса или структуры. Их объединение в одном типе называется *инкапсуляцией*. Согласно принципу инкапсуляции, класс или структура может задать уровень доступности каждого из членов по отношению к коду вне класса или структуры. Уровень доступа `public` (открытый доступ) использован в нижеследующих примерах. Другие уровни доступа здесь не рассматриваются.

В качестве членов структур (в дальнейшем и классов) в настоящем пособии будут использоваться только поля и методы.

Поле – это переменная, объявленная в структуре. У поля есть имя и тип. *Метод* – это функция, определенная в структуре.

Рассмотрим определение структуры, в которой содержатся два поля разных типов:

```
struct Sportsmen
{
    public string famile;
    public int rez;
}
```

Здесь описана структура с именем `Sportsmen` с двумя полями: `famile` типа `string` и `rez` типа `int`. Описание структуры располагается вне метода `Main`. В связи с этим уровень доступа к полям установлен максимальный (`public` – открытый доступ), что дает возможность доступа к полям из метода `Main`.

Экземпляр структуры создается в методе `Main` как обычно указанием типа перед именем переменной:

```
Sportsmen temp;
```

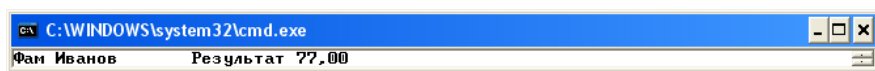
Далее в поля этой переменной можно поместить значения (инициализировать поля структуры). Для доступа к полю нужно указать имя переменной и после точки имя поля. Например,

```
using System;
class Program
{
    struct Sportsmen
    {
        public string famile;
        public int rez;
    }
}
```

```

    }
    static void Main()
    {
        Sportsmen temp;
        temp.famile = "Иванов";
        temp.rez = 77;
        Console.WriteLine("Фам {0}\t Результат {1:f2}",
                           temp.famile, temp.rez);
    }
}

```

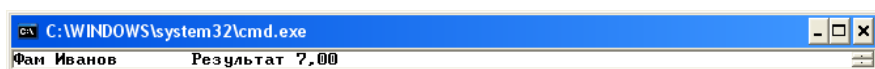


Или

```

using System;
struct Sportsmen
{
    public string famile;
    public int rez;
}
class Program
{
    static void Main()
    {
        Sportsmen temp;
        temp.famile = "Иванов";
        temp.rez = 77;
        Console.WriteLine("Фам {0}\t Результат {1:f2}",
                           temp.famile, temp.rez);
    }
}

```



Объявление массива структур. Например,

```
Sportsmen[] sp = new Sportsmen[5];
```

Здесь объявлен массив *sp* из пяти элементов, каждый из которых содержит два поля.

Использование структур делает представление данных более компактным и наглядным. Структуры можно пересылать одну в другую, если они идентичны. Возможен, например, оператор

```
sp[0] = temp;
```

Пример.

```
using System;
struct Sportsmen
{
    public string famile;
    public int rez;
}
class Program
{
    static void Main()
    {
        Sportsmen temp;
        temp.famile = "Иванов";
        temp.rez = 17;
        Sportsmen[] sp = new Sportsmen[5];
        sp[0] = temp;
        Console.WriteLine("Фам {0}\t Результат {1:f2}",
            sp[0].famile, sp[0].rez);
    }
}
```

Пример 6.1. Протокол соревнований по прыжкам в высоту содержит список фамилий и результатов (одна попытка) в порядке стартовых номеров. Получить итоговую таблицу, содержащую фамилии и результаты в порядке занятых мест. Для размещения исходных данных используется массив структур. Структура содержит информацию – фамилия и результат спортсмена. Массив структур является в данном случае одномерным массивом и для его обработки можно использовать типовые алгоритмы, рассмотренные в Части 1:

```
using System;
```

```

struct Sportsmen
{
    public string famile;
    public double rez;
}
class Program
{
    static void Main()
    {
        Sportsmen[] sp = new Sportsmen[5];
        string[] s = new string[] {
            "Иванов", "Петров", "Сидоров",
            "Кузнецов", "Макаров"};
        double[] r = new double[] {1.50,
            1.55, 1.47, 1.46, 1.54};
        for (int i = 0; i < sp.Length; i++)
        {
            sp[i].famile = s[i];
            sp[i].rez = r[i];
            Console.WriteLine(
                "Фамилия {0} \t Результат {1:f2}",
                sp[i].famile, sp[i].rez);
        }
        //Упорядочение по результатам
        for (int i = 0; i < sp.Length - 1; i++)
        {
            double amax = sp[i].rez;
            int imax = i;
            for (int j = i + 1; j < sp.Length; j++)
            {
                if (sp[j].rez > amax)
                {
                    amax = sp[j].rez;
                    imax = j;
                }
            }
        }
    }
}

```

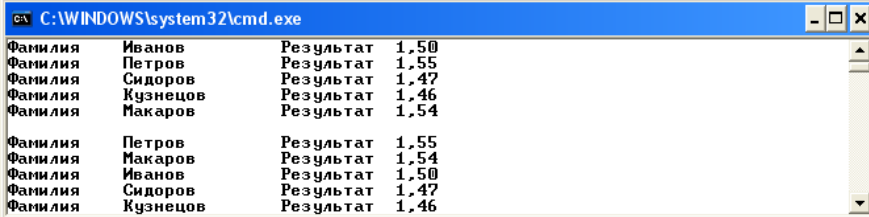
```

        Sportsmen temp;
        temp = sp[imax];
        sp[imax] = sp[i];
        sp[i] = temp;
    }
    Console.WriteLine();
    for (int i = 0; i < sp.Length; i++)
    {
        Console.WriteLine(
            "Фамилия {0} \t Результат {1:f2}",
            sp[i].famile, sp[i].rez);
    }
}
}

```

Здесь исходные данные первоначально заданы в двух массивах: фамилии в массиве *s*, результаты – в массиве *r*. Далее этими данными заполняются поля структуры.

В качестве результата будет выведено:



Фамилия	Иванов	Результат	1.50
Фамилия	Петров	Результат	1.55
Фамилия	Сидоров	Результат	1.47
Фамилия	Кузнецов	Результат	1.46
Фамилия	Макаров	Результат	1.54
Фамилия	Петров	Результат	1.55
Фамилия	Макаров	Результат	1.54
Фамилия	Иванов	Результат	1.50
Фамилия	Сидоров	Результат	1.47
Фамилия	Кузнецов	Результат	1.46

Использование конструктора экземпляра и других методов при работе со структурами

Рассмотрим более подробно различные способы инициализации полей структуры, работы с отдельными экземплярами структуры, в частности, включение методов в описание структуры на примере структуры *Sportsmen* (см. пример 6.1).

В примере 6.1 экземпляр (объект) структуры объявляется как обычная переменная указанием типа перед именем. В этом случае поля структуры остаются без значений, и их нельзя использовать до инициализации всех полей. Например,

```

using System;
namespace ConsoleApplication1
{
    struct Sportsmen

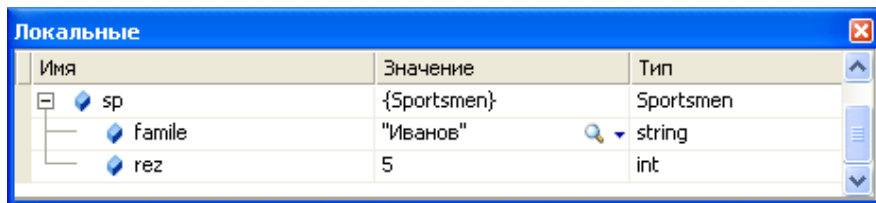
```



```

    {
        public string famile;
        public int rez;
    }
class Program
{
    static void Main(string[] args)
    {
        Sportsmen sp;
        sp.famile = "Иванов";
        sp.rez = 5;
        Console.WriteLine("Фамилия {0} Результат {1:d}",
            sp.famile, sp.rez);
    }
}

```



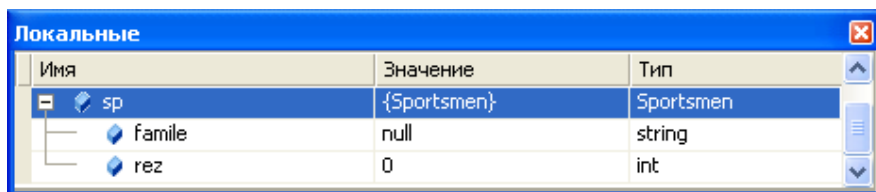
Имя	Значение	Тип
sp	{Sportsmen}	Sportsmen
famile	"Иванов"	string
rez	5	int

Попытка вывести значения полей сразу после объявления переменной (до задания значений полям) приведет к сообщению об ошибке.

Объект структуры может быть создан и другим способом: с использованием оператора new:

```
Sportsmen sp = new Sportsmen();
```

В этом случае при создании объекта вызывается соответствующий конструктор экземпляра, который выполняет инициализацию полей нулями. Значения полей будут следующими:



Имя	Значение	Тип
sp	{Sportsmen}	Sportsmen
famile	null	string
rez	0	int

Вывод значений полей в этом случае не приведет к ошибке.

Конструктор экземпляра – это метод с тем же именем, что и структура, вызываемый оператором `new`. Конструктор может иметь параметры, его можно описать при определении структуры и использовать для инициализации полей отдельных объектов структуры в более компактной и наглядной форме.

Вернемся к примеру 6.1 и в описание структуры включим еще один член: конструктор с параметрами.

```
{
    public string famile;
    public double rez;
    public Sportsmen(string famile1, double rez1)
    {
        famile = famile1;
        rez = rez1;
    }
}
```

Теперь, если объявлен массив структур

```
Sportsmen[] sp = new Sportsmen[5];
```

то задание полей элементов этого массива можно с использованием конструктора выполнить следующим образом:

```
sp[0] = new Sportsmen("Иванов", 1.50);
sp[1] = new Sportsmen("Петров", 1.55);
sp[2] = new Sportsmen("Сидоров", 1.47);
sp[3] = new Sportsmen("Кузнецов", 1.46);
sp[4] = new Sportsmen("Макаров", 1.54);
```

При вызове конструктора оператором `new` на место его первого параметра передается соответствующая фамилия, которая присваивается полю `famile`, а на место второго параметра – результат, который присваивается полю `rez` соответствующего экземпляра структуры (элементу массива структур).

Внутри конструктора могут выполняться и вычисления, необходимые для определения отдельных полей.

Пусть, например, каждый спортсмен выполняет две попытки (`rez1`, `rez2`) и окончательный результат определяется суммой двух попыток (`rez = rez1 + rez2`). Описание структуры в этом случае может быть следующим:

```
struct Sportsmen
```

```

{
    public string famile;
    public double rez1, rez2, rez;
    public Sportsmen(string famile1,
        double rezz1, double rezz2)
    {
        famile = famile1;
        rez1 = rezz1;
        rez2 = rezz2;
        rez = rez1 + rez2;
    }
}

```

Замечания.

1. Имена параметров конструктора не должны совпадать с именами полей структуры. Если они совпадают, то нужно использовать ключевое слово `this`.
Например:

```

struct Sportsmen
{
    public string famile;
    public double rez1, rez2, rez;
    public Sportsmen(string famile,
        double rez1, double rez2)
    {
        this.famile = famile;
        this.rez1 = rez1;
        this.rez2 = rez2;
        rez = rez1 + rez2;
    }
}

```

2. В конструкторе должны быть определены все поля структуры одним из двух способов: либо присваиванием значения передаваемого конструктору аргумента, либо вычислением с использованием значений уже определенных полей.

Следующая программа реализует решение задачи примера 6.1 для случая двух попыток с использованием конструктора:

```

using System;
struct Sportsmen
{
    public string famile;
    public double rez1, rez2, rez;
    public Sportsmen(string famile1,
        double rezz1, double rezz2)
    {
        famile = famile1;
        rez1 = rezz1;
        rez2 = rezz2;
        rez = rez1 + rez2;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Sportsmen[] sp = new Sportsmen[5];
        sp[0] = new Sportsmen("Иванов", 1.50, 1.52);
        sp[1] = new Sportsmen("Петров", 1.55, 1.8);
        sp[2] = new Sportsmen("Сидоров", 1.47, 1.5);
        sp[3] = new Sportsmen("Кузнецов", 1.46, 1.43);
        sp[4] = new Sportsmen("Макаров", 1.54, 1.44);
        for (int i = 0; i < sp.Length; i++)
        {
            Console.WriteLine(
                "Фамилия    {0}\t Результат  {1:f2}",
                sp[i].famile, sp[i].rez);
        }
    }
}

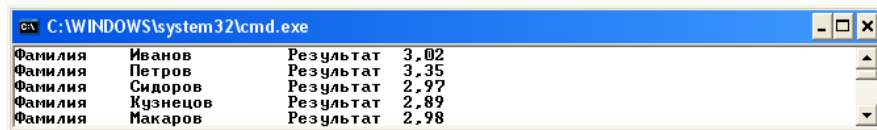
```

Теперь при выполнении, например, оператора

```
sp[0] = new Sportsmen("Иванов", 1.50, 1.52);
```

в поле `rez` будет помещаться сумма 3,02.

Результат выполнения программы:



Фамилия	Результат
Иванов	3.02
Петров	3.35
Сидоров	2.97
Кузнецов	2.89
Макаров	2.98

Помимо конструктора описание структуры может содержать и другие методы. Например, в описании структуры

```
struct Sportsmen
{
    public string famile;
    public double rez;
    public double factor(int i)
    {
        return i * rez;
    }
}
```

описан метод `factor` для умножения результата `rez` на коэффициент `i`.

После объявления в методе `Main` объекта структуры этот метод может быть вызван указанием имени объекта структуры и после точки имени метода и его аргумента в круглых скобках:

```
Sportsmen sp;
sp.famile = "Иванов";
sp.rez = 5.2;
double rez1 = sp.factor(3);
Console.WriteLine(rez1);
```

В результате переменная `rez1` получит значение 15,6.



15.6

В описании структуры может быть описан также статический метод. Статический метод не применяется к конкретному объекту структуры, а относится ко всей структуре, и доступ к нему осуществляется по имени структуры, а не экземпляра (объекта) структуры (см. выше).

В качестве примера рассмотрим описанную выше структуру и добавим в ее описание статический метод для подсчета количества спортсменов. Этот метод будет прибавлять к общему количеству 1 после ввода данных для очередного спортсмена:

```

struct Sportsmen
{
    public string famile;
    public double rez;
    public static int sportsmenCounter = 0;
    public static int AddSportsmen()
    {
        return sportsmenCounter = sportsmenCounter + 1;
    }
}

```

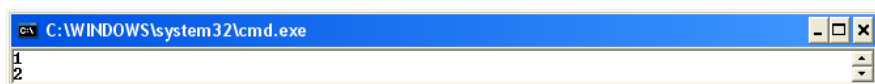
Если в методе Main имеются операторы

```

Sportsmen sp;
sp.famile = "Иванов";
sp.rez = 5.2;
Sportsmen.AddSportsmen();//Вызов статического метода
Console.WriteLine (Sportsmen.sportsmenCounter);
sp.famile = "Петров";
sp.rez = 7.4;
Sportsmen.AddSportsmen();
Console.WriteLine(Sportsmen.sportsmenCounter);

```

то в результате их выполнения первым оператором WriteLine будет выведено 1, а вторым – 2.



Пример 6.2. Студенты одной группы в сессию сдают четыре экзамена. Составить список студентов, средний балл которых по всем экзаменам не менее «4». Результаты вывести в виде таблицы с заголовком в порядке убывания среднего балла:

```

using System;
struct Struct1
{
    public string famile;
    public double[] x;
    public double sred;
    public Struct1(string famile1, double[] x1)

```

```

    {
        sred = 0;
        famile = famile1;
        x = x1;
        for (int i = 0; i < 4; i++)
            sred += x[i];
        sred /= 4; // sred = sred/4
    }
}

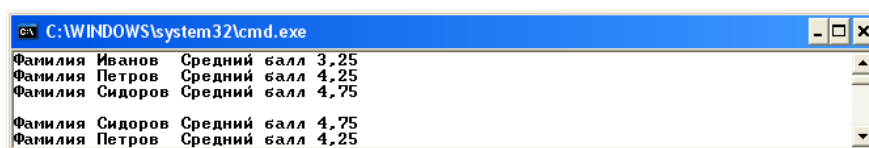
class Program
{
    static void Main(string[] args)
    {
        Struct1[] cl = new Struct1[3];
        cl[0] = new Struct1("Иванов",
            new double[] {3.0, 5.0, 2.0, 3.0});
        cl[1] = new Struct1("Петров",
            new double[] {5.0, 4.0, 5.0, 3.0});
        cl[2] = new Struct1("Сидоров",
            new double[] {5.0, 4.0, 5.0, 5.0});
        for (int i = 0; i < cl.Length; i++)
            Console.WriteLine(
                "Фамилия {0}\t Средний балл {1,4:f2}",
                cl[i].famile, cl[i].sred);
        for (int i = 0; i < cl.Length - 1; i++)
        {
            double amax = cl[i].sred;
            int imax = i;
            for (int j = i + 1; j < cl.Length; j++)
            {
                if (cl[j].sred > amax)
                {
                    amax = cl[j].sred;
                    imax = j;
                }
            }
        }
    }
}

```

```

        Struct1 temp;
        temp = cl[imax];
        cl[imax] = cl[i];
        cl[i] = temp;
    }
    Console.WriteLine();
    for (int i = 0; i < cl.Length; i++)
    {
        if (cl[i].sred >= 4)
            Console.WriteLine(
                "Фамилия {0}\t "
                + "Средний балл {1,4:f2}",
                cl[i].famile, cl[i].sred);
    }
}

```



Здесь членом структуры является конструктор с двумя параметрами, соответствующими двум полям структуры, которые будут заполняться при вызове конструктора для каждого объекта структуры. При этом значение третьего поля вычисляется в конструкторе с использованием значений элементов массива, являющегося вторым полем структуры.

Вопросы для самопроверки

1. Структура. Поля структуры. Члены структуры. Объявление структуры. Доступ к отдельным полям структуры.
2. Создание экземпляра структуры. Инициализация полей структуры.
3. Операции со структурами.
4. Преимущества использования структур.
5. Массивы структур и их обработка.
6. Создание объекта структуры при помощи конструктора. Что такое конструктор экземпляра.

7. Особенности инициализации полей структуры при использовании конструктора экземпляра.
8. Различные способы задания значений полей структуры при использовании конструктора.
9. Возможность использования методов, как членов структуры.

Задания I уровня

Задания I уровня предназначены для приобретения навыков решения задач с использованием структур.

Решить задачи с использованием конструктора и различных вариантов объявления структуры (внутри класса, вне класса). Данные задать самостоятельно. Составить программу и проверить ее решение на компьютере.

1. Результаты соревнований по прыжкам в длину определяются по сумме двух попыток. В протоколе для каждого участника указываются: фамилия, общество, результаты первой и второй попыток. Вывести протокол в виде таблицы с заголовком в порядке занятых мест.
2. Составить программу для обработки результатов кросса на 500 м для женщин. Для каждой участницы ввести фамилию, группу, фамилию преподавателя, результат. Получить результирующую таблицу, упорядоченную по результатам, в которой содержится также информация о выполнении норматива. Определить суммарное количество участниц, выполнивших норматив.
3. Радиокомпания провела опрос слушателей по вопросу: «Кого вы считаете человеком года?». Определить пять наиболее часто встречающихся ответов и их долей (в процентах от общего количества ответов).
4. Результаты соревнований по прыжкам в высоту определяются по лучшей из двух попыток. Вывести список участников в порядке занятых мест.
5. Группе студентов в результате полусеместровой аттестации были выставлены оценки по информатике, а также определено количество пропущенных занятий. Успеваемость каждого студента оценивается следующими баллами: «0» (неаттестован), «2», «3», «4» или «5». Вывести список неуспевающих (оценка «2») студентов в порядке убывания количества пропущенных ими занятий.

Задания II уровня

Задание II уровня предназначено для приобретения навыков решения задач с использованием массивов и методов в качестве полей структуры. Выполнить также все пункты задания I уровня.

1. Студенты одной группы в сессию сдают четыре экзамена. Составить список студентов, средний балл которых по всем экзаменам не менее «4». Результаты вывести в виде таблицы с заголовком в порядке убывания среднего балла.
2. Группа учащихся подготовительного отделения сдает выпускные экзамены по трем предметам (математика, физика, русский язык). Учащийся, получивший "2" сразу отчисляется. Вывести список учащихся, успешно сдавших экзамены, в порядке убывания полученного ими среднего балла по результатам трех экзаменов.
3. На основании результатов соревнований по прыжкам в длину (фамилии и результаты трех попыток) составить итоговый протокол соревнований, считая, что в зачет идет лучший результат.
4. В соревнованиях по прыжкам в воду оценивают 7 судей. Каждый спортсмен выполняет 4 прыжка. Каждый прыжок имеет одну из шести категорий сложности, оцениваемую коэффициентом (от 2,5 до 3,5). Качество прыжка оценивается судьями по 6-балльной шкале. Далее лучшая и худшая оценки отбрасываются, остальные складываются, и сумма умножается на коэффициент сложности. Получить итоговую таблицу, содержащую фамилии спортсменов и итоговую оценку (сумму оценок по 4 прыжкам) в порядке занятых мест.
5. Соревнования по прыжкам на лыжах со 120-метрового трамплина судят 5 судей. Каждый судья выставляет оценку за стиль прыжка по 20-балльной шкале. Меньшая и большая оценки отбрасываются, остальные суммируются. К этой сумме прибавляются очки за дальность прыжка: 120 метров – 60 очков, за каждый метр превышения добавляются по 2 очка, при меньшей дальности отнимаются 2 очка за каждый метр. Получить итоговую таблицу соревнований, содержащую фамилию и итоговый результат для каждого участника в порядке занятых мест.
6. Протокол соревнований по прыжкам в воду содержит список фамилий спортсменов и баллы, выставленные 5 судьями по результатам 2 прыжков. Получить итоговый протокол, содержащий фамилии и результаты, в порядке занятых спортсменами мест по результатам 2 прыжков.
7. После окончания соревнования по шахматам турнирная таблица содержит фамилии участников и результаты сыгранных партий (выигрыш – 1 очко, ничья – 1/2 очка, проигрыш – 0 очков). Составить итоговую таблицу в порядке убывания полученных участниками очков.

8. Для формирования сборной по хоккею предварительно отобрано 30 игроков. На основании протоколов игр составлена таблица, в которой содержится штрафное время каждого игрока по каждой игре (2, 5 или 10 мин). Написать программу, которая составляет список кандидатов в сборную в порядке возрастания суммарного штрафного времени. Игрок, оштрафованный на 10 мин, из списка кандидатов исключается.
9. Результаты соревнований фигуристов по одному из видов многоборья представлены оценками семи судей в баллах (от 0,0 до 6,0). По результатам оценок судьи определяется место каждого участника у этого судьи. Места участников определяются далее по сумме мест, которые каждый участник занял у всех судей. Составить программу, определяющую по исходной таблице оценок фамилии и сумму мест участников в порядке занятых ими мест.

Задания III уровня

Задание III уровня предназначены для приобретения навыков решения задач с использованием структур и методов в сочетании, а также некоторого творческого подхода. Выполнить также все пункты задания II уровня.

1. Результаты сессии содержат оценки 5 экзаменов по каждой группе. Определить средний балл для трех групп студентов одного потока и выдать список групп в порядке убывания среднего бала. Результаты вывести в виде таблицы с заголовком.
2. Соревнования по футболу между командами проводятся в два этапа. Для проведения первого этапа участники разбиваются на две группы по 12 команд. Для проведения второго этапа выбирается 6 лучших команд каждой группы по результатам первого этапа. Составить список команд участников второго этапа.
3. В соревнованиях участвуют три команды по 6 человек. Результаты соревнований представлены в виде мест участников каждой команды (1 - 18). Определить команду – победителя, вычислив количество баллов, набранное каждой командой. Участнику, занявшему 1-е место, начисляется 5 баллов, 2-е – 4, 3-е – 3, 4-е – 2, 5-е – 1, остальным – 0 баллов. При равенстве баллов

победительницей считается команда, за которую выступает участник, занявший 1-е место.

4. Лыжные гонки проводятся отдельно для двух групп участников. Результаты соревнований заданы в виде фамилий участников и их результатов в каждой группе. Расположить результаты соревнований в каждой группе в порядке занятых мест. Объединить результаты обеих групп с сохранением упорядоченности и вывести в виде таблицы с заголовком.
5. Обработать результаты первенства по футболу. Результаты каждой игры заданы в виде названий команд и счета (количество забитых и пропущенных мячей). Сформировать таблицу очков (выигрыш – 3, ничья – 1, проигрыш – 0) и упорядочить результаты в соответствии с занятым местом. Если сумма очков у двух команд одинакова, то сравниваются разности забитых и пропущенных мячей. Вывести результирующую таблицу, содержащую место, название команды, количество очков.
6. Японская радиокompания провела опрос радиослушателей по трем вопросам:
 - а) Какое животное Вы связываете с Японией и японцами?
 - б) Какая черта характера присуща японцам больше всего?
 - в) Какой неодушевленный предмет или понятие Вы связываете с Японией?

Большинство опрошенных прислали ответы на все или часть вопросов. Составить программу получения первых пяти наиболее часто встречающихся ответов по каждому вопросу и доли (в %) каждого такого ответа. Предусмотреть необходимость сжатия столбца ответов в случае отсутствия ответов на некоторые вопросы.

Лабораторная работа №7

Классы

Теоретическое введение. *Класс*, как и структура, представляет собой создаваемый пользователем тип. Класс является ссылочным типом.

Использование классов

Классы определяются с помощью ключевого слова `class`. Далее указывается имя класса и в фигурных скобках определяются члены класса. Классы могут содержать произвольное число различных видов членов: полей, методов и др.

Поле – это переменная, объявленная в классе. У поля есть имя и тип. *Метод* – это функция, определенная в классе.

Рассмотрим определение класса, в котором содержатся два поля разных типов:

```
class Sportsmen
{
    public string famile;
    public int rez;
}
```

Здесь описан класс с именем `Sportsmen` с двумя полями: `famile` типа `string` и `rez` типа `int`. Описание класса располагается вне метода `Main`. В связи с этим уровень доступа к полям установлен максимальный (`public` – открытый доступ), что дает возможность доступа к полям из метода `Main`.

При создании экземпляра класса переменная, к которой назначается экземпляр, сохраняет только ссылку на память. Экземпляр класса создается в методе `Main` как обычно указанием типа перед именем переменной и обязательным использованием ключевого слова `new`:

```
Sportsmen temp = new Sportsmen();
```

Далее в поля этой переменной можно поместить значения (инициализировать поля класса). Для доступа к полю экземпляра класса нужно указать имя переменной и после точки имя поля. Например,

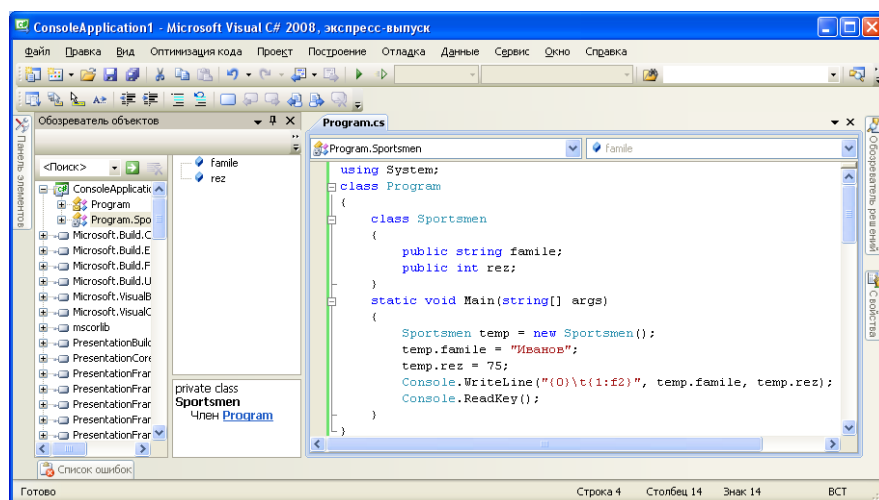
```
using System;
class Sportsmen
{
```

```

        public string famile;
        public int rez;
    }
class Program
{
    static void Main(string[] args)
    {
        Sportsmen temp = new Sportsmen();
        temp.famile = "Иванов";
        temp.rez = 75;
        Console.WriteLine(
            "{0}\t{1:f2}", temp.famile, temp.rez);
    }
}

```

ИЛИ



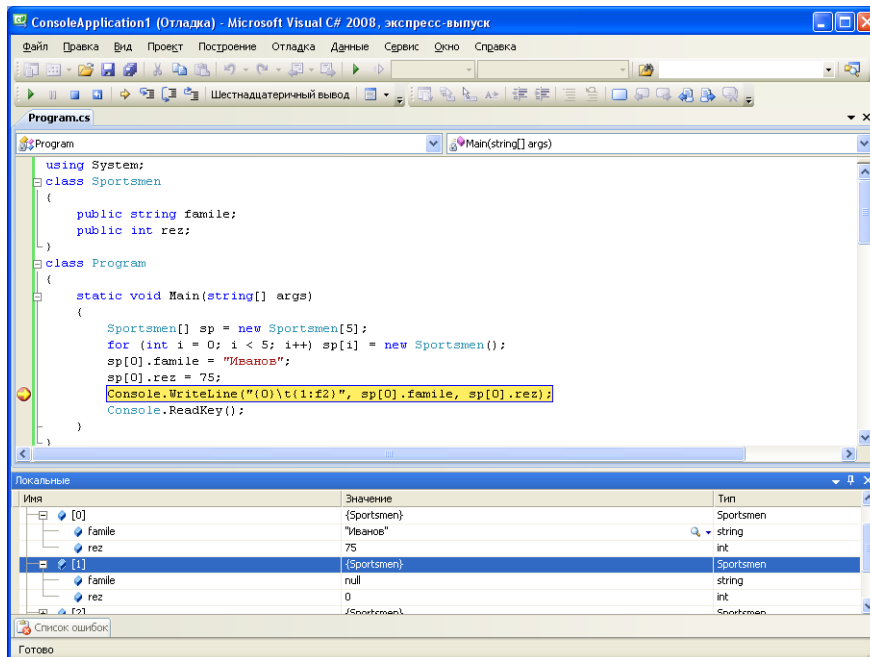
Объявление массива классов. Например,

```
Sportsmen[] sp = new Sportsmen[5];
```

Здесь объявлен массив *sp* из 5 элементов, каждый из которых содержит 2 поля.

Далее для каждого элемента массива необходимо выделить память:

```
for (int i = 0; i < 5; i++) sp[i] = new Sportsmen();
```



Использование классов так же, как использование структур, делает представление данных более компактным и наглядным.

Можно создать конструктор с параметрами для инициализации полей экземпляра класса:

```
using System;

class Sportsmen
{
    public string famile;
    public double rez;
    public Sportsmen(string famile1, double rez1)
    {
        famile = famile1;
        rez = rez1;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Sportsmen[] sp = new Sportsmen[3] {
            new Sportsmen("Иванов", 1.50),
            new Sportsmen("Петров", 1.55),
            new Sportsmen("Сидоров", 1.47)};
    }
}
```

```

        for (int i = 0; i < sp.Length; i++)
            Console.WriteLine(
                "Фам {0}\t Результат    {1:f2}",
                sp[i].famile, sp[i].rez);
    }
}

```

Наследование

В отличие от структур, классы поддерживают *наследование*, фундаментальную характеристику объектно-ориентированного программирования.

Класс, члены которого наследуются, называется *базовым классом*, а класс, который наследует эти члены, называется *производным классом*. При определении класса для наследования от другого класса, производный класс явно получает все члены базового класса, за исключением его конструкторов. Например,

```

using System;
class Sportsmen
{
    public string famile;
    public double rez;
}
class Sportsmen1 : Sportsmen
{
    public string team;
}
class Program
{
    static void Main(string[] args)
    {
        Sportsmen1 sp = new Sportsmen1();
        sp.famile = "Иванов";
        sp.rez = 77;
        sp.team = "Спартак";
        Console.WriteLine("Фам {0}\tКоманда {1}\tРезультат
                           {2:f2}", sp.famile, sp.team, sp.rez);
    }
}

```


Здесь класс `Sportsmen` - базовый, класс `Sportsmen1` наследует ему (является производным). В производном классе могут быть добавлены члены (поля, методы).

В данном примере в классе `Sportsmen1` определено еще одно поле `team`.

В результате будет выведено:



Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах, таким образом, наследование позволяет создавать новые классы на базе уже существующих классов.

Создавая новый тип данных на базе типа данных, определенного ранее, можно упростить работу за счет использования разработанных ранее методов базового класса.

Рассмотрим следующий пример.

Пример 7.1 Пусть нам нужно работать в программе с такими объектами, как прямоугольники.

Для этого мы создаем класс `Rectangle`, инкапсулирующий в себе данные и методы, необходимые для размещения прямоугольника на плоскости:

```
class Rectangle
{
    int xPos; //координаты левого нижнего
    int yPos; //угла прямоугольника
    int width; //ширина прямоугольника
    int height; // высота прямоугольника
    public Rectangle()
    {
        xPos = yPos = width = height = 0;
    }
    public void SetPosition(int x, int y)
    {
        xPos = x;
        yPos = y;
    }
    public void SetSize(int w, int h)
    {
        width = w;
```

```

        height = h;
    }
}

```

Таким образом, содержимое полей класса `Rectangle` однозначно определяет расположение прямоугольника на плоскости и его размеры (рис.8).

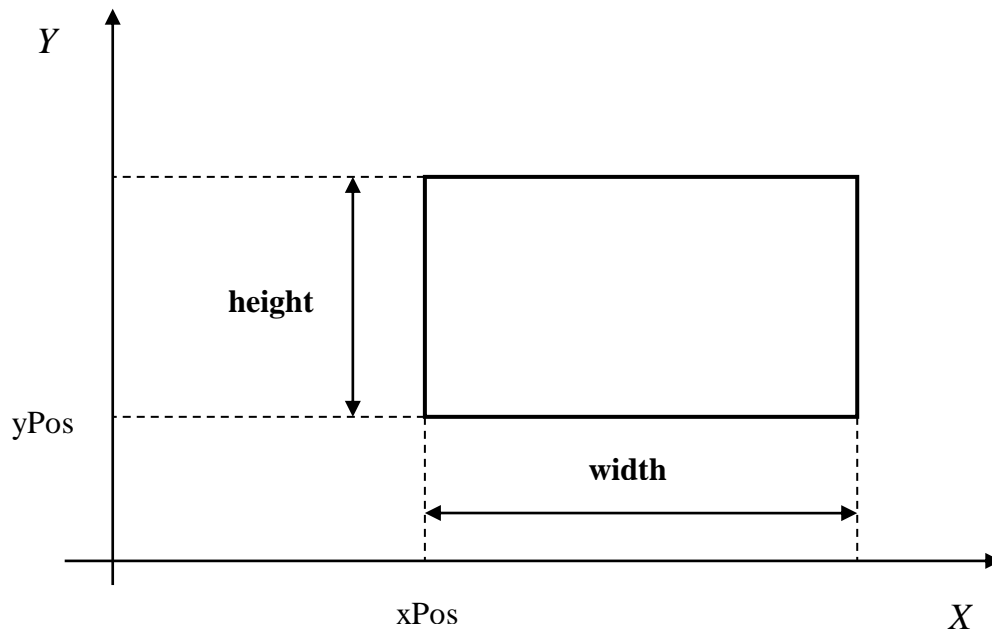


Рис. 8

Конструктор класса `Rectangle` записывает в поля класса нулевые значения, в результате чего все новые прямоугольники создаются в центре системы координат с размерами, равными нулю.

В классе `Rectangle` мы также определили методы `SetPosition` и `SetSize`, позволяющие установить соответственно расположение прямоугольника на плоскости и его размеры.

Работать с этим классом очень просто — нужно сначала создать прямоугольник, а затем установить его расположение и размеры:

```

Rectangle r;
r = new Rectangle();
r.SetPosition (0, 10);
r.SetSize(20, 30);

```

Итак, мы создали базовый класс, описывающий прямоугольники. А теперь представим себе, что нам понадобились не простые прямоугольники, а раскрашенные в

разные цвета. Мы, конечно, можем изменить класс `Rectangle`, добавив в него поля и методы для работы с цветом. Однако более красивое решение заключается в создании на базе класса `Rectangle` нового класса `ColorRectangle`, дополненного средствами представления цвета:

```
class ColorRectangle : Rectangle
{
    byte colorR; // тип byte - целые числа без знака
    byte colorG;
    byte colorB;
    public void SetColor(byte r, byte g, byte b)
    {
        colorR = r;
        colorG = g;
        colorB = b;
    }
}
```

В результате класс `ColorRectangle` наследует все поля и методы своего базового класса, а также добавляет к ним еще 3 поля (типа `byte` – короткое целое без знака): `colorR`, `colorG`, `colorB` и один метод — `SetColor`. Новые поля предназначены для хранения трех основных цветов (красный, зеленый и синий), а метод `SetColor` позволяет установить значения этих полей, раскрасив наш прямоугольник.

```
using System;
namespace Rectangle
{
    class Rectangle
    {
        int xPos;
        int yPos;
        int width;
        int height;
        public Rectangle()
        {
            xPos = yPos = width = height = 0;
        }
    }
}
```

```

public void SetPosition(int x, int y)
{
    xPos = x;
    yPos = y;
}

public void SetSize(int w, int h)
{
    width = w;
    height = h;
}
}

class ColorRectangle : Rectangle
{
    byte colorR;
    byte colorG;
    byte colorB;
    public void SetColor(byte r, byte g, byte b)
    {
        colorR = r;
        colorG = g;
        colorB = b;
    }
}

class RectangleApp
{
    static void Main(string[] args)
    {
        ColorRectangle cr;
        cr = new ColorRectangle();
        cr.SetPosition(0, 10);
        cr.SetSize(20, 30);
        cr.SetColor(0, 0, 0xFF); //красный
        Console.ReadLine();
    }
}
}

```

Получив управление, метод `Main` создает объект производного класса `cr`:

```
ColorRectangle cr;  
cr = new ColorRectangle();
```

Далее этот метод вызывает два метода базового класса для изменения расположения и размеров прямоугольника:

```
cr.SetPosition(0, 10);  
cr.SetSize(20, 30);
```

Обратите внимание: в классе `ColorRectangle` нет определения методов `SetPosition` и `SetSize`, так как это методы базового класса. Тем не менее мы вызываем их для объекта `cr` производного класса. Модификатор доступа `public`, примененный при объявлении методов `SetPosition` и `SetSize` в базовом классе, допускает такой вызов.

Аналогичным образом мы вызываем метод `SetColor` производного класса:

```
cr.SetColor (0, 0, 0xFF);
```

чего **нельзя** сделать для объекта базового класса

Таким образом, мы очень легко добавили новую функциональность - теперь наши прямоугольники стали цветными. При добавлении новой сущности (цвета) нам были несущественны детали реализации базового класса `Rectangle`, такие, как способ размещения прямоугольника на плоскости и способ установки его размеров. Более того, создавая свой собственный класс на базе готового класса, программист может даже и не подозревать о существовании в этом классе каких-то еще полей и методов. Например, в базовом классе `Rectangle` могло существовать поле для хранения запаха прямоугольника, а также метод для установки этого запаха.

Пока мы имеем дело с простыми классами и простыми программами, эффект от использования наследования может показаться небольшим. Однако в реальности наследование открывает перед программистом возможность создания своих классов на базе огромной библиотеки классов `C#`.

Взяв за основу один из десятков тысяч классов библиотеки, вы можете создать на его базе собственный класс, наделив его необходимыми вам свойствами. При этом вам не потребуется реализовывать полную функциональность, так как она уже реализована в базовом классе. Достаточно только добавить свои поля и методы.

Рассмотрим еще один простой пример.

Пример 7.2. Индекс массы тела взрослого человека вычисляется как отношение веса к квадрату роста.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            double d;
            imt hg = new imt();
            hg.r = 1.55;
            hg.w = 90.1;
            hg.fam = "tania";
            d = hg.im();
            Console.WriteLine(d);
        }
    }
    class imt
    {
        public string fam;
        public double r, w;
        public double im ()
        {
            return w / (r * r);
        }
    }
}

```

Однако для людей старшего возраста (старше 50 лет) к полученному результату добавляется 5%, т.е. результат берется с коэффициентом 1,05.

С наследованием

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)

```

```

    {
        double d;
        imt50 hg = new imt50();
        hg.r = 1.55;
        hg.w = 55.1;
        hg.fam = "tania";
        hg.age = 60;
        d = hg.im50();
        Console.WriteLine(d);
    }
}

class imt // базовый класс
{
    public string fam;
    public double r, w;
    public double im ()
    {
        return w / (r * r);
    }
}

class imt50 : imt // производный класс
{
    public double age;
    public double im50()
    {
        if (age > 50)
        {
            return w * 1.05/(r * r);
        }
        else return w / (r * r);
    }
}
}

```

Вопросы для самопроверки

1. Что такое класс? К какому типу относится класс? Как определить класс?
2. Каковы основные члены класса?
3. Как создать экземпляр класса?
4. Инициализация полей класса.
5. Объявление массива классов. Особенности выделения памяти под массив классов.
6. Использование конструктора экземпляра при работе с классами.
7. Что такое наследование классов и как оно реализуется?
8. Укажите основные различия между классами и структурами.

Задания для самостоятельного выполнения

Выполнить задание к ЛР 6 с использованием классов, используя наследование по указанию преподавателя.

Лабораторная работа №8

Работа с текстовыми строками

Теоретическое введение. *Текстовые строки* – переменные типа `string` – могут содержать любое количество символов. Каждый символ представлен в кодировке UNICODE, предполагающей представление одного символа в 2 байтах памяти. Работа с текстовыми строками обычно предполагает решение следующих задач: объединить две (или более) строки в одну, вырезать из строки фрагмент, найти в строке заданную подстроку и т. п.

Для текстовых строк определены операции: *конкатенации (объединение) строк*, эта операция может быть выполнена при помощи оператора `+` или метода `String.Concat`, сравнение строк: операторы равенства `==` и `!=`. Оператор `[]` служит для доступа (только для чтения) к отдельным символам объекта `string` (см. ниже).

Например, объединение строк

```
string str1="Катя";  
string str2="Иванова";  
string res = str1+ " "+ str2;  
Console.WriteLine(res);
```

Строка `res` будет следующей: "Катя Иванова".

Этот же результат можно получить при помощи метода `Concat`:

```
string res1 = String.Concat(str1, " ",str2);
```

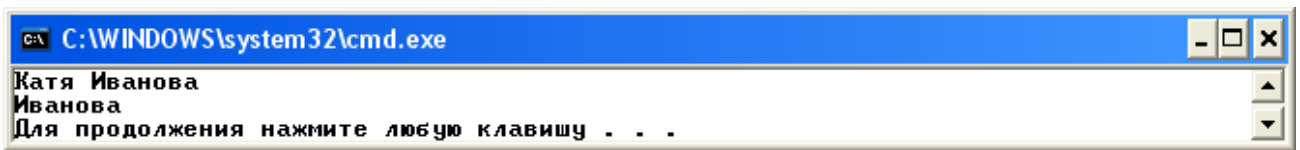
В списке аргументов метода `Concat` может быть не более четырех членов. В приведенном выше примере их три.

Текстовые строки имеет сходство с массивами в том смысле, что доступ к отдельным символам осуществляется по индексу (номеру позиции, нумерация начинается с 0). Переход к следующему символу легко осуществляется изменением номера позиции на 1. Однако следует иметь в виду, что **изменение однажды созданной строки не допускается**. Чтобы внести требуемые изменения, нужно создать новую строку или использовать соответствующий метод.

Для работы со строками определены следующие методы класса `String`.

Метод `Substring` – извлечение подстроки длиной m , начиная с n -го символа. Если m не указано, то выводится весь «хвост» строки, начиная с n -го символа. Например, применение метода `Substring` для строки `res`

```
string str1 = "Катя";  
string str2 = "Иванова";  
string res = str1 + " " + str2;  
Console.WriteLine(res);  
string res2 = res.Substring(5, 7);  
Console.WriteLine(res2);
```



Метод `Insert` – вставка подстроки в исходную строку, начиная с n -ой позиции. При этом необходимо создать новую переменную, в которой вначале будет автоматически продублирована исходная строка, а затем выполнена необходимая операция. (Еще раз отметим, что в C# прямая модификация существующей строки невозможна.) Например,

```
string res3 = "Катя";  
string res4 = res3.Insert(4, " Иванова");  
Console.WriteLine(res4);
```

В результате в `res4` будет "Катя Иванова".

Метод `Replace` – замена подстроки новой подстрокой или замена какого-либо символа во всем тексте на другой символ. Например,

```
string str = "Катя Иванова";  
string str1 = str.Replace("Катя", "Екатерина");
```

В результате будет `str1 = "Екатерина Иванова"`.

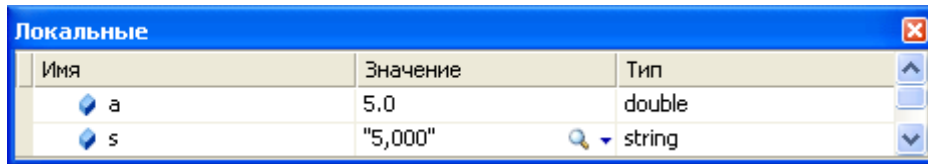
Метод `Remove` – удаление из строки фрагмента заданной длины m , начинающегося с заданной позиции n . Например,

```
string res = "Катя Иванова – моя подруга";  
string res1 = res.Remove(4, 8);  
Console.WriteLine(res1);
```


Удаляются 8 символов, начиная с 4-го (пробел после слова "Катя"). В результате будет `res1 = "Катя – моя подруга"`.

Метод `ToString` – получение строкового представления объекта числового типа. Внутри скобок может ничего не находиться, или может находиться переменная (строка формата), в которой указан способ форматирования числа, сохраненного в строковом виде, при выводе на консоль. Например,

```
double a = 5.0;
string s = a.ToString("f3");
Console.WriteLine(s);
Console.ReadKey();
```



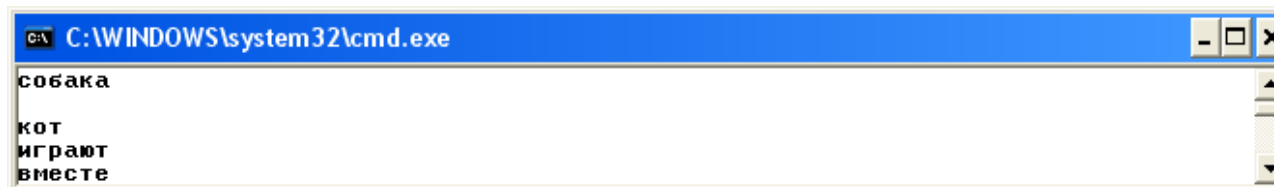
Имя	Значение	Тип
a	5.0	double
s	"5,000"	string



```
C:\WINDOWS\system32\cmd.exe
5,000
Для продолжения нажмите любую клавишу . . . _
```

Метод `Split` (применяется к экземпляру класса `String`, как и описанные выше методы) осуществляет разбор строки, т.е. позволяет выделить отдельные слова или другие сочетания символов, разделенные какими-либо разделителями, перечисляемыми в массиве символов типа `char[]`, являющемся аргументом метода `Split`. Например,

```
string str = "собака, кот играют вместе";
string[] strarr = str.Split(new Char[] { ' ', ',' });
foreach(string res in strarr)
{
    Console.WriteLine(res);
}
```



```
C:\WINDOWS\system32\cmd.exe
собака
кот
играют
вместе
```

Здесь формируется массив `strarr` из слов исходного текста, отделенных друг от друга пробелом или запятой. Список символов-разделителей помещается в массив и передается методу `Split` в качестве аргумента.

При выводе после первого слова выведена пустая строка, т. к. после первого разделителя сразу следует второй и между ними ничего нет (пусто).

Каждый отдельный символ строки является значением типа `char` и может быть выделен в переменную типа `char`. Например, при выполнении оператора

```
char s = str[5];
```

для `str` из предыдущего примера `s` получит значение `a`.

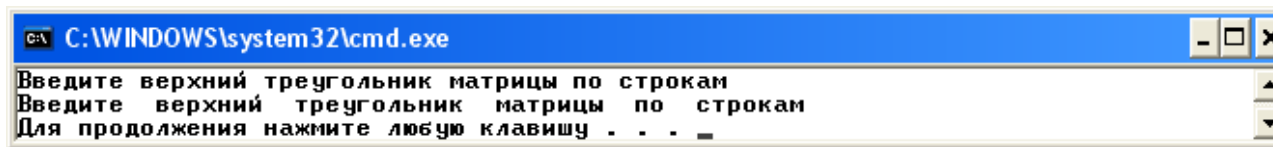
Для работы с отдельными символами строки можно использовать методы структуры `Char`. Например, статический метод

`Char.IsDigit(s)`, где `s` – переменная типа `char` (отдельный символ строки). Этот метод возвращает значение `true`, если `s` – цифра и `false`, если `s` – не цифра. Результат выполнения метода показывает, относится ли указанный символ Юникода к категории десятичных цифр или нет.

Использование этого и других подобных методов иллюстрируется в примере 8.4.

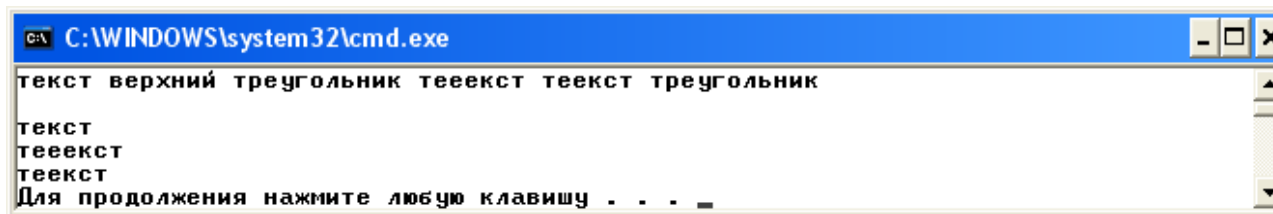
Пример 8.1. В исходном тексте одно слово от другого отделено одним пробелом. Сформировать текст, в котором одно слово от другого отделяется двумя пробелами.

```
using System;
class Program
{
    static void Main()
    {
        string str1 = "Введите верхний треугольник матрицы по строкам";
        string str2 = "";
        for (int i = 0; i < str1.Length; i++)
        {
            if (str1[i] == ' ')
            {
                str2 = str2 + str1[i];
            }
            str2 = str2 + str1[i];
        }
        Console.WriteLine(str1);
        Console.WriteLine(str2);
    }
}
```



Пример 8.2. Выписать из текста слова, начинающиеся и заканчивающиеся на одну и ту же букву.

```
using System;
class Program
{
    static void Main()
    {
        string str1 = "текст верхний треугольник тееекст
теекст треугольник ";
        Console.WriteLine(str1);
        Console.WriteLine();
        int i = 0, j = 0;
        while (i < str1.Length)
        {
            if (str1[i] == ' ')
            {
                if (str1[i - 1] == str1[j])
                {
                    Console.WriteLine(str1.Substring(j, i - j));
                }
                j = i + 1; i++;
            }
            else
            {
                i++;
            }
        }
    }
}
```



Пример 8.3. Подсчитать, сколько слов в тексте начинается на букву т. Слова в тексте разделены пробелами.

```
using System;
class Program
{
    static void Main()
    {
        string str1 = "текст верхний треугольник тееекст
теееекст треугольник верхний";
        Console.WriteLine(str1);
        Console.WriteLine();
        string[] masstr = str1.Split(' ');
        int k = 0;
        for (int i = 0; i < masstr.Length; i++)
        {
            if (masstr[i][0] == 'т') k++;
        }
        Console.WriteLine(k);
    }
}
```

Пример 8.4. Определить, сколько в заданном тексте отдельно цифр (digit), букв (letter), заглавных букв (upper) и разделителей (separator).

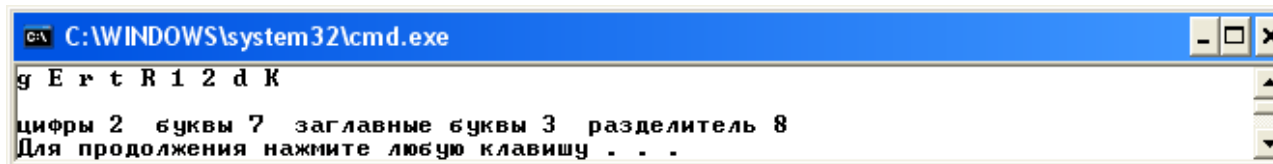
```
using System;
class Program
{
    static void Main()
    {
        string str = "g E r t R 1 2 d K";
        Console.WriteLine(str);
        Console.WriteLine();
    }
}
```

```

int k = 0, l = 0, m = 0, j = 0;
for (int i = 0; i < str.Length; i++)
{
    if (Char.IsDigit(str[i])) k++;
    if (Char.IsLetter(str[i])) l++;
    if (Char.IsUpper(str[i])) m++;
    if (Char.IsSeparator(str[i])) j++;

    Console.WriteLine("цифры {0}   буквы {1}   заглавные
буквы {2}   разделитель {3}", k, l, m, j);
}
}

```



Вопросы для самопроверки

1. Как задать текстовую строку?
2. Какие операции определены для текстовых строк?
3. Как получить доступ к отдельным символам строки?
4. Методы, определенные для символьных строк.
5. Методы, определенные для отдельных символов.
6. Как можно изменить строку в процессе выполнения программы?
7. Метод `Split`. Возможности его использования для ввода данных с клавиатуры.

Задания для самостоятельного выполнения

1. В заданном тексте определить частоту, с которой встречаются в тексте различные буквы русского алфавита (в долях от общего количества букв).
2. Ученики зашифровывают свои записки, записывая все слова наоборот. Составить программу, зашифровывающую и расшифровывающую сообщение.
3. Разбить исходный текст на строки длиной не более 50 символов. Перенос на новую строку осуществлять на месте пробела (слова не переносить).

4. Назовем сложностью предложения сумму количества слов и знаков препинания. Определить сложность заданного предложения.
5. Задан текст, содержащий не более 1000 символов. Напечатать буквы, на которые начинаются слова в тексте, в порядке убывания частоты их употребления.
6. Определить, сколько слов в тексте содержит один слог, два слога, три слога и т.д.
7. Задан текст содержащий не более 1000 символов. Выписать все слова, включающие заданную последовательность букв (например, выписать однокоренные слова).
8. Разделить заданный текст (не более 1000 символов) на строки, содержащие не более 50 символов. (Перенос осуществлять на месте пробела). Добавить равномерно пробелы, чтобы каждая строка содержала ровно 50 символов.
9. Для хранения текста в сжатом виде найти часто повторяющиеся последовательности из двух букв и заменить их кодом. В качестве кода использовать символы, не встречающиеся в тексте. Составить также таблицу кодов.
10. В предыдущей задаче декодировать текст, хранящийся в сжатом виде, используя заданную таблицу кодов.
11. Список фамилий, разделенных запятыми, задан в произвольном порядке. Упорядочить его по алфавиту.
12. Считая, что в памяти компьютера хранится таблица кодов часто встречающихся слов, ввести текст в массив, заменяя слова кодами после ввода. Распечатать текст в исходном виде, т.е. заменяя коды словами.
13. Определить долю в процентах слов, начинающихся на различные буквы. Выписать эти буквы и доли начинающихся на них слов.
14. Текст содержит слова и целые числа от 1 до 10. Найти сумму включенных в текст чисел.
15. Текст содержит слова и целые числа произвольного порядка. Найти сумму включенных в текст чисел.

Лабораторная работа №9

Файлы данных

Теоретическое введение. *Файл данных* — это совокупность (последовательность) компонент, имеющая имя, расположенная на внешнем носителе. Файлы могут быть объединены в каталоги (директории, папки), также имеющие имя. Использование файлов данных позволяет хранить данные на внешнем носителе, обрабатывая при необходимости порциями (например, при больших объемах данных), позволяет многократно использовать один и тот же набор данных (например, при отладке), позволяет использовать результаты выполнения одной программы (формируя из них файл) как входные данные при выполнении другой программы и т. п.

Файлы и потоки

Для существующего файла данных, который хранится на внешнем носителе и имеет имя, при обращении к файлу создается поток с целью сохранения данных в резервном хранилище. Резервное хранилище — это устройство хранения информации, например, диск.

Поток — это абстракция последовательности байтов, например файл или другое устройство, предоставляющее данные. Класс `Stream` (поток) и его производные классы предоставляют универсальное представление различных типов ввода и вывода, избавляя программиста от необходимости знания отдельных сведений операционной системы и базовых устройств.

Потоки включают три основные операции:

1. Чтение из потока — это перенос информации из потока в структуру данных, такую как массив байтов.
2. Запись в поток — это передача данных из структуры данных в поток.
3. Потоки также могут поддерживать поиск.

Программы, составленные на языке C#, работают с каталогами, файлами и потоками при помощи специально предназначенных для этого классов, входящих в состав библиотеки классов Microsoft .NET Framework и содержащихся в пространстве имен `System.IO`, которое необходимо подключить, чтобы обеспечить доступ к классам, определенным для потоков ввода-вывода (см. пример 9.1).

Для работы с папками и файлами предназначены следующие основные классы:

`Directory` предоставляет статические методы операций создания, перемещения и перечисления в директориях и поддиректориях. Класс `DirectoryInfo` предоставляет методы экземпляра.

`File` предоставляет статические методы для создания, копирования, удаления, перемещения и открытия файлов, а также помогает при создании объектов `FileStream`. Класс `FileInfo` предоставляет методы экземпляра.

Для работы с потоками предназначены следующие основные классы:

`FileStream` предоставляет поток в файле, поддерживая операции чтения и записи.

Класс `StreamReader` считывает символы из потоков с учетом кодировки, класс `StreamWriter` записывает символы в потоки, используя кодировку для преобразования символов в байты (см. пример 9.1).

Стандартные потоки связаны, как правило, с консолью и клавиатурой. Для вывода данных в стандартный поток вывода и для ввода из стандартного потока ввода используются методы класса `Console`: `Console.ReadLine`, `Console.Write` и `Console.WriteLine`. Эти методы и использовались до сих пор во всех примерах программ.

Количество классов, предназначенных для работы с файлами, достаточно велико. Здесь будут рассмотрены только классы, предназначенные для чтения из текстового файла или записи в текстовый файл: `StreamReader` и `StreamWriter`.

Для ввода из файла (созданного заранее в текстовом редакторе) необходимо вначале открыть поток класса `StreamReader`, связав его с файлом. В приведенном ниже примере файл, из которого предполагается считывать данные, расположен по адресу `C:\st\Koord.txt` (это полный путь к файлу). Открытие потока и его привязка к файлу осуществляются с помощью конструктора (возможны и другие способы, которые здесь не рассматриваются).

```
StreamReader sr = new StreamReader(path);
```

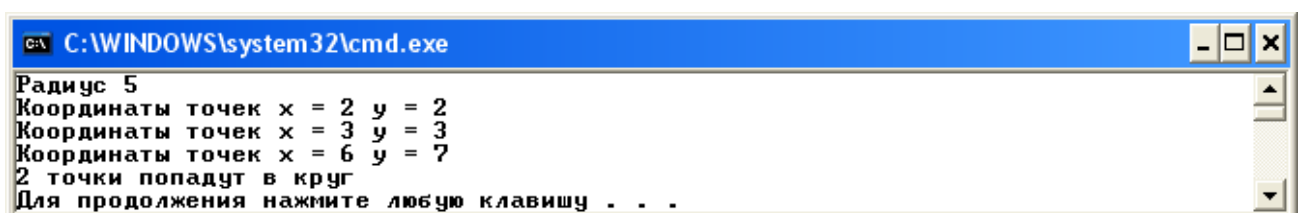
Здесь `sr` – экземпляр класса `StreamReader`, а аргумент `path` передает конструктору строку, содержащую полный путь к файлу (в качестве аргумента можно использовать и константу, содержащую полный адрес файла). Далее строки из файла (в программе это поток `sr`) по очереди считываются в переменную `line`, из которой далее как обычно извлекаются отдельные значения.

После окончания работы с объявленным потоком, его следует закрыть методом `Close`:

```
sr.Close();
```

Пример 9.1. Координаты произвольного количества точек на плоскости размещены в файле Koord.txt на диске C в папке (директории) st по два числа (значения x и y) в строке. В первой строке файла размещено одно число — радиус окружности r . Требуется определить, сколько точек попадет в круг радиуса r .

```
using System;
using System.IO;
class Program
{
    static void Main()
    {
        string path = "C:\\st\\Koord.txt";
        StreamReader sr = new StreamReader(path);
        int n = 0;
        string line;
        line = sr.ReadLine();
        int r = int.Parse(line);
        Console.WriteLine("Радиус {0}", r);
        while ((line = sr.ReadLine()) != null)
        {
            string[] koord = line.Split(' ');
            int x = int.Parse(koord[0]);
            int y = int.Parse(koord[1]);
            Console.WriteLine("Координаты точек x = {0} y = {1}", x, y);
            if (x * x + y * y < r * r) n = n + 1;
        }
        sr.Close();
        Console.WriteLine("{0} точки попадут в круг ", n);
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
Радиус 5
Координаты точек x = 2 y = 2
Координаты точек x = 3 y = 3
Координаты точек x = 6 y = 7
2 точки попадут в круг
Для продолжения нажмите любую клавишу . . .
```

З а м е ч а н и е . В программе в адресе файла вместо одной наклонной черты используется две. Одна наклонная черта в строке могла бы восприниматься как первый символ управляющей последовательности. Использование двух наклонных позволяет избежать этой двусмысленности. Теперь наклонная черта будет восприниматься как символ строки, а не как управляющая последовательность (см. Часть 1). В С# предусмотрен также способ объявления строки, в которой все символы между кавычками трактуются как часть строки. Это специальное объявление – буквальная строка – задается указанием символа @ перед всей строкой и обычно используется для задания пути к файлу. С использованием этого объявления задание строки path может выглядеть так

```
string path = @"C:\st\Koord.txt";
```

При выводе в файл необходимо выполнить аналогичные действия: открыть поток класса StreamWriter, задав имя потока и связав его с файлом, предназначенным для размещения выводимых результатов, вывести в этот поток (т.е. в указанный файл) необходимые результаты и закрыть поток оператором Close() .

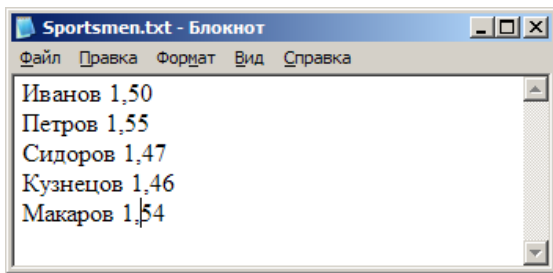
Если в примере 9.1 предполагается также вывод результата в файл, то необходимо добавить следующие инструкции

```
string path1 = "c:\\st\\Koord1.txt";  
StreamWriter sw = new StreamWriter(path1);  
sw.WriteLine(n);  
sw.Close();
```

Далее рассмотрим пример, в котором осуществляется ввод из файла исходных данных и вывод результатов в файл с учетом регионального стандарта (см. Приложение 2).

Пример. 9.2. Протокол соревнований по прыжкам в высоту содержит список фамилий и результатов (одна попытка) в порядке стартовых номеров. Получить итоговую таблицу, содержащую фамилии и результаты в порядке занятых мест. Количество спортсменов не более 30. Для размещения исходных данных используется массив структур. Структура содержит информацию – фамилия и результат спортсмена. Ввод данных осуществлять из заранее подготовленного файла, вывод итоговой таблицы осуществлять в файл. (В примере 6.1 эта же задача решена без использования файлов данных.)

Исходный файл

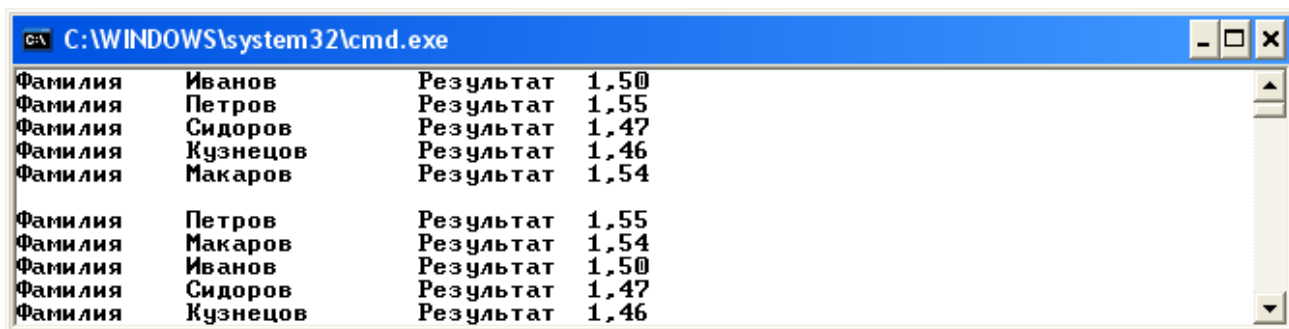


```
using System;
using System.Text;
using System.IO;
namespace ConsoleApplication1
{
    struct Sportsmen
    {
        public string famile;
        public double rez;
    }
    class Program
    {
        static void Main(string[] args)
        {
            Sportsmen[] sp = new Sportsmen[5];
            string line;
            string path = "c:\\st\\Sportsmen.txt";
            //кодировка страницы операционной системы Windows для
            //кириллицы имеет идентификатор 1251
            StreamReader sr = new
StreamReader(path, Encoding.GetEncoding(1251));
            int i = 0;
            while ((line = sr.ReadLine()) != null)
            {
                string[] sports = line.Split(' ');
                sp[i].famile = sports[0];
                sp[i].rez = double.Parse(sports[1]);
                Console.WriteLine("Фамилия {0}\\t Результат
{1:f2}", sp[i].famile, sp[i].rez);
                i++;
            }
        }
    }
}
```

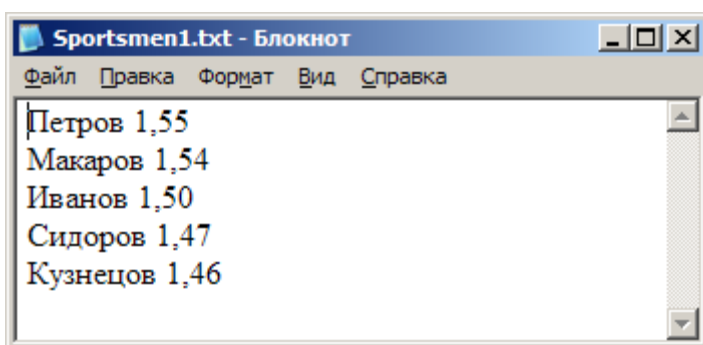
```

    }
    sr.Close();
    for (i = 0; i < sp.Length - 1; i++)
    {
        double amax = sp[i].rez;
        int imax = i;
        for (int j = i + 1; j < sp.Length; j++)
        {
            if (sp[j].rez > amax)
            {
                amax = sp[j].rez;
                imax = j;
            }
        }
        Sportsmen temp;
        temp = sp[imax];
        sp[imax] = sp[i];
        sp[i] = temp;
    }
    Console.WriteLine();
    for (i = 0; i < sp.Length; i++)
    {
        Console.WriteLine("Фамилия      {0}\t  Результат
{1:f2}", sp[i].famile, sp[i].rez);
    }
    string path1 = "c:\\st\\Sportsmen1.txt";
    StreamWriter sw = new StreamWriter(path1);
    for (i = 0; i < sp.Length; i++)
    {
        sw.WriteLine("{0} {1:f2}", sp[i].famile,
sp[i].rez);
    }
    sw.Close();
}
}
}

```



Файл с результатами



З а м е ч а н и е . Дополнительно о региональных стандартах см. в Приложении 2.

Вопросы для самопроверки

1. Что такое файл, директория, поддиректория?
2. В чем преимущества использования файлов для ввода и вывода?
3. Что такое поток? Какие операции определены для потока?
4. Открытие потока для чтения и его привязка к файлу.
5. Что такое «полный путь к файлу»?
6. Считывание из файла в переменные программы.
7. Открытие потока для вывода и привязка его к файлу, предназначенному для вывода результатов.
8. Закрытие потоков.

Задания для самостоятельного выполнения

Выполнить задания к ЛР 7 с вводом исходных данных из файла и формированием файлов с результатами выполнения программы.

Лабораторная работа №10

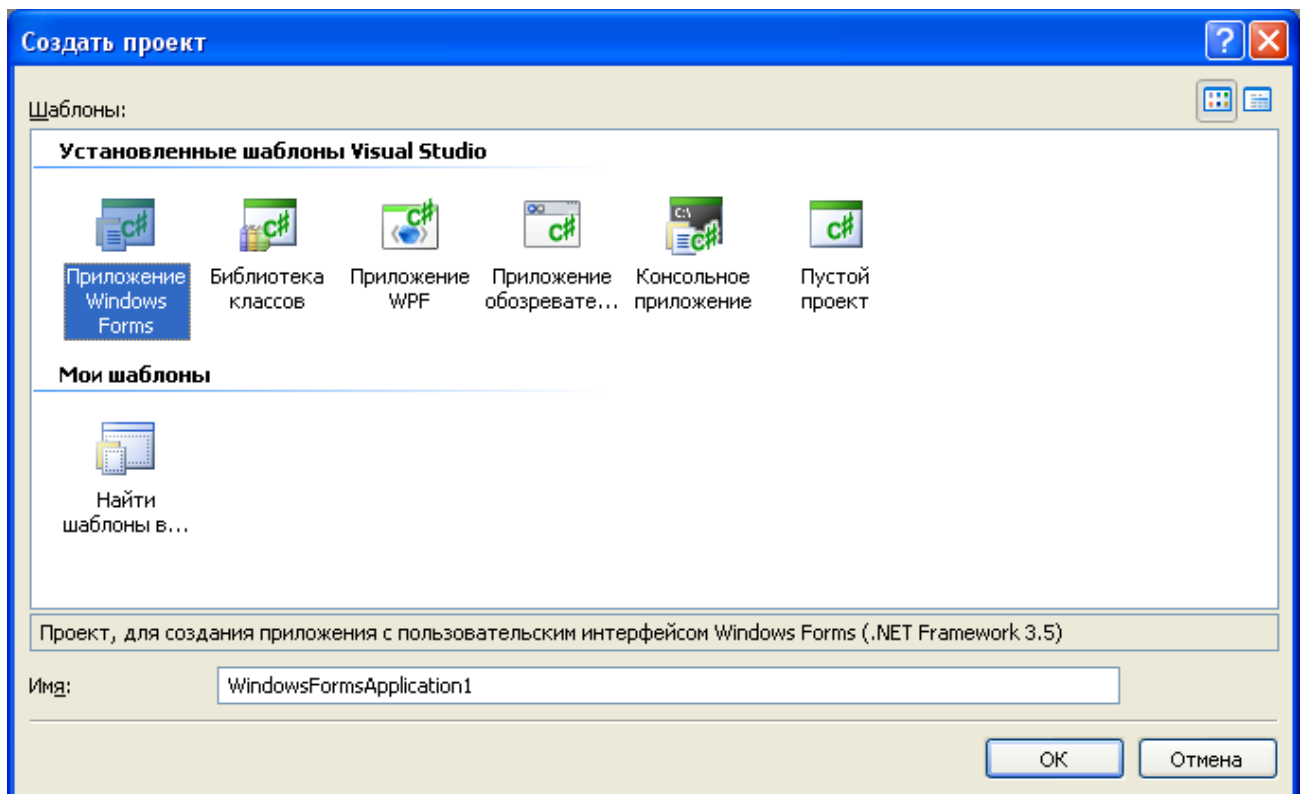
Разработка приложений с графическим интерфейсом пользователя.

Экранные формы

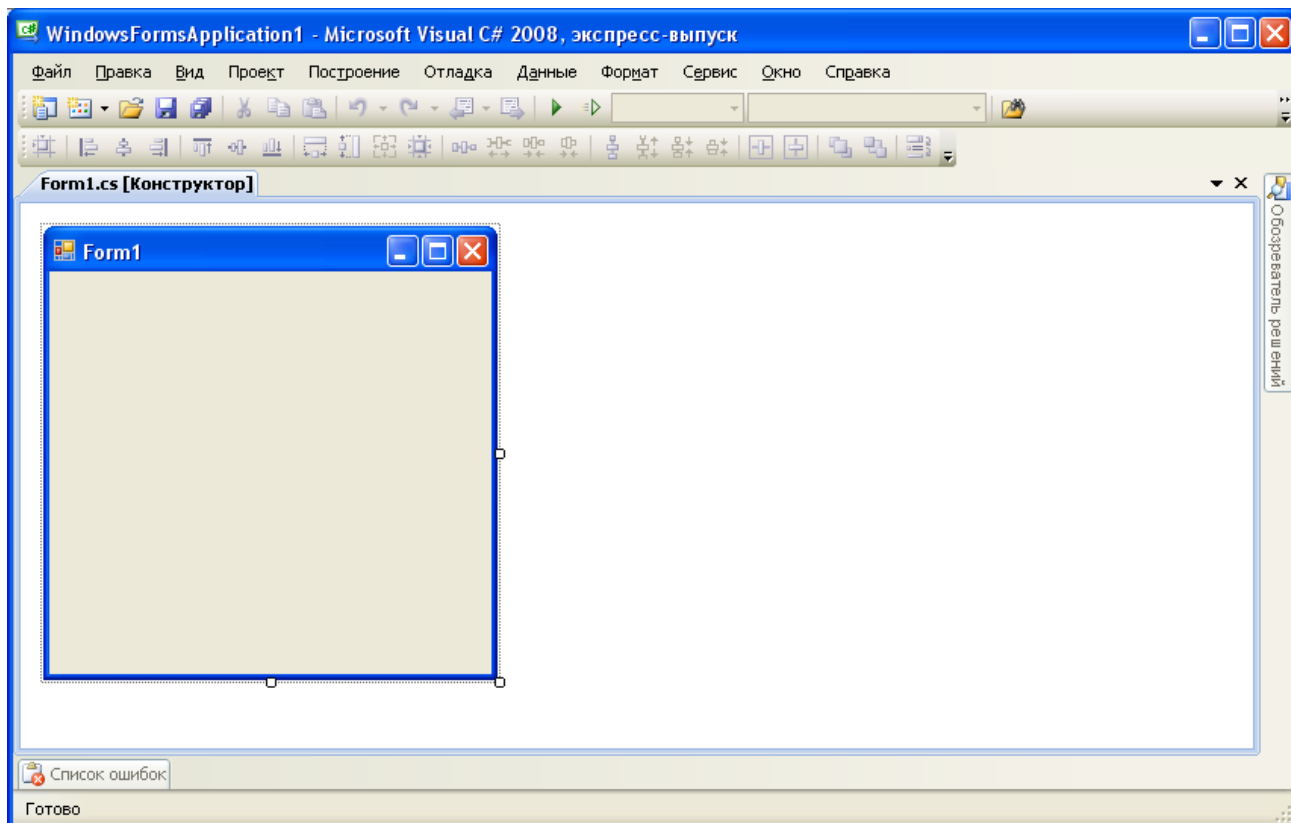
Теоретическое введение. При создании программы вместо стандартного окна для ввода и вывода можно использовать экранные формы, создаваемые специально для конкретной программы. Это позволяет обеспечить ввод данных и вывод результатов в том виде, который требуется в данной задаче, что создает необходимую гибкость и удобство в работе. Элементы управления, помещаемые на форму, обеспечивают возможность вызова метода, связанного программно с этим элементом, в любой удобный пользователю момент, что делает выполнение программы более наглядным.

Далее на наглядных примерах рассматриваются основные возможности, предоставляемые при использовании экранных форм, и их реализация.

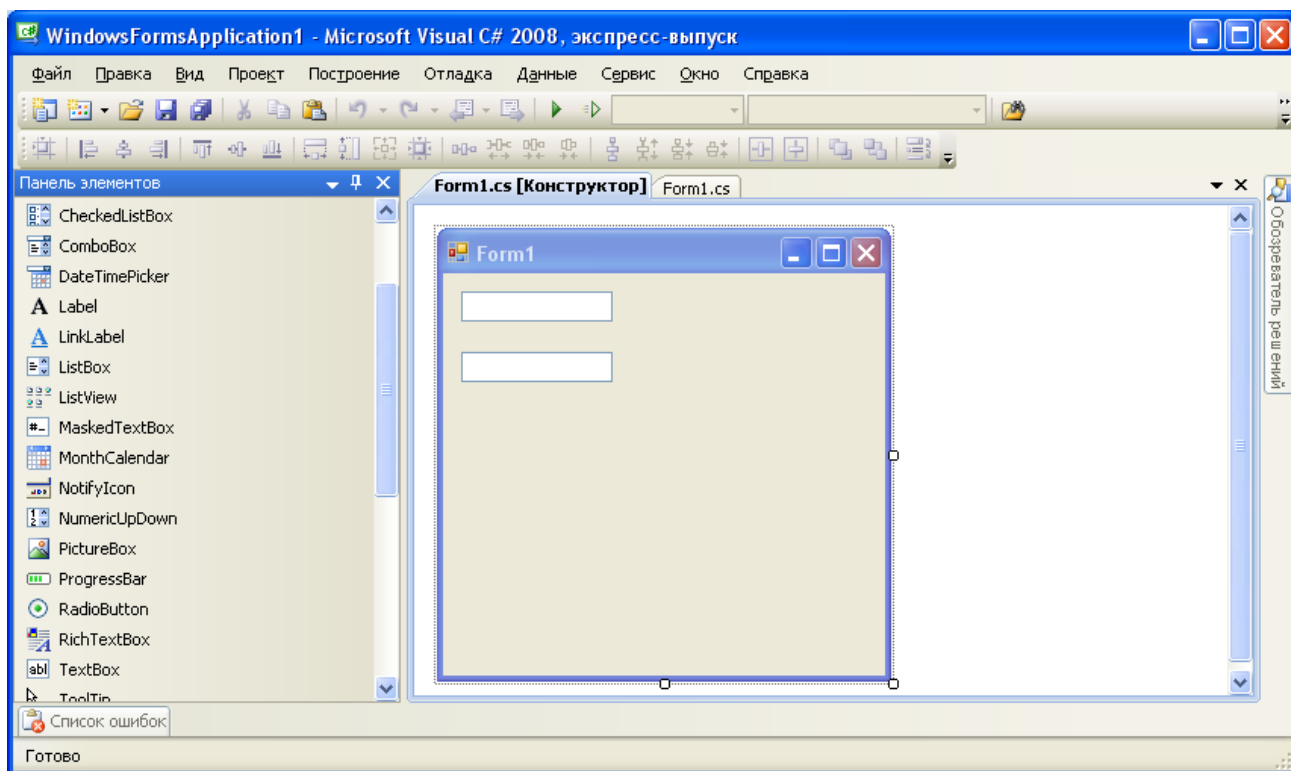
- Войдите в C# Visual. В меню Файл выберите команду «Создать проект».



- Выберите шаблон Приложение Windows Forms (в поле «Имя» можно ввести любое имя проекта вместо стандартного) и нажмите кнопку «ОК».
- Откроется конструктор Windows Forms с формой Windows. Это пользовательский интерфейс для создаваемого приложения.

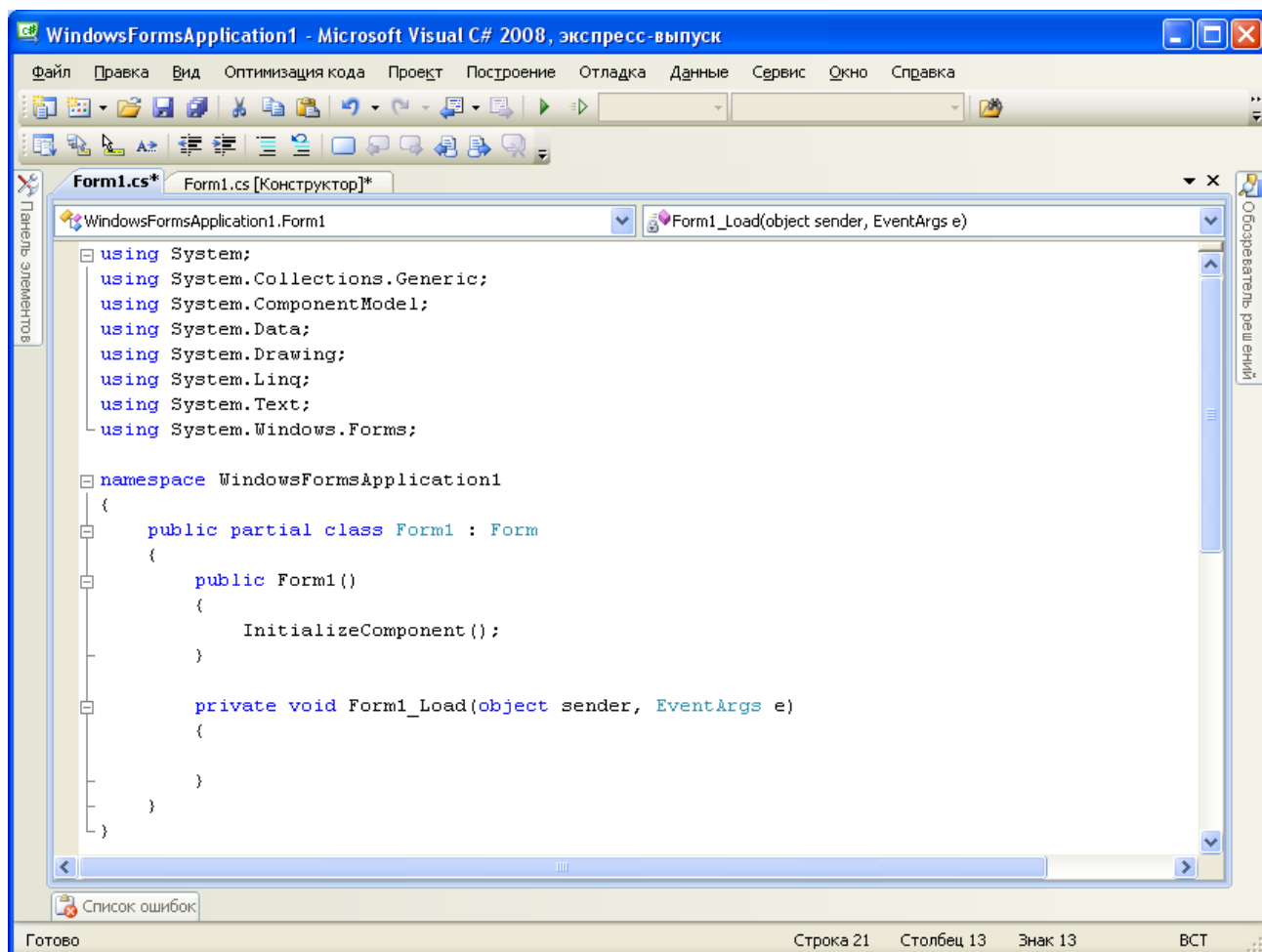


- В меню **Вид** выберите команду **Панель элементов**, чтобы открыть список элементов управления.
- Разверните список **Стандартные элементы управления** и перетащите два элемента управления **TextBox** на форму.



Дважды щелкните на форму Windows (Form1), чтобы открыть редактор кода. Visual C# вставил метод с именем Form1_Load, который выполняется при загрузке формы, –

обработчик события Load, связанного с запуском приложения. Откроется редактор кода, при этом положение курсора окажется внутри обработчика событий. Обработчик событий — это метод, определяющий действия, которые требуется выполнить при возникновении события. События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций.



З а м е ч а н и е . При двойном щелчке на форму подпись метода и его содержимое (пустые кавычки { })

```
private void Form1_Load(object sender, EventArgs e)
{

}
```

генерируются автоматически. Одновременно автоматически генерируется код для вызова метода, который помещается в файл Form1.Designer.cs. Поэтому, если попытаться просто набрать код (без щелчка мышью) самостоятельно, это не даст нужного эффекта (код для вызова метода не будет сгенерирован) и приведет к ошибке.

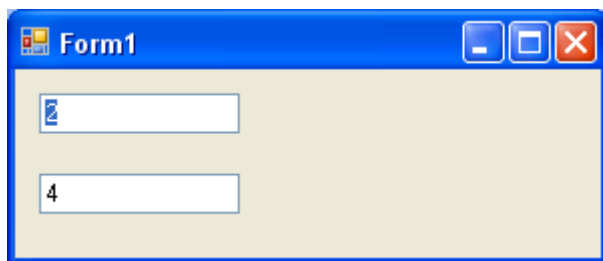
Работа с элементом управления TextBox

Текстовые поля форм Windows Forms используются для приема данных, вводимых пользователем, или для отображения текста. В текстовых полях можно выводить несколько строк текста, размещать текст в соответствии с размером элемента управления и применять основные элементы форматирования. Для вывода числовых данных в текстовое окно необходимо получить их строковое представление. После ввода числовых данных в текстовое окно необходимо использовать метод `Parse` для получения числового значения из строкового представления.

Пример 10.1.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            int b = 2;
            textBox1.Text = b.ToString();
            int a = int.Parse(textBox1.Text);
            a += 2;
            textBox2.Text = a.ToString();
        }
    }
}
```

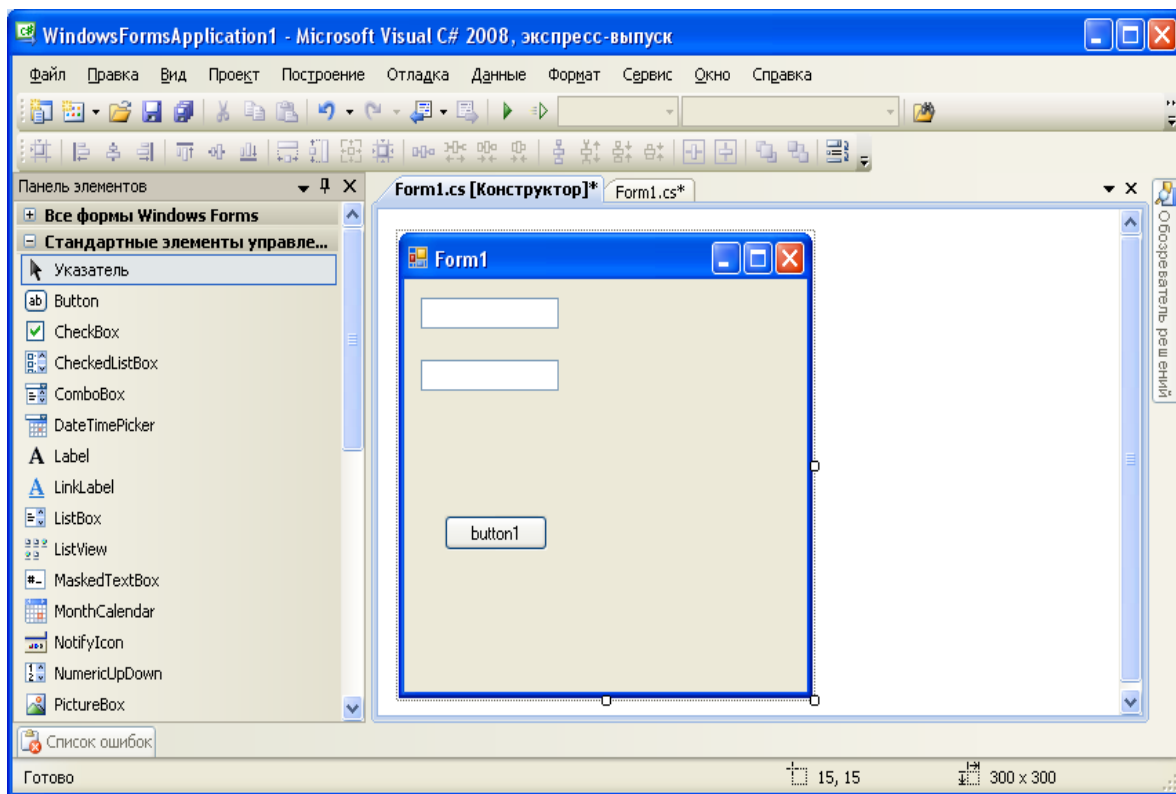
`TextBox` – это класс пространства имен: `System.Windows.Forms`. Экземпляр класса `textBox1` создается при добавлении элемента управления на форму. Здесь `Text` – это свойство, которое имеет тип `string` и возвращает или задает текст.



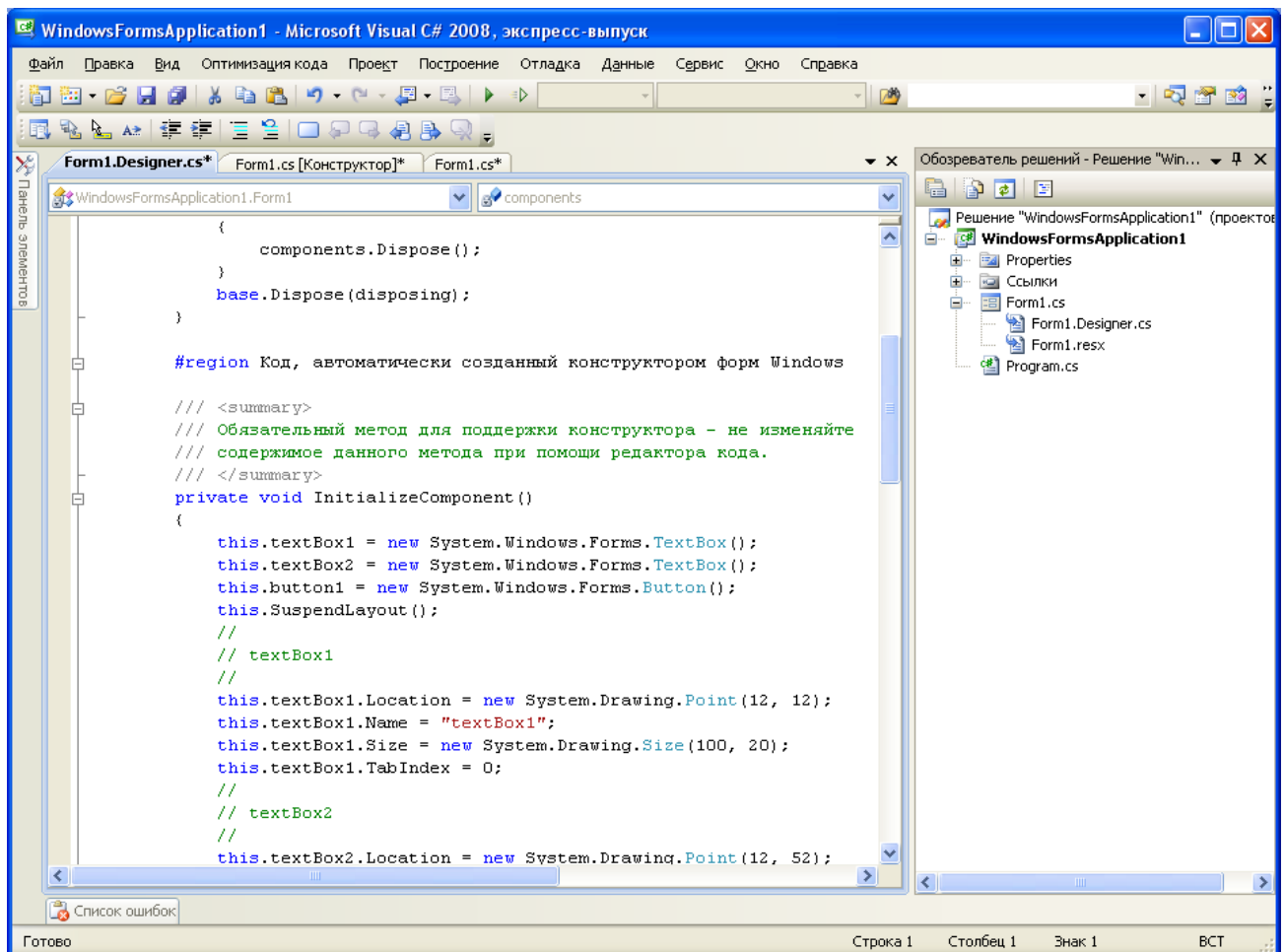
*Работа с элементом управления **Button***

Button – класс пространства имен System.Windows.Forms, представляет элемент управления Windows «Кнопка».

1. В меню **Вид** выберите команду **Панель элементов**, чтобы открыть список элементов управления.
2. Разверните список **Стандартные элементы управления** и перетащите элемент управления **Button** на форму.



При добавлении кнопки на форму автоматически создается экземпляр класса Button с именем button1. Можно открыть файл Form1.Designer.cs и посмотреть соответствующий код.

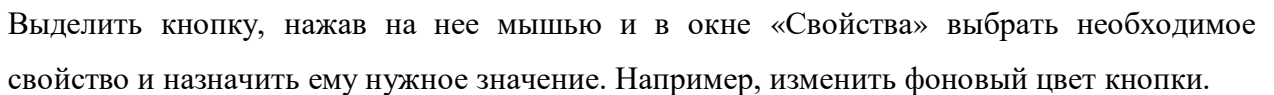


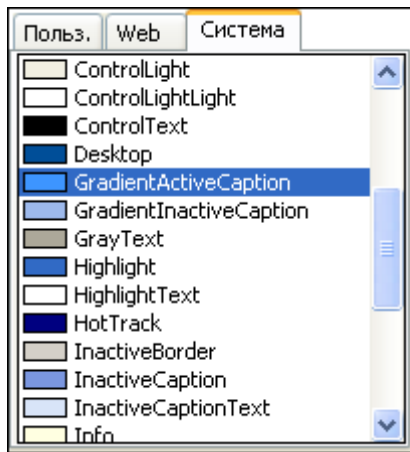
3. Дважды щелкните на кнопку, чтобы открыть редактор кода. Visual C# вставил метод с именем `button1_Click`, который выполняется при нажатии на кнопку – обработчик события `Click`.

Пример 10.2. Изменение цвета кнопки после нажатия на нее.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
```

Изменить свойства элемента управления можно как программно, так и с использованием интегрированной среды разработки. В последнем случае необходимо перейти из окна редактора кода в конструктор Windows Forms с формой Windows и в меню Вид выбрать команду «Окно свойств».





Вернемся к примеру, демонстрирующему работу с текстовым окном.

Пример 10.3.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Text = "2";
            int a = int.Parse(textBox1.Text);
            a += 2;
            textBox2.Text = a.ToString();
        }
    }
}
```

Если пользователю необходимо вводить исходные данные в элемент управления формы Текстовое поле, то необходимо изменить пример и перенести код из обработчика события формы Load в обработчик события Click кнопки.

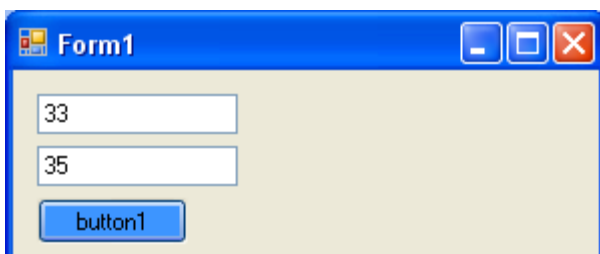
Пример 10.4

```

using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }
        private void button1_Click(object sender, EventArgs e)
        {
            int a = int.Parse(textBox1.Text);
            a += 2;
            textBox2.Text = a.ToString();
        }
    }
}

```

Щелчок мышью по кнопке `button1` является событием, которое вызывает выполнение метода `button1_Click`, соответствующего этому событию. При этом текст, введенный в текстовое поле `textBox1`, преобразуется в целое число и присваивается переменной `a`, значение которой далее увеличивается на 2 и выводится в текстовое поле `textBox2`.



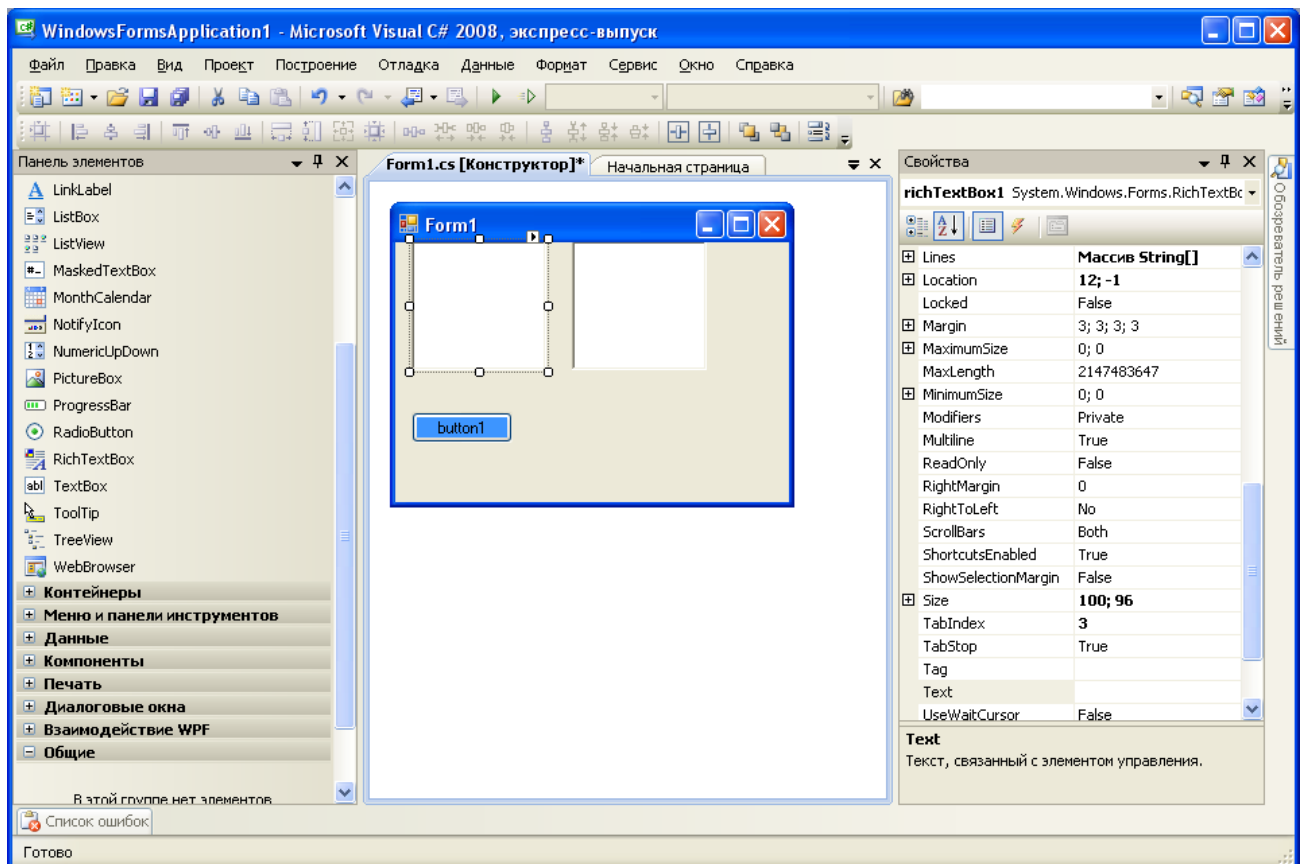
Работа с элементом управления RichTextBox

Элемент управления Windows Forms `RichTextBox` используется для отображения, ввода и изменения текста (если необходимо, с форматированием). Методы этого класса

предоставляют возможности схожие с возможностями текстовых редакторов, например, таких как Microsoft Word. По сравнению с классом `TextBox` он обладает более широкими возможностями. В частности, позволяет считывать данные из файла, а также выводить в файл.

Откройте окно конструктора Windows Forms с формой `Windows`

1. В меню **Вид** выберите команду **Панель элементов**, чтобы открыть список элементов управления.
2. Разверните список **Стандартные элементы управления** и перетащите два элемента управления `RichTextBox` и кнопку на форму.



3. Дважды щелкните на кнопку, чтобы Visual C# вставил метод с именем `button1_Click` и дважды щелкните на форму `Windows (Form1)`, чтобы Visual C# вставил метод с именем `Form1_Load`.

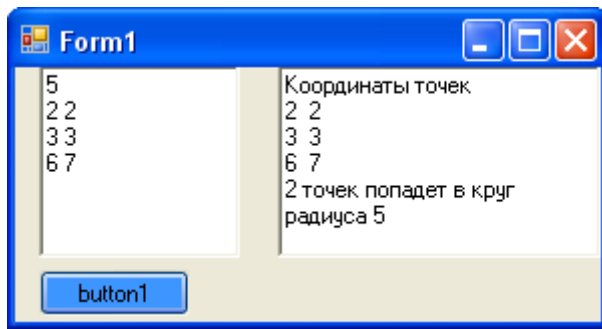
Пример 10.5. Координаты произвольного количества точек на плоскости размещены в файле `Koord.txt`, сохраненном на диске *C* в папке (директории) *st* по два числа (значения *x* и *y*) в строке. В первой строке файла размещено одно число — радиус окружности *r*. Требуется определить, сколько точек попадет в круг радиуса *r*.

```
using System.Text;
```

```

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            string path = "c:\\st \\Koord.txt";
            //метод читает данные из файла в элемент управления
            richTextBox1.LoadFile(path, RichTextBoxStreamType.PlainText);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string text = richTextBox1.Text;
            string[] s = text.Split(' ', '\n');
            int r = int.Parse(s[0]);
            int n = 0;
            richTextBox2.AppendText("Координаты точек" + "\n");
            for (int i = 1; i < 6; i += 2)
            {
                int x = int.Parse(s[i]);
                int y = int.Parse(s[i + 1]);
                richTextBox2.AppendText(x.ToString() + "
+y.ToString() + "\n");
                if (x * x + y * y < r * r) n = n + 1;
            }
            richTextBox2.AppendText(n.ToString() + " точек
попадет в круг радиуса " + r.ToString() + "\n");
        }
    }
}

```



Пример 10.6. Превращение вещества A в вещество B при химической реакции описывается следующими формулами:

$$C_a = C_a^0 \cdot e^{-k \cdot t},$$

$$C_b = C_b^0 + C_a^0 (1 - e^{-k \cdot t}),$$

где C_a^0 и C_b^0 - начальные концентрации веществ A и B ,

C_a и C_b - концентрации этих же веществ в момент времени t ,

k - константа скорости химической реакции.

Построить таблицу распределения концентраций веществ A и B от начала реакции до момента полного превращения вещества A в вещества B .

Полагая $C_b^0 = 0$, в текстовое поле `textBox1` будем вводить начальную концентрацию вещества A , а в текстовое поле `textBox2` – константу скорости химической реакции k .

В текстовое поле `RichTextBox1` будем выводить время реакции t , а в текстовое поле `RichTextBox2` – концентрации веществ A и B в момент времени t от начала реакции и до полного превращения вещества A в вещество B .

```
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

```

private void Form1_Load(object sender, EventArgs e)
{
}

private void button1_Click(object sender, EventArgs e)
{
    int t = 0;
    double Ca0, Ca, Cb, k;
    richTextBox1.AppendText("Время" + "\n");
    richTextBox2.AppendText("Вещество A" + "    Вещество
B" + "\n");
    Ca0 = double.Parse(textBox1.Text);
    k = double.Parse(textBox2.Text);
    do
    {
        Ca = Ca0 * Math.Exp(-k * t);
        Cb = Ca0 * (1 - Math.Exp(-k * t));
        richTextBox1.AppendText(t.ToString() + "\n");
        richTextBox2.AppendText(Ca.ToString("f2") + "
" + Ca.ToString("f2") + "\n");
        t++;
    } while (Math.Abs(Ca) > 0.01);
}
}

```

Время	Вещ. А	Вещ. В
0	1000,00	1000,00
1	670,32	670,32
2	449,33	449,33
3	301,19	301,19
4	201,90	201,90
5	135,34	135,34

button1

1000 0.4

Пример 10.7.. Коэффициент диффузии газа D вычисляется по формуле:

$$D = \frac{1}{3} \cdot \sqrt{\frac{8 \cdot R \cdot T}{\pi \cdot \mu}} \cdot \frac{k \cdot T}{\sqrt{2} \cdot \pi \cdot \sigma^2 \cdot p},$$

где μ – молярная масса газа,

σ – диаметр молекул,

k – постоянная Больцмана,

R – газовая постоянная,

T – температура газа,

p – давление газа.

Построить таблицу изменения коэффициента диффузии D газа от температуры T , изменяющейся от $100\text{ }^{\circ}\text{K}$ до $600\text{ }^{\circ}\text{K}$ с шагом $20\text{ }^{\circ}\text{K}$.

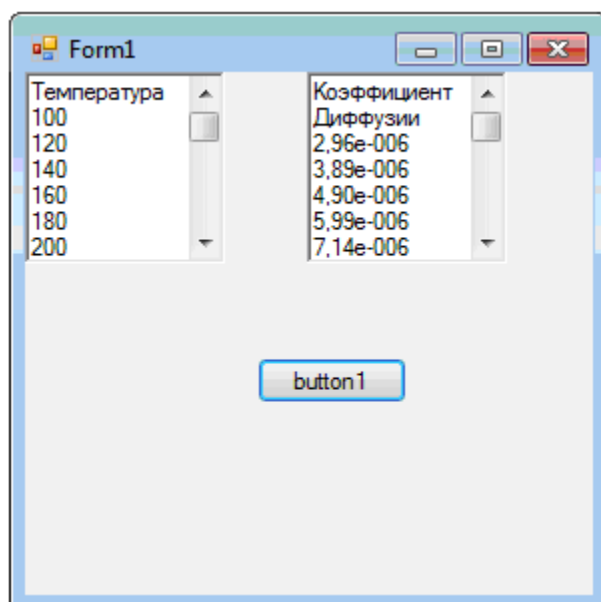
Для азота: $\mu = 0,028\text{ кг/моль}$, $k = 1,38 \cdot 10^{-23}\text{ Дж/К}$, $\sigma = 0,3\text{ нм}$, $p = 10^5\text{ Па}$, $R = 8,31\text{ Дж/(моль}\cdot\text{K)}$.

```
using System;
using System.Drawing;
WindowsFormsApplication2
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
```

```

{
    public Form1()
    {
        InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
    }
    private void button1_Click(object sender, EventArgs e)
    {
        int T, Tn = 100, Tk = 600, dT;
        double D, mu = 0.032, R = 8.31, sgm = 0.3e-9, k =
1.38e-23, p = 1e5;
        dT = (Tk - Tn) / 25;
        richTextBox1.AppendText("Температура" + "\n");
        richTextBox2.AppendText("Коэффициент" + "\n");
        richTextBox2.AppendText("Диффузии" + "\n");
        for (T = Tn; T <= Tk; T = T + dT)
        {
            D = Math.Sqrt(8 * R * T / (Math.PI * mu)) / 3 * k * T
/ (Math.PI * Math.Sqrt(2) * sgm* sgm * p);
            richTextBox1.AppendText(T.ToString()+ "\n");
            richTextBox2.AppendText(D.ToString("e2") + "\n");
        }
    }
}

```

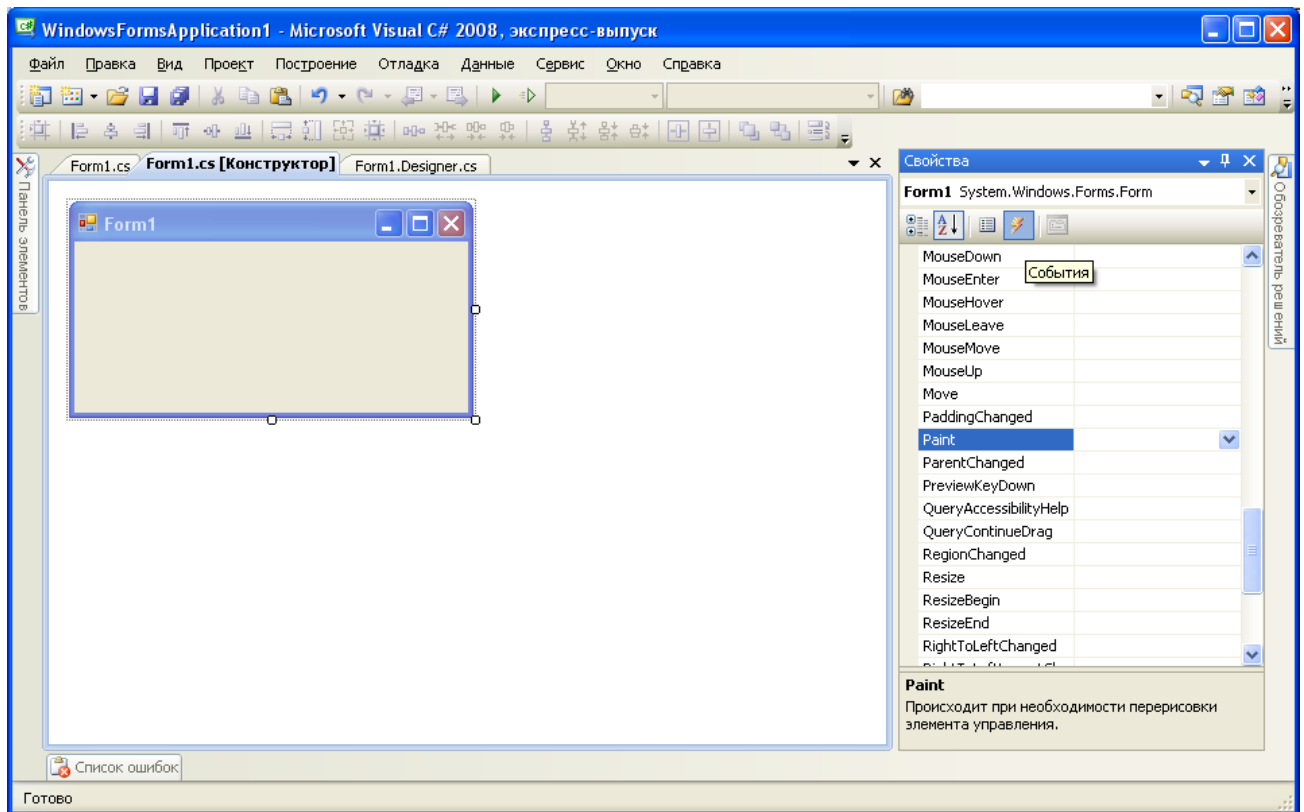


Создание объекта Graphics пространства имен System.Drawing для рисования

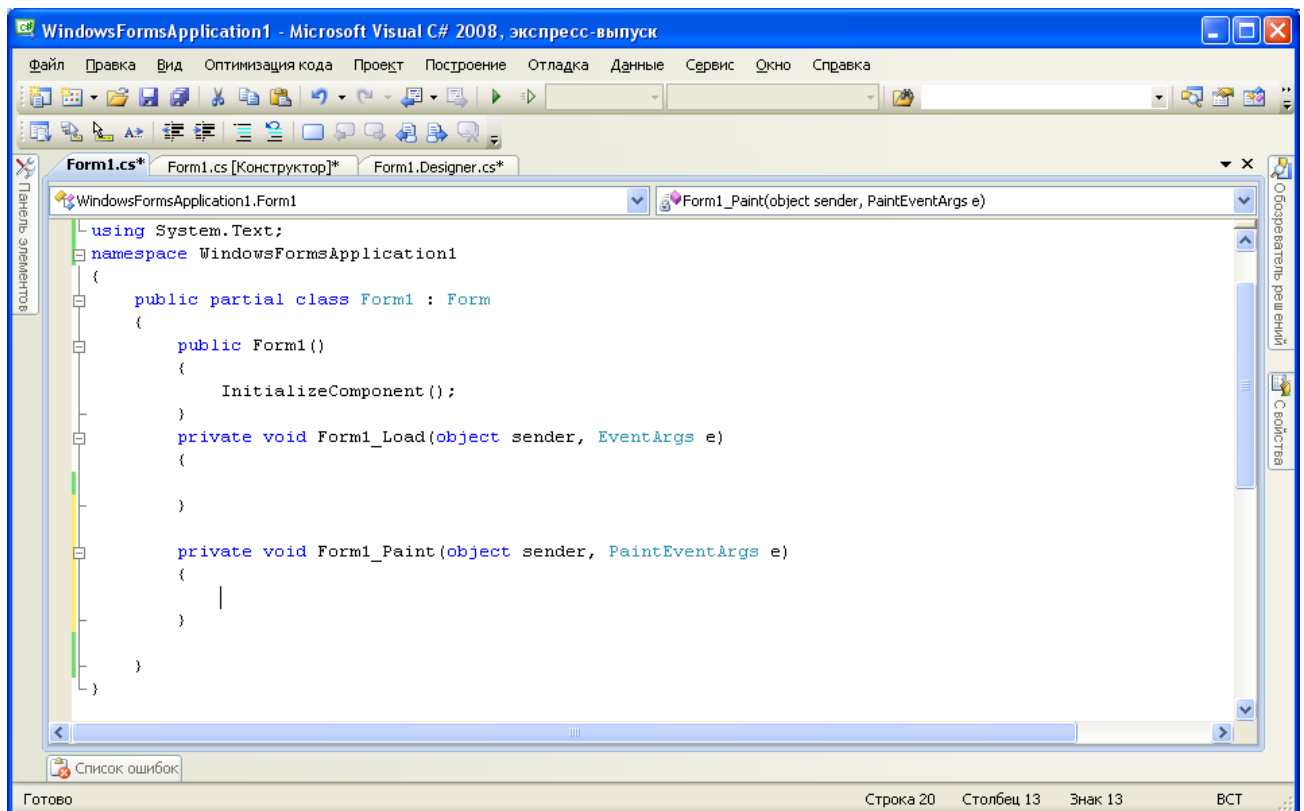
Класс `Graphics` является основой интерфейса GDI+ (GDI+ специальная библиотека). Этот класс непосредственно выполняет рисование прямых и кривых линий, геометрических фигур, вывод рисунков и текста.

Перед тем как рисовать линии и фигуры, отображать текст, выводить изображения и управлять ими, необходимо создать объект класса `Graphics`. Объект `Graphics` представляет поверхность рисования и является объектом, который используется для создания графических изображений.

Откройте окно конструктора Windows Forms с формой Windows. Выделите форму. В окне Свойства нажмите значок событие ⚡.



Выберите из списка событие Paint и дважды нажмите мышью строку списка

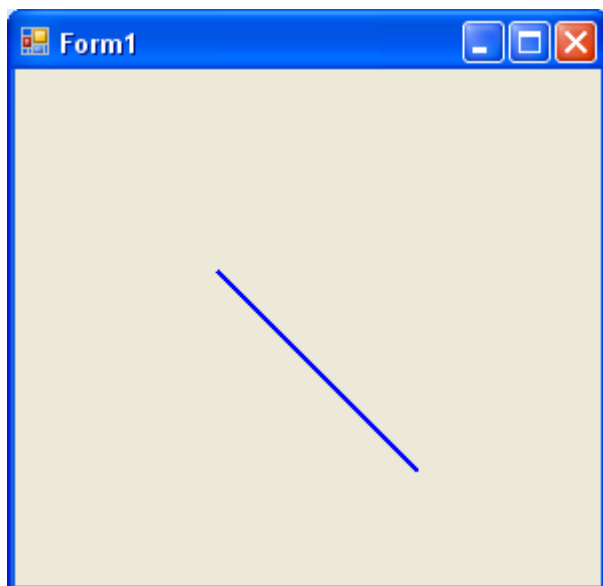


Visual C# вставил метод с именем **Form1_Paint**, который выполняется при перерисовке элемента управления. Далее необходимо получить ссылку на объект **Graphics** из объекта **PaintEventArgs** в событии **Paint**:

1. Объявите объект **Graphics**.
2. Присвойте переменной ссылку на объект **Graphics**, передаваемый как часть **PaintEventArgs**. **PaintEventArgs** – класс, предоставляет данные для события **Paint**
3. Вставьте код для рисования формы или элемента управления.

Пример 10.8. Рисование линии на форму.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.DrawLine(new Pen (Color.Blue, 2.0f), 100, 100, 200, 200);
        }
    }
}
```



З а м е ч а н и я . 1. При создании экземпляра `g` класса `Graphics` ему передается ссылка на форму, куда нужно выводить рисунок (график). Информация о форме содержится в свойстве `Graphics` экземпляра `e` класса `PaintEventArgs`.

2. Метод `DrawLine` предназначен для вывода линии, `Pen` – класс, предоставляющий перо для вычерчивания линии. Первый аргумент задает цвет линии, второй – ее толщину, выраженную числом типа `float`. Остальные параметры задают координаты начала и конца линии.

При рисовании графика необходимо иметь в виду, что координата $(0, 0)$ формы находится в верхнем левом углу. Перенести точку $(0, 0)$ начала координат можно используя метод `TranslateTransform`, указав в качестве аргументов величину сдвига вдоль осей X и Y . Ось OY направлена вниз. Поэтому необходимо отразить график относительно оси OY , поставить перед координатой y знак минус.

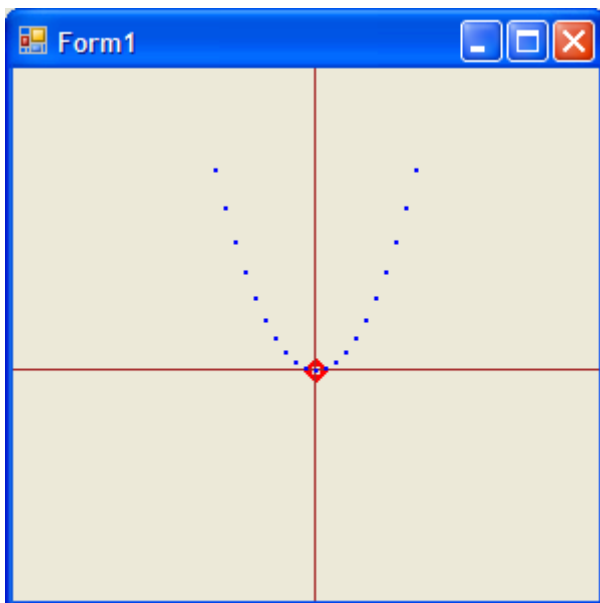
Пример 10.9. Построить по точкам график функции $y = x^2$ при $x = -10, -9, -8, \dots, 10$.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

```

    }
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        int xc = this.Width / 2; //this в данном случае - форма
        int yc = this.Height / 2;
        g.TranslateTransform(xc, yc);
        g.DrawEllipse(new Pen(Color.Red, 8.0f), 0, 0, 1, 1);
        int x, y;
        //вычерчивание осей координат
        g.DrawLine(new Pen(Color.Brown, 1.0f), -200, 0, 200, 0);
        g.DrawLine(new Pen(Color.Brown, 1.0f), 0, -200, 0, 200);
        for (x = -10; x <= 10; x += 1)
        {
            y = x*x;
            g.DrawEllipse(new Pen(Color.Blue, 2.0f), x*5, -y,
1, 1);
        }
    }
}
}
}

```



З а м е ч а н и я . 1. Здесь введен масштаб по оси x , растягивающий ось в 5 раз.

1. Метод `DrawEllipse` рисует окружность, вписанную в квадрат со сторонами 1, 1 (последние два аргумента метода `DrawEllipse`, определяющие толщину точки) для каждой точки графика.

Далее приведены программы графического вывода для примеров 10.6 и 10.7.

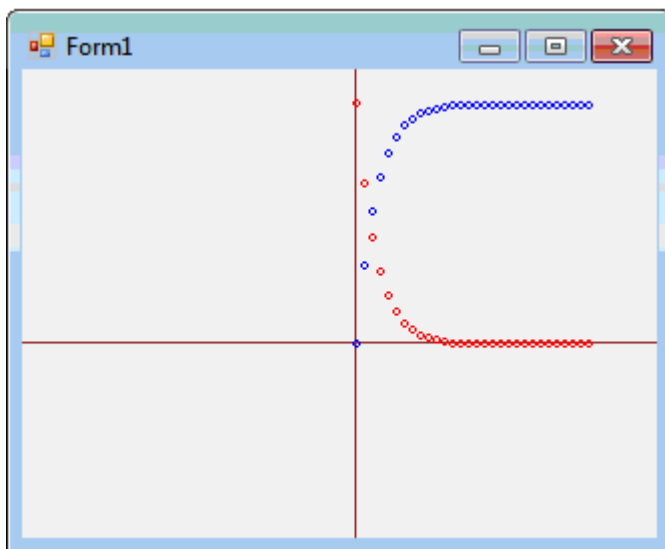
```
using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int x, y, z, t=0;
            int xc = this.Width / 2;
            int yc = this.Height / 2;
            double Ca, Cb, Ca0 = 1000, Cb0 = 0, k = 0.4;
            g.TranslateTransform(xc, yc);
            g.DrawLine(new Pen(Color.Brown, 1.0f), -200, 0, 200, 0);
            g.DrawLine(new Pen(Color.Brown, 1.0f), 0, -200, 0, 200);
            do
            {
                Ca = Ca0 * Math.Exp(-k * t);
```

```

        Cb = Cb0 + Ca0 * (1 - Math.Exp(-k * t));
        x = (int)(t / 30.0*120);
        y = (int)(Ca / Ca0 *120);
        z = (int)(Cb / Ca0 *120);
        g.DrawEllipse(new Pen(Color.Red, 3.0f), x, -y, 1, 1);
        g.DrawEllipse(new Pen(Color.Blue, 3.0f), x, -z, 1, 1);
        t++;
    } while (Math.Abs(Ca) > 0.01);
}

}
}

```



```

using System;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}

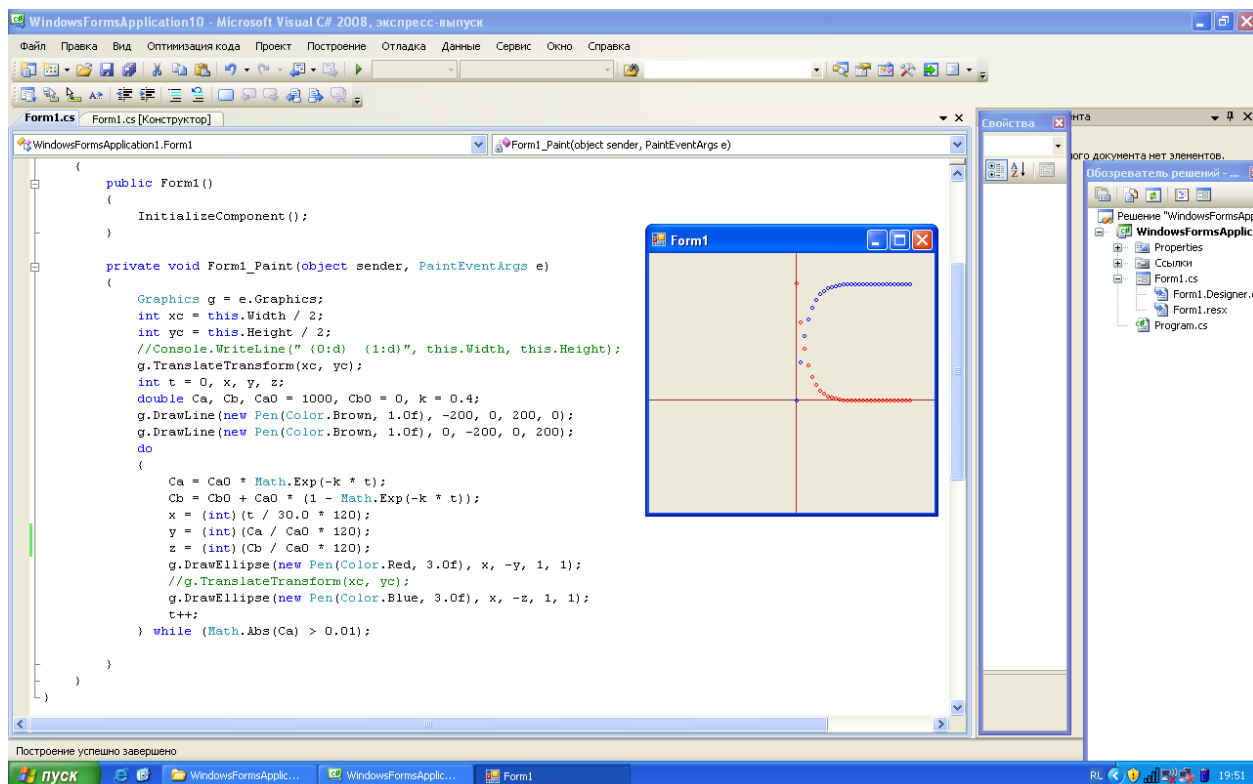
```

```

private void Form1_Load(object sender, EventArgs e)
{
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    int x, y;
    int xc = this.Width / 2;
    int yc = this.Height / 2;
    int T, Tn = 100, Tk = 600, dT;
    double D, mu = 0.028, R = 8.31, sgm = 0.3e-9, k = 1.38e-
23, p = 1e5;
    dT = (Tk - Tn) / 25;
    g.TranslateTransform(xc, yc);
    g.DrawEllipse(new Pen(Color.Red, 8.0f), 0, 0, 1, 1);
    g.DrawLine(new Pen(Color.Brown, 1.0f), -200, 0, 200, 0);
    g.DrawLine(new Pen(Color.Brown, 1.0f), 0, -200, 0, 200);
    for (T = Tn; T <= Tk; T = T + dT)
    {
        D = Math.Sqrt(8 * R * T / (Math.PI * mu)) / 3 * k * T
/ (Math.PI * Math.Sqrt(2) * sgm * sgm * p);
        x = (int)(120 * T / 600);
        y = (int)(120 * D / 4.5e-5);
        g.DrawEllipse(new Pen(Color.Red, 3.0f), x, -y, 1, 1);
    }
}
}
}

```



Вопросы для самопроверки

1. Что такое экранные формы и каковы преимущества их использования?
2. Элементы управления TextBox, Button, RichTextBox. В каких случаях целесообразно их использование?
3. Класс Graphics. Рисование прямых линий.
4. Рисование графиков. Масштабирование при рисовании графиков.

Задания I уровня

Задание I уровня предназначено для приобретения навыков использования текстовых полей форм Windows Forms для ввода и вывода данных.

Составить программы для решения указанных задач. В задачах 1–8 использовать элемент управления «Кнопка» (Button). В задачах 1, 2 использовать элемент управления TextBox. В задачах 3 – 8 использовать элемент управления RichTextBox.

1. Вычислить $c = a + b$, вводя исходные данные в два текстовых поля (TextBox), в третье – вывести результат.
2. Вычислить сумму первых n натуральных чисел, вводя значение n в одно текстовое поле, результат вывести во второе текстовое поле.

3. Решить задачу 1, вводя исходные данные в элемент управления RichTextBox, результат вывести в текстовое поле TextBox.
4. Ввести пять различных чисел в RichTextBox1, разделяя их пробелами. Умножить каждое из этих чисел на 2 и вывести в RichTextBox2, каждое в новую строку, снабдив результат заголовком.
5. Элементы массива размера 6 поместить в RichTextBox1. Найти максимальный элемент массива и вывести его в TextBox.
6. Решить задачу 5, помещая результат с заголовком в RichTextBox.
7. Два массива размера 6 поместить в RichTextBox1 и RichTextBox2. Найти сумму этих массивов, суммируя каждую пару элементов. Результат с заголовком поместить в RichTextBox3.
8. Матрицу размера 4×4 разместить в RichTextBox1. Сформировать массив из сумм элементов строк и поместить его в RichTextBox2.

Задания II уровня

Задание II уровня предназначено для приобретения навыков визуализации графических данных.

Составить программы для решения указанных задач. Построить график функции $z = f(x)$ при $a \leq x \leq b$, разбивая отрезок $[a, b]$ на n частей. График начертить точками или отрезками прямых линий по указанию преподавателя, выполнив предварительно масштабирование. При вычерчивании графика функции предусмотреть вывод координатных осей.



1. $z = 2 \sin x + \sin 2x; a = -\pi, b = \pi, n = 50.$

2. $= \quad = \quad = \quad =$

3. $z = \sin x + 0,5x^2; a = -2\pi, b = 2\pi, n = 40.$

4. $z = \frac{3 \sin x}{x}; a = 0, b = 4\pi, n = 50.$

5. $z = x^2 - 18x + 72; a = 5, b = 20, n = 40.$

6. $z = x^3 + 5x^2 + 14x - 56; a = 1, b = 10, n = 40.$

7. $z = \sqrt{x+1} - 1/x; a = 0,5, b = 1, n = 20.$

8. Физическое, эмоциональное и умственное состояния человека изменяются со дня рождения циклически с периодом 23, 28 и 33 дня соответственно. Состояние для d -го дня со дня рождения определяется по формуле $y = \sin(x)$, где $x = 2\pi(d/x - [d/x])$ ($x = 23, 28$ или 33 для физического, эмоционального и умственного состояния). Нарисовать графики биоритмов на текущую неделю. (При определении количества дней, прошедших со дня рождения, учитывать високосные годы!)

Задания III уровня

Задание III уровня предназначено для приобретения навыков решения задач с использованием диалогового режима. Требуется использования творческого подхода.

Составить схему игры, продумать организацию диалога. Выбрать способ представления данных. Разработать алгоритм, составить программу. В программе предусмотреть наглядный вывод результатов и соответствующих сообщений по ходу игры и после ее окончания.

1. Игра "Скачки" (вариант 1). В игре участвуют 10 наездников; за каждый тур игры каждый из них продвигается вперед на расстояние от 1 до 5 км случайным образом. Длина дистанции - 50 км. Всего проводится 5 заездов, победителю каждого заезда начисляется 5 очков. Просмотр наездников в туре осуществлять последовательно. Победителем считается наездник, набравший наибольшее количество очков во всех заездах. Перед началом заездов участник игры выбирает номер наездника, с которым он будет идентифицироваться во время игры. Количество участников игры не превышает 10, в программе предусмотреть возможность случайного распределения номеров наездников.
2. Игра "Скачки" (вариант 2). В каждом туре с вероятностью 0,1 каждый наездник может упасть, т.е. продвинуться за этот тур на ноль км.
3. Игра "Трек". В велогонке на треке участвуют 20 спортсменов, велогонка длится 20 кругов. На финише каждого круга занявшему 1-е место начисляется 7 очков, 2-е место - 5 очков, 3-е место - 3 очка, 4-е место - 2 очка, 5-е место - 1 очко. Победителем (победителями) считается велосипедист, набравший наибольшее количество очков или победивший в 5 кругах. Перед велогонкой участники игры случайным образом разыгрывают номера, под которыми они выступают.
4. Игра "Гонка с выбыванием". В мотокроссе участвуют 15 спортсменов, участники должны преодолеть 14 кругов. После каждого круга участник, занимающий

последнее место, снимается с соревнований. Перед началом соревнований участники игры случайным образом разыгрывают номера, под которыми они выступают. Количество участников не более 15.

5. Игра "Набери сумму" (вариант 1). Два участника по очереди набирают сумму из слагаемых, которые могут иметь целые значения от K до L и задаются случайным образом. Задача заключается в том, чтобы вовремя прекратить набор суммы, когда она достаточно близка к заданному по условиям игры числу N (или равна ему), но не превосходит его. Выигравшим считается участник, набравший сумму, более близкую к N . Если сумма какого-либо игрока превзойдет N , то он сразу проигрывает, и игра прекращается. В случае равенства сумм у обоих игроков победителем считается набравший сумму вторым. Одним из партнеров является компьютер, другим - человек. Компьютер набирает сумму S , пока $S < N - 5$. Второй партнер может принять решение о прекращении набора перед получением очередного слагаемого.
6. Игра "Набери сумму" (вариант 2). Первым сумму набирает человек. Компьютер при наборе суммы использует следующую стратегию. Она стремится набрать сумму, равную или большую, чем сумма первого игрока. Если суммы равны, выигрыш присуждается компьютеру.
7. Игра "Набери сумму" (вариант 3). Играют два партнера. Компьютер лишь генерирует случайным образом слагаемые суммы и определяет результат игры. Предусмотреть возможность повторного проведения игры и определения общего счета после ее окончания.
8. Игра "Карусель-лото" (вариант 1). За один ход компьютер генерирует случайное целое число в интервале $[0; 36]$. Перед этим участники заказывают одну комбинацию из следующих возможных (стараясь угадать число или интервал, в который число попадает):
 - a) выпадает четное или нечетное число;
 - b) число попадает в интервал $[1, 18]$ или $[19, 36]$;
 - c) число попадает в одну из трех дюжин $[1, 12]$, $[13, 24]$, $[25, 36]$;
 - d) число попадает в одну из четырех девяток: $[1, 9]$, $[10, 18]$, $[19, 27]$, $[28, 36]$;
 - e) число попадает в одну из шестерок: $[1, 6]$, $[7, 12]$, $[13, 18]$, $[19, 24]$, $[25, 30]$, $[31, 36]$;
 - f) число попадает в одну из троек: $[1, 3]$, $[4, 6]$, ..., $[34, 36]$;
 - g) число попадает в одну из пар: $[1, 2]$, $[3, 4]$, ..., $[35, 36]$;
 - h) выпадет число K от 1 до 36.

- За угадывание комбинации а) или б) игрок получает 1 очко; с) - 2 очка; d) - 3 очка; е) - 5 очков; f) - 11 очков; g) - 17 очков; h) - 35 очков. Генерирование числа осуществляется заданное количество раз и определяется условиями игры. В случае выпадения нуля ни один из участников не получает ни одного очка. Побеждает набравший большее количество очков.
9. Игра "Карусель-лото" (вариант 2). Перед игрой каждый из участников имеет 8 очков. Перед началом партии участник как бы "ставит" очко на выбранную комбинацию. В случае успеха он получает назад свое очко плюс приз в очках (см. условие задачи 8), в случае неудачи - теряет свое очко. При выпадении нуля теряются все поставленные в партии очки. Участник может пропускать партию, ставить несколько очков на одну комбинацию (в этом случае они считаются независимыми). Если у участника кончились очки, он выбывает из игры.
10. Игра "Морской бой" (вариант 1). Игра происходит между двумя участниками, один из которых компьютер, на поле размером 10×10 клеток. Каждый из участников расставляет на поле 10 одноклеточных катеров (катера не могут касаться друг друга). При попадании в катер при очередном ходе он считается уничтоженным. Побеждает уничтоживший первым катера соперника. Если участник во время очередного хода уничтожил катер противника, то он ходит еще раз. Ход в клетку, соседнюю с уничтоженным катером, считается недействительным и повторяется. Компьютер расставляет катера случайным образом. Его партнер вводит координаты катеров с клавиатуры.
11. Игра "Морской бой" (вариант 2). На поле каждый участник расставляет эскадру, состоящую из одного 4-клеточного линкора, двух 3-клеточных крейсеров, трех 2-клеточных эсминцев и четырех 1-клеточных катеров. Все многоклеточные корабли расположены линейно, т.е. только горизонтально или вертикально.
12. Игра "Морской бой" (вариант 3). Корабли могут располагаться произвольно, но не должны касаться друг друга (находиться в соседних клетках).
13. Игра в "кости" (вариант 1). Кость - кубик с размеченными гранями, на грани нанесены 1, 2, ..., 6 точек (очков). Каждый игрок сначала бросает 6 костей сразу. После первого хода игрок выбирает какую-либо из выпавших граней и откладывает кости, которые выпали на эту грань (эти кубики выбывают из игры). Далее игрок сначала бросает 6 костей сразу. После первого хода игрок выбирает какую-либо из выпавших граней и откладывает кости, которые выпали на эту грань (эти кубики выбывают из игры). Далее игрок бросает оставшиеся кости и снова откладывает кости, выпавшие на выбранную после первого броска грань, и т.д., пока не будет

отложено не менее пяти костей. Далее кости бросает второй игрок. Побеждает игрок, отложивший не менее пяти костей за меньшее число бросков. При одинаковом числе бросков победителем считается игрок, выбравший грань с большим числом очков.

14. Игра в "кости" (вариант 2). Закончив выбрасывание костей на первую выбранную грань $M1$ (отложив не менее пяти костей), игрок опять бросает шесть костей, выбирает вторую грань $M2 \neq M1$ и снова повторяет броски, пока не будет отложено не менее пяти костей, выпавших на грань $M2$, и т.д., пока не будут выбраны все шесть граней. Далее кости бросает второй игрок.
15. Игра "Быки и коровы" (вариант 1). Компьютер генерирует случайное четырехзначное число, в котором все цифры различны. Игрок должен отгадать число, делая несколько попыток. После ввода очередного числа компьютер сообщает о степени совпадения введенного числа с исходным, т.е. количество «быков» и «коров»: «корова» - это цифра в числе игрока, совпадающая по разряду с такой же цифрой в загаданном числе; «бык» - это цифра в числе игрока, не совпадающая по разряду с такой же цифрой в загаданном числе. Если, например, загадано число 6482, то число 5428 содержит 1«корову» и 2«быка».
16. Игра "Быки и коровы" (вариант 2). Цифры в числе могут совпадать. Ноль не может быть старшей цифрой числа.
17. Игра «Баше». Игра происходит между двумя участниками, один из которых компьютер. Имеется N предметов. Соперники ходят по очереди. За каждый ход игрок может взять от 1 до K предметов. Проигрывает тот, кто вынужден взять последний предмет.
18. Игра «Жизнь». Игра ведется на поле размером 10×10 клеток. В каждой клетке может располагаться 0 или 1. Соседями данной клетки считаются все клетки, расположенные рядом с ней по вертикали, горизонтали или диагонали. На каждом шаге происходит следующее преобразование: если сумма значений соседей меньше двух или больше трех, то в клетку помещается значение 0; если сумма значений соседей равна трем, то в клетку помещается значение 1; иначе значение не изменяется.

Проследить за развитием исходной конфигурации в нескольких поколениях. Предусмотреть возможность прекращения работы программы по желанию пользователя.

СПИСОК ЛИТЕРАТУРЫ

1. Куренкова Т.В., Светозарова Г.И. Основы алгоритмизации и объектно-ориентированного программирования : учеб. пособие – М. : Изд. Дом МИСиС, 2011. – 197 с.
2. Рихтер Дж. Программирование на платформе Microsoft .NET Framework. – М.: Издательско-торговый дом «Русская редакция»; СПб.: Питер, 2005. – 512 с.
3. Библиотека MSDN (по-русски) <http://msdn.microsoft.com/ru-ru/library/default.aspx>
4. Спецификация языка C#.
5. Фролов А. В., Фролов Г. В. Язык C#. Самоучитель. - М.: ДИАЛОГ-МИФИ, 2003. - 560 с.

Таблицы встроенных типов

Таблица целых типов

Тип	Диапазон	Размер
sbyte	От -128 до 127	8-разрядное целое число со знаком
byte	От 0 до 255	8-разрядное целое число без знака
char	от U+0000 до U+ffff	16-разрядный символ Юникода
short	От -32 768 до 32 767	16-разрядное целое число со знаком
ushort	От 0 до 65 535	16-разрядное целое число без знака
int	От -2 147 483 648 до 2 147 483 647	32-разрядное целое число со знаком
uint	От 0 до 4 294 967 295	32-разрядное целое число без знака
long	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	64-разрядное целое число со знаком
ulong	От 0 до 18 446 744 073 709 551 615	64-разрядное целое число без знака

Таблица типов с плавающей запятой

Тип	Приблизительный диапазон	Точность
float	От $\pm 1,5e-45$ до $\pm 3,4e38$	7 знаков
double	От $\pm 5,0e-324$ до $\pm 1,7e308$	15-16 знаков

Ключевое слово `decimal` обозначает 128-разрядный тип данных. По сравнению с типом данных с плавающей запятой, тип `decimal` имеет более точный и узкий диапазон, благодаря чему он подходит для финансовых расчетов.

Тип	Приблизительный диапазон	Точность
decimal	от $\pm 1,0 \times 10^{-28}$ до $\pm 7,9 \times 10^{28}$	28-29 значимых цифр

Ключевое слово `bool` используется для объявления переменных для хранения логических значений, `true` и `false`.

Тип данных `string` — это последовательность, содержащая любое число знаков Юникода.

Региональные стандарты

Региональные стандарты выбираются как параметры установки пользователем при установке Windows. Платформа .NET Framework предоставляет широкие возможности для разработки международных приложений. При разработке международных приложений рекомендуется разделять процесс на три этапа: глобализация, обеспечение возможности локализации и локализация.

Глобализация является первым этапом разработки международных приложений. На этом шаге пишется исполняемый код приложения. По-настоящему глобальные приложения должны быть нейтральны в отношении языка и региональных параметров.

Перед переходом к локализации необходимо выполнить промежуточную проверку, позволяющую определить локализуемость приложения. Если приложение локализуемо, то исполняемый код приложения корректно отделен от его ресурсов. При правильной оценке локализуемости приложения нет необходимости изменять исходный код приложения во время локализации.

Последний шаг при построении международных приложений — это локализация, которая заключается в настройке приложения под определенный язык и регион.

Локализация — это процесс перевода ресурсов приложения в локализованные версии для каждого языка и региональных параметров, которые поддерживает приложение.

Идентификатор **языкового стандарта** определяет региональные параметры и язык для конкретной географической области. К некоторым категориям, зависящим от языкового стандарта, относится формат дат и отображения денежных значений.

Язык определяет соглашения по форматированию текста и даты, а страна или регион определяют национальные соглашения. Для каждого языка существует уникальное сопоставление, представленное кодовыми страницами, в которых содержатся символы, отличные от символов в алфавите (такие как знаки препинания и цифры). **Кодовая страница** — это набор символов, связанный с языком. А языковой стандарт — это уникальная комбинация языка, страны или региона и кодовой страницы.

Различные языки могут использовать разные кодовые страницы. Например, кодовая страница 1252 используется для английского и большинства европейских языков, а кодовая страница 932 используется для японского иероглифического языка, **кодовая страница 1251 используется для русского языка.**

Подробнее в <http://msdn.microsoft.com/ru-ru/library/8w60z792.aspx>.