МЕТОДЫ

Основные понятия

Метод – это функциональный элемент класса, который реализует вычисления, или другие действия, выполняемые классом, или его экземпляром (объектом). Метод представляет собой законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может многократно. Совокупность методов класса определяет, что конкретно может делать класс. Например, стандартный класс Math содержит методы, которые позволяют вычислять значения математических функций.

Синтаксис объявления метода:

```
[атрибуты] [спецификторы] тип_результата имя_метода
                                ([список_формальных_параметров])
{
    тело_метода;
    return значение;
}
```

где:

1) *атрибуты* и *спецификторы* являются необязательными элементами в описании метода. На данном этапе атрибуты нами использоваться не будут, а из всех спецификаторов мы в обязательном порядке будем использовать спецификатор static, который позволяет обращатся к методу класса без создания его экземпляра**.** Остальные спецификаторы мы рассмотрим позже.
2) *тип_результата* определяет тип значения, возвращаемого методом. Это может быть любой тип, включая типы классов, создаваемые программистом, а также тип void, который говорит о том, что метод ничего не возвращает.
3) *имя_метода* будет использоваться для обращания к нему из других мест программы и должно быть корректно заданным, с учетом требований, накладываемых на идентификаторы в C#.
4) *список_формальных_параметров* представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми. Формальные параметры – это переменные, которые получают значения, передаваемые методу при вызове. Если метод не имеет параметров, то *список_параметров* остается пустым.
5) *return* – это оператор безусловного перехода, который завершает работу метода и возвращает *значение*, стоящие после оператора return, в точку его вызова. Тип *значения* должен соответствовать *типу_результата*, или приводиться к нему. Если метод не должен возвращать никакого значения, то указывается тип void, и в этом случае оператор return либо отсутствует, либо указывается без возвращаемого значения.

Рассмотрим простейший пример метода:

```
class Program
{
   static void Func() //дополнительный метод
   {
      Console.Write("x= ");
      double x = double.Parse(Console.ReadLine());
      double y = x*x;
```

```
        Console.WriteLine("y({0})={1}", x, y );
    }
    static void Main() //точка входа в программу
    {
        Func(); //первый вызов метода Func
        Func(); //второй вызов метода Func
    }
}
```

В даном примере в метод Func не передаются никакие значения, поэтому список параметров пуст. Кроме того, метод ничего не возвращает, поэтому тип возвращаемого значения void, и в теле метода отсутствует оператор return. В основном методе Main мы вызвали метод Func два раза. При необходимости данный метод можно будет вызвать столько раз, сколько потребуется для решения задачи.

Изменим исходный пример так, чтобы в него передавалось значение x, а сам метод возвращал значение y.

```
class Program
{
    static double Func( double x) //дополнительный метод
    {
        return x*x;   //возвращаемое значение
    }
    static void Main() //точка входа в программу
    {
        Console.Write("a=");
        double a = double.Parse(Console.ReadLine());
        Console.Write("b=");
        double b = double.Parse(Console.ReadLine());
        Console.Write("h=");
        double h = double.Parse(Console.ReadLine());
        for (double x = a; x <= b; x += h)
        {
            double y = Func(x);//вызов метода Func
            Console.WriteLine("y({0:f1})={1:f2}", x, y);
        }
    }
}
```

В данном примере метод Func содержит параметр x типа double. Для того, чтобы метод Func возвращал в вызывающий его метод Main значение выражения x*x (типа double), перед именем метода указывается тип возвращаемого значения – double, а в теле метода используется оператор передачи управления – *return*. Оператор *return* завершает выполнение метода и передает управление в точку его вызова.

Задание 1

Преобразуйте программу так, чтобы метод Func возвращал значение выражения:

$x^2+x-1;$

Задание 2

Преобразуйте программу так, чтобы метод Func возвращал значение выражения:

$$\begin{cases} x^2, & при \ \ x \ge 0; \\ 1/x, & при \ \ x < 0. \end{cases}$$

Рассмотрим другой пример:

```
class Program
{
   static int Func( int x, int y)     // 1
   {
      return (x>y)? x:y;
   }
   static void Main()
   {
      Console.Write("a=");
      int a = int.Parse(Console.ReadLine());
      Console.Write("b=");
      int b = int.Parse(Console.ReadLine());
      Console.Write("c=");
      int c = int.Parse(Console.ReadLine());
      int max = Func(Func(a, b), c);  //2 - вызовы метода Func
      Console.WriteLine("max({0}, {1}, {2})={3}", a, b, c, max);
   }
}
```

На этапе описания метода (строка 1) указывается, что метод Func имеет два целочисленных *формальных* параметра – x, y. На этапе вызова (строка 2) в метод передаются *фактические* параметры, которые по количеству и по типу совпадают с формальными параметрами. Если количество фактических и формальных параметров будет различным, то компилятор выдаст соответствующее сообщение об ошибке. Если параметры будут отличаться типами, то компилятор попытается выполнить неявное преобразование типов. Если неявное преобразование невозможно, то также будет сгенерирована ошибка.

Обратите внимание на то, что при вызове метода Func использовалось вложение одного вызова в другой.

Задание

Преобразуйте программу, не изменяя метод Func, чтобы можно было найти наибольшее значение из четырех чисел: a, b, c, d.

Таким образом, параметры используются для обмена информацией между вызывающим и вызываемым методами. В C# предусмотрено четыре типа параметров: параметры-значения, параметры-ссылки, выходные параметры и параметры, позволяющие создавать методы с переменным количеством аргументов.

При передаче параметра *по значению* метод получает копии параметров, и операторы метода работают с этими копиями. Доступа к исходным значениям параметров у метода нет, а следовательно, нет и возможности их изменить. Все примеры, рассмотренные ранее, использовали передачу данных по значению.

Рассмотрим небольшой пример:

```
class Program
{
   static void Func(int x)
   {
      x += 10;// изменили значение параметра
      Console.WriteLine("In Func: " + x);
   }
   static void Main()
   {
      int a = 10;
      Console.WriteLine("In Main: {0}", a);
      Func(a);
      Console.WriteLine("In Main: {0}", a);
   }
}
```

Результат работы программы:

```
In Main: 10
In Func: 20
In Main: 10
```

В данном примере значение формального параметра x было изменено в методе Func, но эти изменения не отразились на фактическом параметре *a* метода Main.

Замечание

Передача параметров по значению *позволяет гарантировать целостность исходных параметров только в случае, если исходные параметры имеют тип, являющийся размерным типом (value-type). Если исходный параметр имеет ссылочный тип (reference type), то в метод копируется ссылка на объект и, следовательно, можно менять значения объектов.*

При передаче параметров *по ссылке* метод получает копии адресов параметров, что позволяет осуществлять доступ к ячейкам памяти по этим адресам и изменять исходные значения параметров. Для того чтобы параметр передавался по ссылке, необходимо при описании метода перед формальным параметром, и при вызове метода перед соответствующим фактическим параметром поставить спецификатор *ref*.

```
class Program
{
   static void Func(int x, ref int y)
   {
      x += 10; y += 10;  //изменение параметров
      Console.WriteLine("In Func: {0}, {1}", x, y);
   }
   static void Main()
   {
      int a = 10, b = 10;  // 1
      Console.WriteLine("In Main: {0}, {1}", a, b);
      Func(a, ref b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
   }
}
```

Результат работы программы:

```
In Main: 10  10
In Func: 20  20
In Main: 10  20
```

В данном примере в методе Func были изменены значения формальных параметров x и y. Эти изменения не отразились на фактическом параметре a, т.к. он передавался по значению, но значение b было изменено, т.к. он передавался по ссылке.

Передача параметра по ссылке требует, чтобы аргумент был инициализирован до вызова метода (см. строку 1).

Задание

Удалите в строке 1 инициализацию переменных a и b, и попробуйте откомпилировать и запустить программу.

Итак, при использовании неинициализированных фактических параметров компилятор выдаст сообщение об ошибке. Однако не всегда имеет смысл инициализировать параметр до вызова метода, например, если метод считывает значение этого параметра с клавиатуры, или из файла. В этом случае параметр следует передавать как выходной, используя спецификатор *out*:

```
class Program
{
   static void Func(int x, out int y)
   {
      // определение значения выходного параметра y
      x += 10; y = x;
      Console.WriteLine("In Func: {0}, {1}", x, y);
   }
   static void Main()
   {
      int a = 10, b;
      Console.WriteLine("In Main: {0}", a);
      Func(a, out b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
   }
}
```

Результат работы программы:

```
In Main: 10
In Func: 20  20
In Main: 10  20
```

В данном примере в методе Func формальный параметр y и соответствующий ему фактический параметр b метода Main были помечены спецификатором out. Поэтому, значение b до вызова метода Func можно было не определять, но изменение параметра повлекло за собой изменение значения параметра b.

Замечание

Параметры, позволяющие создавать методы с переменным количеством аргументов, основаны на использовании массивов, поэтому они будут рассмотрены позже.

Перегрузка методов

Представляется разумным, чтобы методы, реализующие один и тот же алгоритм для различного количества параметров, или различных типов данных, имели одно и то же имя. Использование таких методов называется *перегрузкой методов*. Компилятор определяет, какой именно метод требуется вызвать, по типу и количеству фактических параметров.

Рассмотрим следующий пример:

```
class Program
{
   //первая версия метода Max – возвращает значение
   //наибольшей цифры заданного числа
   static int Max(int a)
   {
      int b = 0;
      while (a > 0)
      {
         if (a % 10 > b)
         {
            b = a % 10;
         }
         a /= 10;
      }
      return b;
   }

   // вторая версия метода max – возвращает значение
   // наибольшего из двух чисел
   static int Max(int a, int b)
   {
      if (a > b)
         return a;
      else
         return b;
   }

   // третья версия метода max – возвращает значение
   // наибольшего из трех чисел
   static int Max(int a, int b, int c)
   {
      if (a > b && a > c)
         return a;
      else if (b > c)
            return b;
         else
            return c;
   }

   static void Main()
   {
      int a = 1283, b = 45, c = 35740;
      Console.WriteLine(Max(a));
```

```
            Console.WriteLine(Max(a, b));
            Console.WriteLine(Max(a, b, c));
        }
    }
```

При вызове метода Max компилятор выбирает вариант, соответствующий типу и количеству передаваемых в метод аргументов. Если точного соответствия не найдено, выполняются неявные преобразования типов в соответствии с общими правилами. Если преобразование невозможно, выдается сообщение об ошибке. Если выбор перегруженного метода возможен более чем одним способом, то выбирается «лучший» из вариантов (вариант, содержащий меньшее количество и длину преобразований в соответствии с правилами преобразования типов). Если существует несколько вариантов, из которых невозможно выбрать подходящий, выдается сообщение об ошибке.

Замечание

В C# существует другой способ создания методов с переменным количеством параметров, он основывается на использовании массивов. Этот способ будет рассмотрен позже.

Если нужно реализовать один и тот же метод для различных типов (например, наш метод Max нужно было бы реализовать для int, double и float), то совсем не обязательно записывать несколько одинаковых методов. В .NET версий 2.0 и выше можно использовать сокращенную запись перегруженных методов, сообщая компилятору только общие сведения о типе, используются обобщенные типы, или Generic'и. Они отчасти напоминают шаблоны языка C++.

Generic'и по сути являются обобщенными типами данных. Т. е., при объявлении такого типа мы используем некоторый формальный (не существующий тип). Когда же мы создаем экземпляр Generic'a, то мы указываем уже конкретный тип. При этом формальный тип для данного экземпляра Generic'a заменяется на этот конкретный тип.

Рассмотрим небольшой пример:

```
using System;
// для работы с Generic мы подключаем специальное пространство имен
using System.Collections.Generic;
using System.Text;

namespace Example
{
   class Program
   {
      // Данный метод меняет значения двух переменных типа <T>.
      // Тип T пока не конкретизирован
      static void Swap<T>(ref T a, ref T b)
      {
         Console.WriteLine("Передаем в Swap() тип {0}",
                                              typeof(T));
         T temp;
         temp = a;
         a = b;
         b = temp;
      }
      static void Main(string[] args)
      {
         int a = 1, b = 2;
```

```
            Console.WriteLine("Перед swap: {0}, {1}", a, b);
            Swap<int>(ref a, ref b); //передаем в Swap целый тип
            Console.WriteLine("После swap: {0}, {1}", a, b);
            Console.WriteLine();
            double x = 3.2, y = -123.27;
            Console.WriteLine("Перед swap: {0} {1}", x, y);
            Swap<string>(ref x, ref y);
            Console.WriteLine("После swap: {0} {1}", x, y);
            Console.ReadLine();
        }
    }
}
```

Перегрузка методов является проявлением *полиморфизма*, одного из основных свойств ООП. Программисту гораздо удобнее помнить одно имя метода и использовать его для работы с различными типами данных, а решение о том, какой вариант метода вызвать, возложить на компилятор. Этот принцип широко используется в классах библиотеки .NET. Например, в стандартном классе Console метод WriteLine перегружен 19 раз для вывода величин разных типов.

Рекурсивные методы

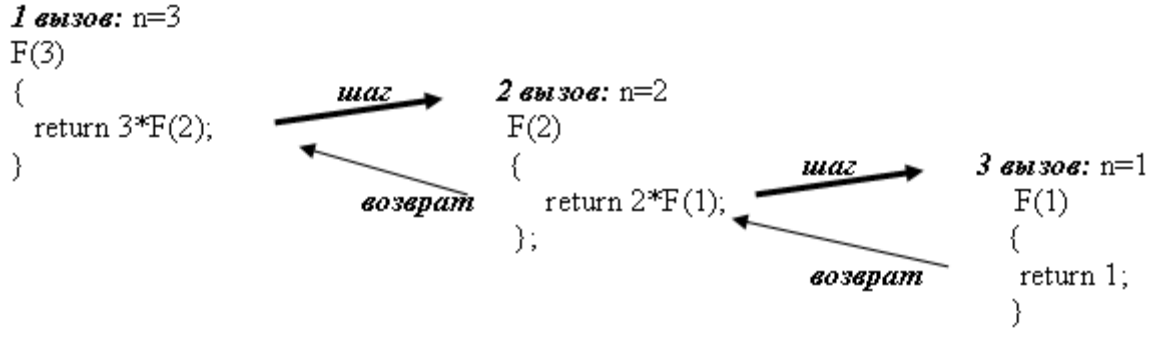
Рекурсивным называют метод, который вызывает сам себя в качестве вспомогательного. В основе рекурсивного метода лежит так называемое «рекурсивное определение» какого-либо понятия. Классическим примером рекурсивного метода является метод, вычисляющий факториал.

Из курса математики известно, что 0!=1!=1, n!=1*2*3…*n. С другой стороны, $n! = (n-1)! * n$. Таким образом, известны два значения параметра n, а именно, n=0 и n=1, при которых мы без каких-либо дополнительных вычислений можем определить значение факториала. Во всех остальных случаях, то есть для n>1, значение факториала может быть вычислено через значение факториала для n-1. Таким образом, рекурсивный метод будет иметь следующий вид:

```
class Program
{
   static ulong F(ulong n) //рекурсивный метод
   {
      if (n==0 || n==1)
      {
         return 1;     //нерекурсивная ветвь
      }
      else
      {
         //шаг рекурсии - повторный вызов метода с другим параметром
         return n * F(n-1);
      }
   }
   static void Main()
   {
      Console.Write("n=");
      ulong n = ulong.Parse( Console.ReadLine());
      ulong f = F(n);   //вызов метода F
```

```
            Console.WriteLine("{0}!={1}", n, f);
        }
    }
```

Рассмотрим работу описанного выше рекурсивного метода для n=3



Первый вызов метода F осуществляется из метода Main. В нашем случае это происходит во время выполнения оператора присваивания f=F(3). Этап вхождения в рекурсию обозначим жирными стрелками. Он продолжается до тех пор, пока значение переменной n не становится равной 1. После этого начинается выход из рекурсии (тонкие стрелки). В результате вычислений получается, что F(3)=3*2*1.

Рассмотренный вид рекурсии называют прямой, простейший вариант которой можно описать следующей структурой:

```
if (<условие>)
   <оператор>;
else
   <вызов данного метода с другими параметрами>;
```

В качестве <условия> обычно записываются некоторые граничные случаи параметров, передаваемых рекурсивному методу, при которых результат его работы заранее известен. Далее следует простой оператор, или блок, а в ветви else происходит рекурсивный вызов данного метода с другими параметрами.

Что необходимо знать для реализации рекурсивного процесса? Со входом в рекурсию осуществляется вызов метода, а для выхода необходимо помнить точку возврата, т.е., то место программы, откуда мы пришли и куда нам нужно будет возвратиться после завершения выполнения метода. Место хранения точек возврата называется стеком вызовов и для него выделяется определенная область оперативной памяти. В стеке запоминаются не только адреса точек возврата, но и копии значений всех параметров, по которым восстанавливается при возврате вызывающий метод. Из-за создания копий параметров при развертывании рекурсии может произойти переполнение стека. Это является основным недостатком рекурсивного метода. Другой недостаток – это время, которое затрачивается на вызовы метода. Вместе с тем рекурсивные методы довольно часто позволяют перейти к более компактной записи алгоритма.

Любой рекурсивный метод можно преобразовать в обычный. И наоборот, любой метод, содержащий циклы, можно преобразовать в рекурсивный, если выявить рекуррентное соотношение между вычисляемыми с помощью метода значениями.

Далее каждую задачу будем решать с использованием обычного и рекурсивного методов.

Пример 1

Найти сумму цифр числа A.

При разработке нерекурсивного метода воспользуемся алгоритмом разложения натурального числа на составляющие его цифры, рассмотренный в разделе «Реализация алгоритмов».

При разработке рекурсивного метода вначале построим рекуррентное соотношение. Если число A=0, то

$$sum(A)=0 \quad \textbf{(1)}$$

Если A принимает ненулевое значение, например 1234, то сумму цифр данного числа можно представить следующим образом:

sum(1234)=sum(123)+4=(sum(12)+3)+4=(((sum(1)+2)+3)+4)=(((sum(0)+1)+2)+3)+4

то есть,

$$sum(A)=sum(A/10)+A\%10 \quad \textbf{(2)}$$

Формулы (1)-(2) определяют рекуррентные соотношения, которые будут использоваться при разработке рекурсивного метода.

```
class Program
{
   static ulong F(ulong a)  //нерекурсивный метод
   {
      ulong sum = 0;
      while (a>0)
      {
         sum += a % 10;
         a /= 10;
      }
      return sum;
   }
   static ulong F_Rec(ulong a)   //рекурсивный метод
   {
      if (a==0)
      {
         return 0;
      }
      else
      {
         return F_Rec(a/10) + a % 10;
      }
   }
   static void Main()
   {
      Console.Write("n=");
      ulong n = ulong.Parse(Console.ReadLine());
      Console.WriteLine("Нерекурсивный вызов: {0}!={1}", n, F(n));
      Console.WriteLine("Рекурсивный вызов: {0}!={1}", n, F_Rec(n));
   }
}
```

Задание

Измените программу так, чтобы методы F и F_Rec вычисляли количество цифр в заданном числе.

Пример 2

Вычислить n-ный член последовательности Фиббоначи.

Напомним, что первые два члена последовательности Фиббоначи равны 1, остальные получаются по рекуррентной формуле $a_n=a_{n-1}+a_{n-2}$.

```
class Program
{
   static ulong F(int n)    //нерекурсивный метод
   {
      if (n==1 || n==2)
      {
         return 1;
      }
      else
      {
         ulong a, a1 = 1, a2 = 1;
         for (byte i = 3; i <= n; i++)
         {
            a = a1 + a2;
            a2 = a1;
            a1 = a;
         }
         return a1;
         }
      }
   static ulong F_Rec(int n)  //рекурсивный метод
   {
      if (n==1 || n==2)
      {
         return 1;
      }
      else
      {
         return F_Rec(n-1) + F_Rec(n-2);
      }
   }
   static void Main()
   {
      Console.Write("n=");
      int n = int.Parse( Console.ReadLine());
      Console.WriteLine("Нерекурсивный вызов: {0}!={1}",n, F(n));
      Console.WriteLine("Рекурсивный вызов: {0}!={1}",n, F_Rec(n));
   }
}
```

Задание

Измените программу так, чтобы методы F и F_Rec вычисляли сумму n членов последовательности Фиббоначи.

Рассмотренные выше рекурсивные методы возвращали некоторое значение, заданное рекуррентным соотношением. Однако не всегда требуется возвращать значение. Кроме того, рассмотренные выше методы представляют собой простой вариант рекурсии. В общем

случае, рекурсивный метод может включать в себя некоторое множество операторов и один, или несколько операторов рекурсивного вызова. Рассмотрим примеры «сложных» рекурсивных методов, не возвращающих значение.

Пример 3

Для заданного значения n вывести на экран n строк, в каждой из которых содержится n звездочек. Например, для n=5 на экран нужно вывести следующую таблицу:

```
*
**
***
****
*****
```

Текст программы, решающей эту задачу, приведен ниже:

```
class Program
{
   //вспомогательный метод вывода на экран строки из n звездочек
   static void Stroka(int n)
   {
      for (int i = 1; i <= n; i++)
      {
         Console.Write("*");
      }
      Console.WriteLine();
   }
   //нерекурсивный метод
   static void F(int n)
   {
      //выводит n строк по i звездочек в каждой
      for (int i = 1 ; i <= n ; i++)
      {
         Stroka(i);
      }
   }
   //рекурсивный метод
   static void F_Rec(int n, int i)
   {
   //если номер текущей строки не больше номера последней строки, то
      if (i<=n)
      {
         //выводим i звездочек в текущей строке и
         Stroka(i);
         //рекурсивно переходим к формированию следующей строки
         F_Rec(n, i+1);
      }
   }
   static void Main()
   {
      Console.Write("n=");
      int n =int.Parse( Console.ReadLine());
      Console.WriteLine("Нерекурсивный вызов:");
      F(n);
```

```
            Console.WriteLine("\nРекурсивный вызов:");
         // определяет номер текущей строки и количество звездочек в ней
            const int i=1;
            F_Rec(n, i);
         }
      }
```

Задание

Измените программу так, чтобы методы F и F_Rec выводили на экран следующую таблицу (n=4):

```
1
22
333
4444
```

Пример 4

Для заданного нечетного значения n (например, для n=7) вывести на экран следующую таблицу:

```
*******
 *****
  ***
   *
   *
  ***
 *****
*******
```

Данную таблицу условно можно разделить на две части. Рассмотрим отдельно верхнюю часть:

Номер строки	**Содержимое экрана**	**i - количество пробелов в строке**	**Количество звездочек в строке**
0	*******	0	7
1	*****	1	5
2	***	2	3
3	*	3	1

Таким образом, если нумеровать строки с нуля, то номер строки совпадает с количеством пробелов, которые нужно напечатать в начале этой строки. Так как n определяет количество звездочек в нулевой строке, а в каждой следующей строке оно уменьшается на 2, то количество звездочек в строке с номером i можно определить по формуле n-2i. Всего нужно напечатать n/2+1 строк.

Аналогичную зависимость можно выявить и для нижней части таблицы.

```
class Program
{
   //метод выводит на экран строку из n заданных символов
   static void Stroka(int n, char a)
   {
```

```
            for (int i = 1; i <= n; i++)
            {
               Console.Write(a);
            }
         }
         static void F(int n)  //нерекурсивный метод
         {
            for (int i = 0; i <= n/2; i++)
            {
               Stroka(i,' ');    //печатаем пробелы
               Stroka(n-2*i,'*');   //затем звездочки
               //затем переводим курсор на новую строку
               Console.WriteLine();
               }
               // аналогично выводим нижнюю часть таблицы
               for (int i = n/2; i >= 0; i--)
               {
                  Stroka(i,' ');
                  Stroka(n-2*i,'*');
                  Console.WriteLine();
               }
            }
            static void F_Rec(int n, int i)  //рекурсивный метод
            {
               if (n>0 )
               {
                  //действия до рекурсивного вызова - позволяют
                  //вывести верхнюю часть таблицы
                  Stroka(i, ' ');
                  Stroka(n, '*');
                  Console.WriteLine();

                  //рекурсивный вызов метода
                  F_Rec(n-2,i+1);

                  //действия после рекурсивного вызова - позволяют
                  //вывести нижнюю часть таблицы
                  Stroka(i, ' ');
                  Stroka(n, '*');
                  Console.WriteLine();
               }
            }
            static void Main()
            {
               Console.Write("n=");
               int n =int.Parse( Console.ReadLine());
               Console.WriteLine("Нерекурсивный вызов:");
               F(n);
               //определяет номер начальной строки и количество
               //пробелов в ней
               const int n=0;
```

```
            Console.WriteLine("\nРекурсивный вызов:");
            F_Rec(n,i);
         }
      }
```

Задание

Измените программу так, чтобы методы F и F_Rec выводили на экран следующую таблицу (n=5):

```
  *
 ***
*****
 ***
  *
```

Все примеры, рассмотренные ранее, относились к прямой рекурсии. Однако существует еще и косвенная рекурсия, в которой метод вызывает себя через другой вспомогательный метод. Продемонстрируем косвенную рекурсию на примере программы, которая для заданного значения n выводит на экран следующее за ним простое число.

Данная программа содержит метод Prim, который возвращает true, если ее параметр является простым числом, и false в противном случае. Чтобы установить, является ли число j простым, нужно проверить делимость числа j на все простые числа, не превышающие квадратный корень из j. Перебор таких простых чисел можно организовать следующим образом:

1) рассмотреть первое простое число – 2;
2) используя метод NextPrim, возвращающий следующее за значением его параметра простое число, получить все простые числа, не превышающие квадрата числа j;
3) используя метод Prim определяем, является ли заданное число простым.

Таким образом, методы Prim и NextPrim перекрестно вызывают друг друга. В этом и проявляется косвенная рекурсия.

```
class Program
{
   static bool Prim (int j)
   {
      int k=2;  //первое простое число
      //значение k «пробегает» последовательность
      //простых чисел, начиная с 2 до корня из j,
      //при этом проверяется, делится ли j на одно из
      //таких простых чисел
      while (k*k<=j && j%k!=0)
      {
         k=NextPrim(k);  //вызов метода NextPrim
      }
      return (j%k==0)? false: true;
   }
   static int NextPrim(int i)
   {
      int p=i+1;
      while (!Prim(p))//вызов метода Prim
      {
         ++p;
      }
```

```
            return p;
        }
        static void Main()
        {
            Console.Write("n=");
            int n = int.Parse(Console.ReadLine());
            Console.WriteLine("Следующее за {0} простое число равно {1}",
                                        n, NextPrim(n));
        }
    }
```

Рекурсия является удобным средством решения многих задач: сортировки числовых массивов, обхода таких структур данных как деревья и графы, и ряда других.

С другой стороны, применение рекурсивных методов в большинстве случаев оказывается нерациональным. Вспомним рекурсивный метод подсчета n-ного члена последовательности Фиббоначи. Данный метод будет работать весьма неэффективно. F_Rec(17) вычисляется в нём как F_Rec(16)+F_Rec(15). В свою очередь, F_Rec(16) вычисляется как F_Rec(15)+F_Rec(14). Таким образом, F_Rec(15) будет вычисляться 2 раза, F_Rec(14) – 3 раза, F_Rec(13) – 5 раз и т.д. Всего для вычисления F_Rec(17) потребуется выполнить более тысячи операций сложения. Для сравнения: при вычислении F(17), т.е., используя нерекурсивный метод подсчета n-ного члена последовательности Фиббоначи, потребуется всего лишь 15 операций сложения.

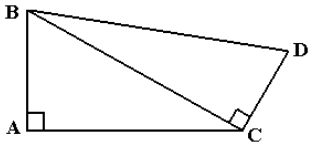
Таким образом, при разработке рекурсивных методов следует задуматься об их эффективности.

Практикум №5

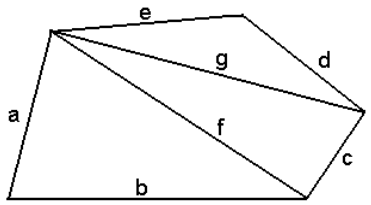
Задание 1

1. Разработать метод min(a,b) для нахождения минимального из двух чисел. Вычислить с его помощью значение выражения z=min(3x,2y)+min(x-y,x+y).
2. Разработать метод min(a,b) для нахождения минимального из двух чисел. Вычислить с его помощью минимальное значение из четырех чисел x, y, z, v.
3. Разработать метод max(a,b) для нахождения максимального из двух чисел. Вычислить с его помощью значение выражения z=max(x,2y-x)+max(5x+3y,y).
4. Разработать метод f(x), который вычисляет значение по следующей формуле: f(x) = x3 − sin x.
   Определить, в какой из точек: a, или b, функция принимает наибольшее значение.
5. Разработать метод f(x), который вычисляет значение по следующей формуле: f(x) = cos(2x) + sin(x − 3).
   Определить, в какой из точек: a, или b, функция принимает наименьшее значение.
6. Разработать метод f(x), который возвращает младшую цифру натурального числа x. Вычислить с его помощью значение выражения z=f(a)+f(b).
7. Разработать метод f(x), который возвращает вторую справа цифру натурального числа x. Вычислить с его помощью значение выражения z=f(a)+f(b)-f(c).
8. Разработать метод f(n), который для заданного натурального числа n находит значение $\sqrt{n}+n$. Вычислить с его помощью значение выражения $\frac{\sqrt{6}+6}{2}+\frac{\sqrt{13}+13}{2}+\frac{\sqrt{21}+21}{2}$.

9. Разработать метод f(n, x), которая для заданного натурального числа n и вещественного x находит значение выражения $\frac{x^n}{n}$. Вычислить с помощью данного метода значение выражения $\frac{x^2}{2}+\frac{x^4}{4}+\frac{x^6}{6}$.
10. Разработать метод f(x), который нечетное число заменяет на 0, а четное число уменьшает в два раза. Продемонстрировать работу данного метода на примере.
11. Разработать метод f(x), который число, кратное 5, уменьшает в 5 раз, а остальные числа увеличивает на 1. Продемонстрировать работу данного метода на примере.
12. Разработать метод f(x), который в двузначном числе меняет цифры местами, а остальные числа оставляет без изменения. Продемонстрировать работу данного метода на примере.
13. Разработать метод f(x), который в трехзначном числе меняет местами первую с последней цифрой, а остальные числа оставляет без изменения. Продемонстрировать работу данного метода на примере.
14. Разработать метод f(a, b), который по катетам a и b вычисляет гипотенузу. С помощью данного метода найти периметр фигуры ABCD по заданным сторонам AB, AC и DC.



15. Разработать метод f(x, y, z), который по длинам сторон треугольника x, y, z вычисляет его площадь. С помощью данного метода по заданным вещественным числам a, b, c, d, e, f, g найти площадь пятиугольника, изображенного на рисунке.



16. Разработать метод f(x_1, y_1, x_2, y_2), который вычисляет длину отрезка по координатам вершин (x_1, y_1) и (x_2, y_2), и метод d(a, b, c), который вычисляет периметр треугольника по длинам сторон a, b, c. С помощью данных методов найти периметр треугольника, заданного координатами своих вершин.
17. Разработать метод f(x_1, y_1, x_2, y_2), который вычисляет длину отрезка по координатам вершин (x_1, y_1) и (x_2, y_2), и метод max(a, b), который вычисляет максимальное из чисел a и b. С помощью данных методов определить, какая из трех точек на плоскости наиболее удалена от начала координат.
18. Разработать метод f(x_1, y_1, x_2, y_2), который вычисляет длину отрезка по координатам вершин (x_1, y_1) и (x_2, y_2), и метод min(a, b), который вычисляет минимальное из чисел a и b. С помощью данных методов найти две из трех заданных точек на плоскости, расстояние между которыми минимально.
19. Разработать метод f(x_1, y_1, x_2, y_2), который вычисляет длину отрезка по координатам вершин (x_1, y_1) и (x_2, y_2), и метод t(a, b, c), который проверяет, существует ли треугольник с длинами сторон a, b и c. С помощью данных методов проверить, можно ли построить треугольник по трем заданным точкам на плоскости.
20. Разработать метод f(x_1, y_1, x_2, y_2), который вычисляет длину отрезка по координатам вершин (x_1, y_1) и (x_2, y_2), и метод t(a, b, c), который проверяет, существует ли треугольник с длинами сторон a, b и c. С помощью данных методов проверить, сколько различных треугольников можно построить по четырем заданным точкам на плоскости.

Задание 2

1. Разработать метод, который для заданного натурального числа N возвращает значение true, если число простое и false, если число составное. С помощью данного метода:
   1) вывести на экран все простые числа на отрезке [a, b];
   2) найти количество всех простых чисел на отрезке [a, b];
   3) найти сумму всех составных чисел на отрезке [a, b];
   4) для заданного числа A вывести на экран ближайшее предшествующее по отношению к нему простое число.
2. Разработать метод, который для заданного натурального числа N возвращает количество его делителей. С помощью данного метода:
   1) для каждого целого числа на отрезке [a, b] вывести на экран количество делителей;
   2) вывести на экран только те целые числа отрезка [a, b], у которых количество делителей равно заданному числу;
   3) вывести на экран только те целые числа отрезка [a, b], у которых количество делителей максимально;
   4) для заданного числа A вывести на экран ближайшее следующее по отношению к нему число, имеющее столько же делителей, сколько и число A.
3. Разработать метод, который для заданного натурального числа N возвращает сумму его делителей. С помощью данного метода:
   1) для каждого целого числа на отрезке [a, b] вывести на экран сумму его делителей;
   2) вывести на экран только те целые числа отрезка [a, b], у которых сумма делителей равна заданному числу;
   3) вывести на экран только те целые числа отрезка [a, b], у которых сумма делителей максимальна;
   4) для заданного числа A вывести на экран ближайшее предшествующее по отношению к нему число, сумма делителей которого равна сумме делителей числа A.
4. Разработать функцию, которая для заданного натурального числа N возвращает сумму его цифр. С помощью данной функции:
   1) для каждого целого числа на отрезке [a, b] вывести на экран сумму его цифр;
   2) вывести на экран только те целые числа отрезка [a, b], у которых сумма цифр числа равна заданному значению;
   3) вывести на экран только те целые числа отрезка [a, b], у которых сумма цифр нечетная;
   4) для заданного числа A вывести на экран ближайшее предшествующее по отношению к нему число, сумма цифр которого равна сумме цифр числа A.
5. Разработать функцию, которая для заданных натуральных чисел возвращает их наибольший общий делитель. С помощью данной функции:
   1) сократить дробь вида a/b;
   2) найти наименьшее общее кратное для двух натуральных чисел;
   3) вычислить значение выражения $\frac{a}{b}+\frac{d}{c}$; результат представить в виде обыкновенной дроби, выполнив сокращение;
   4) найти наибольший общий делитель для n натуральных чисел.

Задание 3

Постройте таблицу значений функции y=f(x) для x∈[a, b] с шагом h.

Замечание

При решении задачи разработайте две версии метода f так, чтобы их сигнатуры соответствовали следующим описаниям:

```
static double f (double x)
static void f (double x, out double y)
```

Продемонстрируйте работу перегруженных методов. Объясните, какой из них лучше.

1) $y=\begin{cases}\dfrac{1}{(0.1+x)^2}, \text{ если } x\ge 0.9;\\ 0.2x+0.1, \text{ если } 0\le x<0.9;\\ x^2+0.2, \text{ если } x<0\end{cases}$

2) $y=\begin{cases}\sin(x), \text{если } |x|<3;\\ \dfrac{\sqrt{x^2+1}}{\sqrt{x^2+5}}, \text{если } 3\le|x|<9;\\ \sqrt{x^2+1}-\sqrt{x^2+5}, \text{ если } |x|\ge 9.\end{cases}$

3) $y=\begin{cases}0, \text{ если } x<a;\\ \dfrac{x-a}{x+a}, \text{ если } x>a;\\ 1, \text{ если } x=a.\end{cases}$

4) $y=\begin{cases}x^3-0.1, \text{ если } |x|\le 0.1;\\ 0.2x-0.1, \text{ если } 0.1<|x|\le 0.2;\\ x^3+0.1, \text{ если } |x|>0.2.\end{cases}$

5) $y=\begin{cases}a+b, \text{ если } x^2-5x<0;\\ a-b, \text{ если } 0\le(x^2-5x)<10;\\ ab, \text{ если } x^2-5x\ge 10.\end{cases}$

6) $y=\begin{cases}x^2, \text{ если } (x^2+2x+1)<2;\\ \dfrac{1}{x^2-1}, \text{если } 2\le(x^2+2x+1)<3;\\ 0, \text{ если } (x^2+2x+1)\ge 3.\end{cases}$

7) $y=\begin{cases}-4, \text{если } x<0;\\ x^2+3x+4, \text{если } 0\le x<1;\\ 2, \text{ если } x\ge 1.\end{cases}$

8) $y=\begin{cases}x^2-1, \text{ если } |x|\le 1;\\ 2x-1, \text{ если } 1<|x|\le 2;\\ x^5-1, \text{ если } |x|>2.\end{cases}$

9) $y=\begin{cases}(x^2-1)^2, \text{ если } x<1;\\ \dfrac{1}{(1+x)^2}, \text{ если } x>1;\\ 0, \text{ если } x=1.\end{cases}$

10) $y=\begin{cases}x^2, \text{ если } (x+2)\le 1;\\ \dfrac{1}{x+2}, \text{если } 1<(x+2)<10;\\ x+2, \text{ если } (x+2)\ge 10;\end{cases}$

11) $y=\begin{cases}x^2+5, \text{ если } x\le 5;\\ 0, \text{если } 5<x<20;\\ 1, \text{ если } x\ge 20.\end{cases}$

12) $y=\begin{cases}0, \text{ если } x<0;\\ x^2+1, \text{если } x\ge 0 \text{ и } x\ne 1;\\ 1, \text{если } x=1.\end{cases}$

13) $y=\begin{cases}1, \text{если } x=1 \text{ или } x=-1;\\ \dfrac{-1}{1-x}, \text{если } x\ge 0 \text{ и } x\ne 1;\\ \dfrac{1}{1+x}, \text{если } x<0 \text{ и } x\ne -1.\end{cases}$

14) $y=\begin{cases}0.2x^2-x-0.1, \text{ если } x<0;\\ \dfrac{x^2}{x-0.1}, \text{если } x>0 \text{ и } x\ne 0.1;\\ 0, \text{ если } x=0.1.\end{cases}$

15) $y=\begin{cases}1, \text{ если } (x\text{-}1)<1;\\ 0, \text{ если } (x\text{-}1)=1;\\ -1, \text{если } (x\text{-}1)>1.\end{cases}$

16) $y=\begin{cases}x, \text{ если } x>0;\\ 0, \text{ если } -1\le x\le 0;\\ x^2, \text{ если } x<-1.\end{cases}$

17) $y=\begin{cases}a+bx, \text{ если } x<93;\\ b-ac, \text{ если } 93\le x\le 120;\\ abx, \text{ если } x>120.\end{cases}$

18) $y=\begin{cases}x^2-0.3, \text{ если } y<3;\\ 0, \text{ если } 3\le x\le 5;\\ x^2+1, \text{ если } x>5.\end{cases}$

19) $y=\begin{cases}\sqrt{5x^2+5}, e\text{если} |x|<2;\\ \dfrac{|x|}{\sqrt{5x^2+5}}, \text{ если } 2\le |x|<10;\\ 0, \text{ если } |x|\ge 10.\end{cases}$

20) $y=\begin{cases}\sin(x), \text{ если } |x|<\dfrac{\pi}{2};\\ \cos(x), \text{ если } \dfrac{\pi}{2}\le |x|\le \pi;\\ 0, \text{ если } |x|>\pi.\end{cases}$

Задание 4

Разработать рекурсивный метод, возвращающий значение:

1) n-го члена последовательности $b_1=-10,\quad b_2=2,\quad b_{n+2}=|b_n|-6b_{n+1}$;
2) n-го члена последовательности $b_1=5,\quad b_{n+1}=\dfrac{b_n}{n^2+n+1}$;
3) количества цифр заданного натурального числа;
4) наибольшего общего делителя методом Евклида;
5) $НОД(a,b)=\begin{cases}a, если\, a=b;\\ НОД(a-b,b), если\, a>b;\\ НОД(a,b-a), если\, b>a.\end{cases}$
6) функции Аккермана для неотрицательных чисел n и m. Функция Аккермана определяется следующим образом:
$$A(n,m)=\begin{cases}m+1, \text{ если } n=0;\\ A(n-1,1), \text{ если } n\ne 0, m=0;\\ A(n-1,A(n,m-1)), \text{ если } n>0, m>0.\end{cases}$$
7) числа сочетаний *C(n, m),* где $0\le m\le n$, используя следующие свойства $C_n^0=C_n^n=1;\ C_n^m=C_{n-1}^m+C_{n-1}^{m-1}$ при *0<m<n*;

8) числа *a*, для которого выполняется неравенство $2^{a-1} \le n \le 2^a$, где n – натуральное число. Для подсчета числа *a* использовать формулу: $a(n) = \begin{cases} 1, n = 1; \\ a(n/2) + 1, n > 1. \end{cases}$

9) x^n (*x*–вещественное, $x \neq 0$, а *n*–целое) по формуле:

$$x^n = \begin{cases} 1 & при\ n = 0, \\ 1/x^{|n|} & при\ n < 0, \\ x \cdot x^{n-1} & при\ n > 0. \end{cases}$$

10) функции $F(N) = \frac{N}{\sqrt{1 + \sqrt{2 + \sqrt{3 + \ldots\sqrt{N}}}}}$;

11) цепной дроби: $\cfrac{x}{1 + \cfrac{x}{2 + \cfrac{x}{3 + \ldots\cfrac{x}{n + x}}}}$;

12) заданного десятичного числа в двоичной системе счисления;

13) заданного двоичного числа в десятичной истеме счисления.

Задание 5

Замечание

Разработанные методы не должны возвращать значений.

1. Даны первый член и разность арифметической прогрессии. Написать рекурсивный метод для нахождения n-го члена и суммы n первых членов прогрессии.
2. Даны первый член и знаменатель геометрической прогрессии. Написать рекурсивный метод для нахождения n-го члена и суммы n первых членов прогрессии.
3. Разработать рекурсивный метод, который по заданному натуральному числу N (N≥1000) выведет на экран все натуральные числа, не превышающие N, в порядке возрастания. Например, для N=8, на экран выводится 1 2 3 4 5 6 7 8.
4. Разработать рекурсивный метод, который по заданному натуральному числу N (N≥1000) выведет на экран все натуральные числа, не превышающие N, в порядке убывания. Например, для N=8, на экран выводится 8 7 6 5 4 3 2 1.
5. Разработать рекурсивный метод для вывода на экран стихотворения:

```
10 лунатиков жили на луне
10 лунатиков ворочались во сне
Один из лунатиков упал с луны во сне
9 лунатиков осталось на луне
9 лунатиков жили на луне
9 лунатиков ворочались во сне
Один из лунатиков упал с луны во сне
8 лунатиков осталось на луне
...
И больше лунатиков не стало на луне
```

6. Дано натуральное число n. Разработать рекурсивный метод для вывода на экран следующей последовательности чисел:

```
1
2  2
3  3  3
…
n  n  n  …  n
```

7. Дано натуральное число n. Разработать рекурсивный метод для вывода на экран следующей последовательности чисел:

```
1
2    1
3    2    1
…
n    n-1  n-2  …  1
```

8. Разработать рекурсивный метод для вывода на экран цифр натурального числа в прямом порядке. Применить этот метод ко всем числам из интервала от A до B.
9. Разработать рекурсивный метод для вывода на экран всех делителей заданного натурального числа n.
10. Дано натуральное четное число n. Разработать рекурсивный метод для вывода на экран следующей картинки:

```
*********   (n звездочек)
********    (n-1 звездочка)
*******     (n-2 звездочки)
…
*           (1 звездочка)
```

11. Дано натуральное четное число n. Разработать рекурсивный метод для вывода на экран следующей картинки:

```
*********   (0 пробелов, n звездочек)
 ********   (1 пробел, n-1 звездочка)
  *******   (2 пробела, n-2 звездочки)
   …
    *       (n-1 пробел, 1 звездочка)
```

12. Дано натуральное четное число n. Разработать рекурсивный метод для вывода на экран следующей картинки:

* *	(n пробелов между звездочками)
** **	(n-2 пробела)
*** ***	(n-4 пробела)
…	…
***** *****	(2 пробела)
**********	(0 пробелов)
***** *****	(2 пробела)
…	…
*** ***	(n-4 пробела)
** **	(n-2 пробела)
* *	(n пробелов

13. Дано натуральное число n. Разработать рекурсивный метод для вывода на экран следующей картинки:

1	(1 раз)
222	(3 раза)
33333	(5 раз)
…	(n раз)
33333	(5 раз)
222	(3 раза)
1	(1 раз)

14. Разработать рекурсивный метод для вывода на экран следующей картинки:

AAAAAAAAAA…AAAAAAAAAA	(78 раз)
BBBBBBBBB…BBBBBBBB	(76 раз)
CCCCCCCC…CCCCCCCC	(74 раз)
…	…
YYY…YYY	(30 раза)
ZZ…ZZ	(28 раз)
YYY…YYY	(30 раза)
…	…
CCCCCCCC…CCCCCCCC	(74 раз)
BBBBBBBBB…BBBBBBBB	(76 раз)
AAAAAAAAAA…AAAAAAAAAA	(78 раз)

Задание 6

1. Разработать рекурсивную функцию для вывода на экран всех возможных разложений натурального числа n на множители (без повторений). Например, для n=12 на экран может быть выведено:

```
2*2*3=12
2*6=12
3*4=12
```

2. Разработать рекурсивную функцию для вывода на экран всех возможных разложений натурального числа n на слагаемые (без повторений). Например, для n=5 на экран может быть выведено:

```
1+1+1+1+1=5
1+1+1+2=5
1+1+3=5
1+4=5
2+1+2=5
2+3=5
```

3. Разработать рекурсивную функцию для вычисления определителя заданной матрицы, пользуясь формулой разложения по первой строке: $\det(A) = \sum_{k=1}^{n} (-1)^{k+1} a_{1k} \det(B_k)$. Продемонстрируйте работу данной функции на примерах.