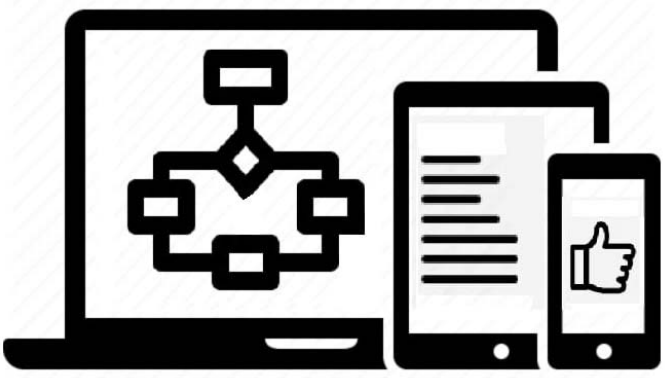
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов специальности 6-05-0612-03 «Системы управления информацией» дневной и заочной форм обучения

Часть 2



Могилев 2024

УДК 004.4
ББК 32.973
О75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «6» декабря 2023 г., протокол № 5

Составитель ст. преподаватель А. И. Кашпар

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации предназначены к выполнению лабораторных работ для студентов специальности 6-05-0612-03 «Системы управления информацией» дневной и заочной форм обучения.

Учебное издание

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Часть 2

Ответственный за выпуск	В. В. Кутузов
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 16 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Дисциплина «Основы алгоритмизации и программирования» формирует у студентов основополагающие знания, умения и навыки в области алгоритмизации вычислительных процессов, принципов разработки, тестирования и анализа программного кода на языке высокого уровня C#, необходимые для успешного изучения дисциплин профессионального цикла и дальнейшей профессиональной деятельности.

С целью закрепления теоретического материала и приобретения студентами практических навыков программирования рабочей программой дисциплины предусмотрено проведение лабораторных работ.

Методические рекомендации содержат цикл лабораторных работ в объеме 34 часов аудиторных занятий, предназначенный для выполнения в весеннем семестре студентами первого курса специальности 6-05-0612-03 «Системы управления информацией» дневной и заочной форм обучения. Для студентов заочной формы обучения в осеннем семестре обязательны к выполнению лабораторные работы № 13, 14 (задания 1, 2), 16 и 18.

Отчет по лабораторным работам оформляется индивидуально каждым студентом на листах формата А4. В состав отчета входят титульный лист, цель работы, текст индивидуального задания, выполнение индивидуального задания (код программы).

1 Лабораторная работа № 13. Статические методы

Цель работы: получение навыков использования статических методов.

Постановка задачи. Решение задач с использованием пользовательских методов.

1.1 Основные сведения

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 метода:

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

где атрибуты и спецификторы являются необязательными элементами синтаксиса описания метода. На данном этапе атрибуты использоваться не будут, а из всех спецификаторов в обязательном порядке будем использовать спецификатор static, который позволит обращаться к методу класса без создания его экземпляра;

тип_возвращаемого_результата определяет тип значения, возвращаемого методом. Это может быть любой тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип void (в этом случае в теле метода отсутствует оператор return);

имя_метода – идентификатор, заданный программистом с учетом требований накладываемыми на идентификаторы в C#, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости;

список_параметров представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми. Параметры – это переменные или константы, которые получают значения, передаваемые методу при вызове. Если метод не имеет параметров, то список_параметров остается пустым;

значение определяет значение, возвращаемое методом. Тип значения должен соответствовать типу_результата или приводится к нему.

Рассмотрим простейший ***пример*** метода.

```
class Program
    {   static void Func()  //дополнительный метод
        {
            Console.Write("x= ");
            double x=double.Parse(Console.ReadLine());
            double y = 1 / 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 данном примере в метод Func не передаются никакие значения, поэтому список параметров пуст. Кроме того, метод ничего не возвращает, поэтому тип возвращаемого значения void. В основном методе Main вызвали метод Func 2 раза. Если будет необходимо, то данный метод можно будет вызвать еще 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 1 / x; //Возвращаемое значение
        }
        static void Main()  //точка входа в программу
        {
          Console.Write("x=");
          double x=double.Parse(Console.ReadLine());
          double y = Func(x); //вызов метода Func
          Console.WriteLine("y({0:f1})={1:f2}", x, y);
        }
```

В данном примере метод Func содержит параметр *x*, тип которого double. Для того чтобы метод Func возвращал в вызывающий его метод Main значение выражения $1/x$ (тип которого 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.

В общем случае параметры используются для обмена информацией между вызывающим и вызываемым методами. В C# для обмена предусмотрено четыре типа параметров: параметры-значения, параметры-ссылки, выходные параметры, параметры-массивы.

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.

Замечание – 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: "+ a);
        Func(a);
        Console.WriteLine("In Main: " + 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.

При использовании параметров по ссылке и выходных параметров 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 служебное слово ref, перед выходным – out.

```
class Program
    {   static void Func(int x, ref int y, out int z)
        {   x += 10; y += 10;  //изменение параметров
            z = 30;  // определение значения выходного параметра z
            Console.WriteLine("In Func: {0}, {1}, {2}", x, y, z);
         }
        static void Main()
        {
            int a=10, b=10, c;  // строка 1
            Console.WriteLine("In Main: {0}, {1}", a, b);
            Func(a, ref b, out c);
            Console.WriteLine("In Main: {0}, {1}, {2}", a, b, c);
        }    }
```

Результат работы программы:

In Main: 10, 10
In Func: 20, 20, 30
In Main: 10, 20, 30

В данном примере в методе Func были изменены значения формальных параметров *x, y* и *z*. Эти изменения не отразились на фактическом параметре, а т. к. он передавался по значению, но значения *b* и *c* были изменены, то они передавались по ссылке.

Передача параметра по ссылке требует, чтобы аргумент был инициализирован до вызова метода (см. строку 1). Если в этой строке не проводить инициализацию переменных, то компилятор выдаст сообщение об ошибке. Выходной параметр *c* до вызова метода Func можно не определять, но изменение параметра *z* отразилось на изменении значения параметра *c*.

Задание 1

1 Разработать метод *f* (*n*), который для заданного натурального числа *n* находит значение $\sqrt{n}+n$. Вычислить с помощью его значение выражения $\frac{\sqrt{6}+6}{2}+\frac{\sqrt{13}+13}{2}+\frac{\sqrt{21}+21}{2}$.

2 Разработать метод *f*(*n*, *x*), который для заданного натурального числа *n* и вещественного *x* находит значение выражения $\frac{x^n}{n}$. Вычислить с помощью данного метода значение выражения $\frac{x^2}{2}+\frac{x^4}{4}+\frac{x^6}{6}$.

3 Разработать метод *f* (*x*), который нечетное число заменяет на 0, а четное число уменьшает в 2 раза. Продемонстрировать работу данного метода на примере.

4 Разработать метод *f* (*x*), который число, кратное 5, уменьшает в 5 раз, а остальные числа увеличивает на 1. Продемонстрировать работу данного метода на примере.

5 Разработать метод *f* (*x*), который в двузначном числе меняет цифры местами, а остальные числа оставляет без изменения. Продемонстрировать работу данного метода на примере.

6 Разработать метод *f* (*x*), который в трехзначном числе меняет местами первую цифру с последней, а остальные числа оставляет без изменения. Продемонстрировать работу данного метода на примере.

7 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 (рисунок 1.1).

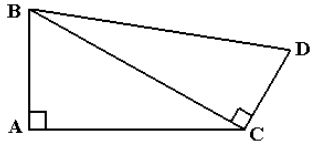


Рисунок 1.1

8 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 1.2.

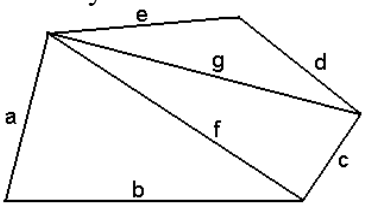


Рисунок 1.2

9 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.

10 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*, *b*. С помощью данных методов определить, какая из

трех точек на плоскости наиболее удалена от начала координат.

11 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, b. С помощью данных методов найти две из трех заданных точек на плоскости, расстояние между которыми минимально.

12 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, c. С помощью данных методов проверить, можно ли построить треугольник по трем заданным точкам на плоскости.

Задание 2

Постройте таблицу значений функции $y = f(x)$ для $x \in [a, b]$ с шагом h. Для решения задачи использовать вспомогательный метод.

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

2 $y = \begin{cases} 0, & \text{если } x < 0; \\ x^2 + 1, & \text{если } x \geq 0 \text{ и } x \neq 1; \\ 1, & \text{если } x = 1. \end{cases}$

3 $y = \begin{cases} a + bx, & \text{если } x < 93; \\ b - ac, & \text{если } 93 \leq x \leq 120; \\ abx, & \text{если } x > 120. \end{cases}$

4 $y = \begin{cases} x^2, & \text{если } (x^2 + 2x + 1) < 2; \\ \dfrac{1}{x^2 - 1}, & \text{если } 2 \leq (x^2 + 2x + 1) < 3; \\ 0, & \text{если } (x^2 + 2x + 1) \geq 3. \end{cases}$

5 $y = \begin{cases} (x^2 - 1)^2, & \text{если } x < 1; \\ \dfrac{1}{(1+x)^2}, & \text{если } x > 1; \\ 0, & \text{если } x = 1. \end{cases}$

6 $y = \begin{cases} x^2 + 5, & \text{если } x \leq 5; \\ 0, & \text{если } 5 < x < 20; \\ 1, & \text{если } x \geq 20. \end{cases}$

7 $y = \begin{cases} x^2 - 1, & \text{если } |x| \leq 1; \\ 2x - 1, & \text{если } 1 < |x| \leq 2; \\ x^5 - 1, & \text{если } |x| > 2. \end{cases}$

8 $y = \begin{cases} x^3 - 0{,}1, & \text{если } |x| \leq 0{,}1; \\ 0{,}2x - 0{,}1, & \text{если } 0{,}1 < |x| \leq 0{,}2; \\ x^3 + 0{,}1, & \text{если } |x| > 0{,}2. \end{cases}$

9 $y = \begin{cases} -4, & \text{если } x < 0; \\ x^2 + 3x + 4, & \text{если } 0 \leq x < 1; \\ 2, & \text{если } x \geq 1. \end{cases}$

10 $y = \begin{cases} \sin(x), & \text{если } |x| < 3; \\ \dfrac{\sqrt{x^2 + 1}}{\sqrt{x^2 + 5}}, & \text{если } 3 \leq |x| < 9; \\ \sqrt{x^2 + 1} - \sqrt{x^2 + 5}, & \text{если } |x| \geq 9. \end{cases}$

11 $y = \begin{cases} x^2, & \text{если } (x+2) \leq 1; \\ \dfrac{1}{x+2}, & \text{если } 1 < (x+2) < 10; \\ x + 2, & \text{если } (x+2) \geq 10. \end{cases}$

12 $y = \begin{cases} 0, & \text{если } x < a; \\ \dfrac{x-a}{x+a}, & \text{если } x > a; \\ 1, & \text{если } x = a. \end{cases}$

2 Лабораторная работа № 14. Массивы как параметры методов

Цель работы: закрепление навыков использования статических методов для обработки массивов.

Постановка задачи. Решение задач с массивами с использованием пользовательских методов. Ввод, вывод, выполнение действий с массивами реализовать в виде методов.

2.1 Основные сведения

Одномерный массив как параметр метода. Так как имя массива фактически является ссылкой, то он передается в метод по ссылке и, следовательно, все изменения элементов массива, являющегося формальным параметром, отразятся на элементах соответствующего массива, являющимся фактическим параметром.

Рассмотрим *пример* передачи массива как параметра (свойство Length позволяет определять количество элементов в массиве):

```
class Program
    {
      static int[] Input() // метод возвращает массив
        {
        Console.Write(("Введите размерность: ");
        int n=int.Parse(Console.ReadLine());
        int []a=new int[n];
        for (int i = 0; i < n; i++)
        {
            Console.Write("a[{0}] = ", i+1);
            a[i]=int.Parse(Console.ReadLine());
        }
        return a;
        }
      static void Print(int[] a) // передаем только ссылку на массив
        {
        for (int i = 0; i < a.Length; i++) Console.Write("{0} ", a[i]);
        Console.WriteLine();
        }
        static void Change(int[] a)
        {
        for (int i = 0; i < a.Length; i++)
            if (a[i] > 0) a[i] = 0;
        }
        static void Main()
        {
        int[] myArray = Input();
        Print(myArray);
        Change(myArray);
        Print(myArray);
}    }
```

Многомерный массив как параметр. При работе с многомерными массивами можно использовать приемы, которые рассмотрены для одномерных массивов.

При обращении к свойству Length для двумерного массива получим общее

количество элементов в массиве. Чтобы получить количество строк, нужно обратиться к методу GetLength с параметром 0. Чтобы получить количество столбцов, – к методу GetLength с параметром 1.

Пример 1 – Подсчитать среднее арифметическое нечетных элементов, расположенных выше главной диагонали.

```
using System;
namespace ConsoleApplication
{  class Class
{
static int [,] Input ()
{
   Console.WriteLine("введите размерность массива");
   Console.Write("n = ");
   n=int.Parse(Console.ReadLine());
   int [,]a=new int[n, n];
   for (int i = 0; i < n; ++i)
   for (int j = 0; j < n; ++j)
   {
      Console.Write("a[{0},{1}]= ", i, j);
      a[i, j]=int.Parse(Console.ReadLine());
   }
   return a;
}
static void Print(int[,] a)
{
   for (int i = 0; i < a.GetLength(0); ++i,Console.WriteLine() )
   for (int j = 0; j < a.GetLength(1); ++j)
      Console.Write("{0,5} ", a[i, j]);
}
static double Rezalt(int[,] a)
{
   int k=0;
   double s=0;
   for (int i = 0; i < a.GetLength(0); ++i)
   for (int j = i+1; j < a.GetLength(1); ++j)
      if (a[i, j] %2!= 0) {++k; s+=a[i, j];}
   if (k!=0) return s/k;
   else return 0;
}
static void Main()
{
   int[,] myArray=Input();
   Console.WriteLine("Исходный массив:");
   Print(myArray);
   double rez=Rezalt(myArray);
   Console.WriteLine("Среднее арифметическое ={0:f2}", rez);
} } }
```

Задания

1 Дана последовательность целых чисел. Задачи из данного пункта решить двумя способами, используя одномерный массив, а затем двумерный. Размерность массива вводится с клавиатуры.

1.1 Заменить все положительные элементы противоположными им числами.

1.2 Заменить все элементы, меньшие заданного числа, этим числом.

1.3 Заменить все элементы, попадающие в интервал [*a, b*], нулем.

1.4 Заменить все отрицательные элементы, не кратные 3, противоположными им числами.

1.5 Все элементы, меньшие заданного числа, увеличить в 2 раза.

1.6 Подсчитать среднее арифметическое элементов.

1.7 Подсчитать среднее арифметическое отрицательных элементов.

1.8 Подсчитать количество нечетных элементов.

1.9 Подсчитать сумму элементов, попадающих в заданный интервал.

1.10 Подсчитать сумму элементов, кратных 9.

1.11 Подсчитать количество элементов, не попадающих в заданный интервал.

1.12 Подсчитать сумму квадратов четных элементов.

1.13 Вывести на экран номера всех элементов, которые не делятся на 7.

2 Дана последовательность из n действительных чисел. Задачи из данного пункта решить, используя одномерный массив.

2.1 Подсчитать количество максимальных элементов.

2.2 Вывести на экран номера всех минимальных элементов.

2.3 Заменить все максимальные элементы нулями.

2.4 Заменить все минимальные элементы на противоположные.

2.5 Поменять местами максимальный элемент и первый.

2.6 Вывести на экран номера всех элементов, не совпадающих с максимальным.

2.7 Найти номер первого минимального элемента.

2.8 Найти номер последнего максимального элемента.

2.9 Подсчитать сумму элементов, расположенных между максимальным и минимальным элементами (минимальный и максимальный элементы в массиве единственные). Если максимальный элемент встречается позже минимального, то выдать сообщение об этом.

2.10 Найти номер первого максимального элемента.

2.11 Найти номер последнего минимального элемента.

2.12 Подсчитать сумму элементов, расположенных между первым максимальным и последним минимальными элементами. Если максимальный элемент встречается позже минимального, то выдать сообщение об этом.

3 Дан массив размером $n \times n$, элементы которого целые числа. При решении задач из данного пункта использовать двумерный массив.

3.1 Подсчитать среднее арифметическое четных элементов, расположенных ниже главной диагонали.

3.2 Подсчитать сумму элементов, расположенных на побочной диагонали.

3.3 Подсчитать среднее арифметическое ненулевых элементов, расположенных над побочной диагональю.

3.4 Подсчитать среднее арифметическое элементов, расположенных под побочной диагональю.

3.5 Поменять местами столбцы по правилу: первый с последним, второй с предпоследним и т. д.

3.6 Поменять местами две средних строки, если количество строк четное, и первую со средней строкой, если количество строк нечетное.

3.7 Поменять местами два средних столбца, если количество столбцов четное, и первый со средним столбцом, если количество столбцов нечетное.

3.8 Если количество строк в массиве четное, то поменять строки местами по правилу: первую строку со второй, третью – с четвертой и т. д. Если количество строк в массиве нечетное, то оставить массив без изменений.

3.9 Если количество столбцов в массиве четное, то поменять столбцы местами по правилу: первый столбец со вторым, третий – с четвертым и т. д.

3.10 Если количество столбцов в массиве нечетное, то оставить массив без изменений.

3.11 Вычислить A^n, где n – натуральное число.

3.12 Подсчитать норму матрицы по формуле $\|A\| = \sum_i \max_j a_{i,j}$.

3.13 Подсчитать норму матрицы по формуле $\|A\| = \sum_j \max_i a_{i,j}$.

4 Дан массив размером $n \times n$, элементы которого целые числа. Для хранения массив $n \times n$ использовать ступенчатый массив.

Пример 2 – Найти максимальный элемент в каждой строке и записать данные в новый массив.

```
using System;
namespace ConsoleApplication
{  class Class
{
static int [][] Input ()
{
   Console.WriteLine("введите размерность массива");
   Console.Write("n = ");
   int n=int.Parse(Console.ReadLine());
   int [][]a=new int[n][];
   for (int i = 0; i < n; ++i)
   {
   a[i]=new int [n];
   for (int j = 0; j < n; ++j)
   {
      Console.Write("a[{0},{1}]= ", i, j);
      a[i][j]=int.Parse(Console.ReadLine());
   }
   }
   return a;
}
static void Print(int[] a)
{
   for (int i = 0; i < a.Length; ++i)
   Console.Write("{0,5} ", a[i]);
}
static void Print(int[][] a)
{
   for (int i = 0; i < a.Length; ++i,Console.WriteLine() )
   for (int j = 0; j < a[i].Length; ++j)
      Console.Write("{0,5} ", a[i][j]);
}
static int Max(int[] a)
{
   int max=a[0];
```

```
   for (int i = 1; i < a.Length; ++i)

   if (a[i] >max) {max=a[i];}
   return max;
}
static void Main()
{
   int[][] myArray=Input();
   Console.WriteLine("Исходный массив:");
   Print(myArray);
   int[]rez=new int [myArray.Length];
   for (int i=0;i<myArray.Length; ++i)
   rez[i]=Max(myArray[i]);
   Console.WriteLine("Новый массив:");
   Print(rez);
} } }
```

4.1 Найти минимальный элемент в каждом столбце и записать данные в новый массив.

4.2 Четные столбцы таблицы заменить на вектор *X*.

4.3 Нечетные строки таблицы заменить на вектор *X*.

4.4 Вычислить *A***X*, где *A* – двумерная матрица, *X* – вектор.

4.5 Для каждой строки подсчитать количество положительных элементов и записать данные в новый массив.

4.6 Для каждого столбца подсчитать сумму отрицательных элементов и записать данные в новый массив.

4.7 Для каждого столбца подсчитать сумму четных положительных элементов и записать данные в новый массив.

4.8 Для каждой строки подсчитать количество элементов, больших заданного числа, и записать данные в новый массив.

4.9 Для каждого столбца найти первый положительный элемент и записать данные в новый массив.

4.10 Для каждой строки найти последний четный элемент и записать данные в новый массив.

4.11 Для каждого столбца найти номер последнего нечетного элемента и записать данные в новый массив.

4.12 Для каждой строки найти номер первого отрицательного элемента и записать данные в новый массив.

3 Лабораторная работа № 15. Рекурсивные методы

Цель работы:

– закрепление навыков использования статических методов;

– изучение механизма рекурсии.

Постановка задачи. Решение задач реализовать двумя способами: без использования рекурсии и с использованием рекурсии.

3.1 Основные сведения

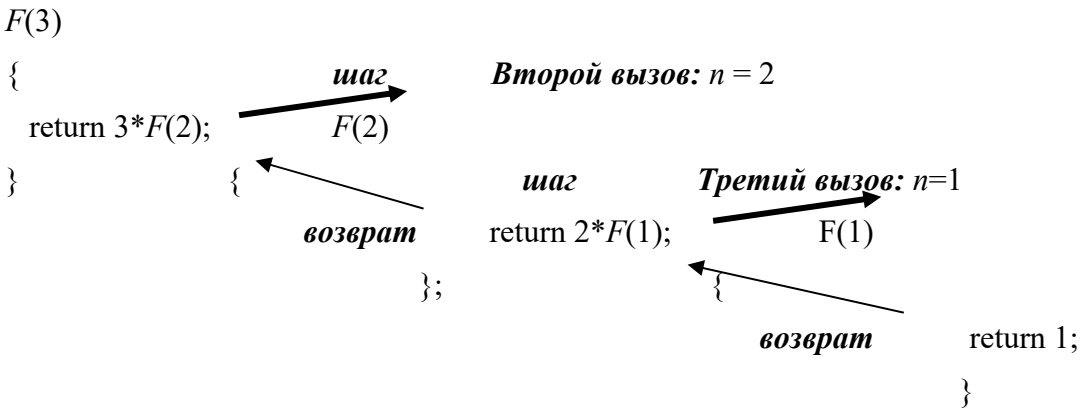
Рекурсивным называют метод, если он вызывает сам себя в качестве вспомогательного. В основе рекурсивного метода лежит так называемое «рекурсивное определение» какого-либо понятия. Классическим примером рекурсивного метода является метод, вычисляющий факториал.

Из курса математики известно, что 0! = 1! = 1, $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$. С другой стороны, $n! = (n-1)! \cdot n$. Таким образом, известны два частных случая параметра n, а именно $n = 0$ и $n = 1$, при которых без каких-либо дополнительных вычислений можно определить значение факториала. Во всех остальных случаях, т. е. для $n > 1$, значение факториала может быть вычислено через значение факториала для параметра $n - 1$. Таким образом, рекурсивный метод будет иметь вид:

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

Рассмотрим работу описанного выше рекурсивного метода для $n = 3$.

Первый вызов: $n = 3$



Первый вызов метода осуществляется из метода Main, в нашем случае командой $f = F(3)$. 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. Метод с прямой рекурсией обычно содержит следующую структуру:

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, поэтому 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 метода. Место хранения точек возврата называется стеком вызовов и для него выделяется определенная область оперативной памяти. В этом стеке запоминаются не только адреса точек возврата, но и копии значений всех параметров. По этим копиям восстанавливается при возврате вызывающий метод. При развертывании рекурсии за счет создания копий параметров возможно переполнение стека. Это является основным недостатком рекурсивного метода. С другой стороны, рекурсивные методы позволяют перейти к более компактной записи алгоритма.

Следует понимать, что любой рекурсивный метод можно преобразовать в обычный метод. И практически любой метод можно преобразовать в рекурсивный, если выявить рекуррентное соотношение между вычисляемыми в методе значениями.

Задания

1 Разработать рекурсивный метод (возвращающий значение).

Сравните результаты с нерекурсивным методом.

Пример 1 – Вычислить *n*-й член последовательности Фиббоначи.

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

```
class Program  {
static int Fb(int n) //нерекурсивный алгоритм
{  int a, a1=1, a2=1;
   if (n==1||n==2) return 1;
   else
   {
   for (int i=2; i<=n; ++i)
   {
      a=a1+a2;
      a1=a2;
      a2=a;
   }
   return a1;
   }
}
```

```
static int FbR(int n) //рекурсивный алгоритм
{  if (n==1 || n==2 )return 1;
   else return FbR(n-1)+FbR(n-2);
}

static void Main()
{
   Console.Write("n=");
   int n=int.Parse(Console.ReadLine());
   Console.WriteLine("Нерекурсивный метод: "+Fb(n));
   Console.WriteLine("Рекурсивный метод: "+FbR(n));
}    }
```

1.1 Для вычисления n-го члена следующей последовательности $b_1 = -10, \quad b_2 = 2, \quad b_{n+2} = |b_n| - 6b_{n+1}$.

1.2 Для вычисления n-го члена следующей последовательности $b_1 = 5, \quad b_{n+1} = \dfrac{b_n}{n^2 + n + 1}$.

1.3 Для нахождения наибольшего общего делителя методом Евклида:

$$НОД(a,b) = \begin{cases} a, \text{если } a = b; \\ НОД(a-b,b), \text{если } a > b; \\ НОД(a,b-a), \text{если } b > a. \end{cases}$$

1.4 Для вычисления значения функции Аккермана для неотрицательных чисел n и m. Функция Аккермана определяется следующим образом:

$$A(n,m) = \begin{cases} m+1, \text{ если } n = 0; \\ A(n-1,1), \text{ если } n \neq 0, m = 0; \\ A(n-1, A(n, m-1)), \text{ если } n > 0, m > 0. \end{cases}$$

1.5 Для вычисления 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 $0 < m < n$.

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

1.7 Для вычисления x^n (x – вещественное, $x \neq 0$, а n – целое) по формуле

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

Вычислить значение для различных x и n.

1.8 Для вычисления $\sum_{i=1}^{n} i$, где n – натуральное число. Для заданных натуральных чисел m и k вычислить с помощью разработанного метода значение выражения $\sum_{i=1}^{m} i + \sum_{i=1}^{2k} i$.

1.9 Для вычисления значения функции $F(N)=\frac{N}{\sqrt{1+\sqrt{2+\sqrt{3+\ldots\sqrt{N}}}}}$.

Найти ее значение при заданном натуральном *N*.

1.10 Для вычисления цепной дроби: $\cfrac{x}{1+\cfrac{x}{2+\cfrac{x}{3+\ldots\cfrac{x}{n+x}}}}$.

Найти значение данной дроби при заданном натуральном *n*.

2 Разработка рекурсивных методов (не возвращающих значений):

Пример 2 – Для заданного значения *n* (например, для *n* = 7) вывести на экран следующую таблицу:

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

Данную таблицу условно можно разделить на две части. Рассмотрим в таблице 3.1 отдельно верхнюю часть.

Таблица 3.1 – Ход построения

Номер строки	Содержимое экрана	Количество пробелов в строке *i*	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, которых нужно напечатать в начале этой строки. При этом количество звездочек в строке, можно определить по формуле $n - 2i$, где *n* – это количество звездочек в нулевой строке. Так как количество звездочек в каждой строке уменьшается на 2, то всего нужно напечатать $n/2 + 1$ строк.

Аналогичную зависимость можно выявить и для нижней части таблицы.

```
class Program
{
static void Stroka(int n, char a) //выводит на экран n раз символ a
{
   for (int i=1; i<=n; ++i)
      Console.Write(a);
}
static void Star(int n) //нерекурсивный метод
{
   for (int i=0; i<=n/2;++i) //выводим верхнюю часть таблицы, в которой в каждой строке
вначале
   {
```

```
     Stroka(i,' '); //печатаем пробелы
     Stroka(n-2*i,'*'); //затем звездочки
     Console.WriteLine(); //затем переводим курсор на новую строку
     }
     for (int i=n/2; i>=0;--i) // аналогично выводим нижнюю часть таблицы
     {
     Stroka(i,' ');
     Stroka(n-2*i,'*');
     Console.WriteLine();
   } }
//рекурсивный метод, где i определяет номер текущей строки, n – количество звездочек в
строке
   static void StarR(int i, int n)
   {
     if (n>0 )
     {
//действия до рекурсивного вызова – позволят вывести верхнюю часть таблицы
     Stroka(i, ' ');
     Stroka(n, '*');
     Console.WriteLine();
//вызываем этот же метод, увеличивая номер строки, и уменьшая количество звездочек в ней
     StarR(i+1,n-2);
//действия после рекурсивного вызова – позволят 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 метод: ");
     Star(n);
     Console.WriteLine("Рекурсивный метод: ");
     StarR(0,n);
}    } }
```

2.1 Даны первый член и разность арифметической прогрессии. Написать рекурсивный метод для нахождения n-го члена и суммы n первых членов прогрессии.

2.2 Даны первый член и знаменатель геометрической прогрессии. Написать рекурсивный метод для нахождения n-го члена и суммы n первых членов прогрессии.

2.3 Разработать рекурсивный метод, который по заданному натуральному числу N ($N \geq 1000$) выведет на экран все натуральные числа не больше N в порядке убывания. Например, для $N = 8$, на экран выводится 8 7 6 5 4 3 2 1.

2.4 Разработать рекурсивный метод для вывода на экран стихотворения:

10 лунатиков жили на Луне.
10 лунатиков ворочались во сне.
И вдруг один лунатик упал с Луны во сне.
9 лунатиков осталось на Луне.
9 лунатиков жили на Луне.
9 лунатиков ворочались во сне.
И вдруг один лунатик упал с Луны во сне.

8 лунатиков осталось на Луне.

...

И больше лунатиков не стало на Луне.

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

1
2 2
3 3 3
…
n *n* *n* … *n*

2.6 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

2.7 Разработать рекурсивный метод для вывода на экран цифр натурального числа в прямом порядке. Применить эту процедуру ко всем числам из интервала от *A* до *B*.

2.8 Разработать рекурсивный метод для перевода числа из десятичной системы счисления в двоичную.

2.9 Разработать рекурсивный метод для перевода числа из двоичной системы счисления в десятичную.

2.10 Разработать рекурсивный метод для вывода на экран всех делителей заданного натурального числа *n*.

2.11 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)

2.12 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

4 Лабораторная работа № 16. Файлы и потоки

Цель работы:

- изучение принципов работы с файлами;
- изучение классов, связанных с файловыми операциями.

Постановка задачи. Решение задач для работы с бинарными и текстовыми файлами.

4.1 Основные сведения

Создание потоков, связанных с файлами. Если нужно создать входной или выходной поток, связанный с локальным файлом, содержащим двоичные данные, следует воспользоваться классами BinaryWriter и BinaryReader из библиотеки Microsoft .NET Framework. Что же касается чтения из файла или записи в файл текстовых данных, то для решения этой задачи обычно используются классы StreamReader и StreamWriter.

В любом случае, перед тем как читать данные из файла или записывать их в файл, необходимо выполнить операцию *открытия* потока, связанного с файлом. Когда работа с файлом окончена, соответствующий поток необходимо *закрыть* явным образом.

Открытие потока FileStream. Для работы с двоичными файлами программа должна вначале создать поток класса FileStream, воспользовавшись соответствующим конструктором, например:

```
FileStream fs = new FileStream("mayfile.dat", FileMode.CreateNew);
```

В качестве первого параметра конструктору необходимо передать полный путь к файлу или имя файла, а в качестве второго – режим открытия потока (таблица 4.1).

Таблица 4.1 – Режимы открытия файла FileMode

Режим	Описание
Append	Если файл существует, он открывается. Текущая позиция устанавливается на конец файла. Если указанного файла нет, то он создается. Использовать только совместно с FileAccess .Write
Create	ОС должна создать новый файл. Если указанный файл уже существует, он будет перезаписан
CreateNew	ОС должна создать новый файл. Если указанный файл уже существует, возникнет исключение IOException
Open	Требуется открыть существующий файл. Необходим доступ FileAccess.Read. Если требуемый файл не найден, возникает исключение System.FileNotFoundException
OpenOrCreate	Если указанный файл существует, он должен быть открыт. В противном случае, ОС должна создать и открыть указанный файл
Truncate	Требуется открыть существующий файл. После открытия файл обрезается до нулевой длины, при этом все ранее хранившиеся в нем данные пропадают. При попытке чтения из файла, открытого подобным образом, возникает исключение

В зависимости от того, для какой цели создается файл, можно выбрать тот или иной режим открытия потока. Например, если поток, связанный с файлом, открывается только для чтения, выбирайте режим FileMode.Open. В этом случае будет открыт существующий файл. Если же файл открывается для записи, то можно либо открыть существующий файл, перезаписав его содержимое или дописав в него новые данные, а также создать новый файл.

В классе FileStream существует несколько конструкторов, позволяющих определить не только путь к файлу, но и режим его открытия, разрешенный доступ к файлу и режим совместного использования файла:

```
FileStream fs = new FileStream(name, FileMode.Open, FileAccess.Read);
FileStream fsl = new FileStream(name, FileMode.Open, FileAccess.Read, FileShare.Read);
```

Остановимся на режиме доступа к файлу, передаваемом приведенным выше конструкторам через третий параметр. Режим доступа задается как статическая константа класса FileAccess.

Таблица 4.2 – Режимы доступа FileAccess

Режим	Тип доступа
Read	Доступ только на чтение
ReadWrite	Доступ на чтение и запись
Write	Доступ на запись

При необходимости программа может задать режим совместного использования файла, когда для одного и того же файла одновременно создается несколько потоков. Этот режим задается при помощи статических констант класса FileShare (таблица 4.3).

Таблица 4.3 – Режимы совместного использования файла FileShare

Режим	Тип доступа
Inheritable	Идентификатор файла может наследоваться дочерним процессом
None	Запрет совместного использования файла. Любые дополнительные запросы на открытие файла будут запрещены то тех пор, пока файл не будет закрыт
Read	Допускаются дополнительные запросы на открытие файла для чтения
ReadWrite	Допускаются дополнительные запросы на чтение и запись
Write	Допускаются дополнительные запросы на запись

Открытие потоков BinaryWriter и BinaryReader. На базе полученного потока FileStream необходимо создать потоки классов BinaryWriter и BinaryReader, предназначенные соответственно для записи в файл и чтения из файла двоичных данных;

```
BinaryWriter bw = new BinaryWriter(fs);
BinaryReader br = new BinaryReader(fs);
```

Закрытие потоков. После того как программа завершила работу с потоками, она должна их закрыть. Для закрытия потоков используется метод Close. Закрывайте потоки в порядке, обратном открытию:

```
FileStream fs   = new FileStream("myfile.datн, FileMode.CreateNew);
BinaryWriter bw = new BinaryWriter(fs);
BinaryReader br = new BinaryReader(fs);
// Работа с потоками
bw.Close();
br.Close();
fs.Close();
```

Запись двоичных данных. Для записи двоичных данных в поток в C# используется класс BinaryWriter, который содержит несколько перегруженных методов Write. Выбор конкретного метода зависит от типа данных, который вы хотите записать.

Чтобы записать данные определенного типа в файл, вы передаете ссылку на экземпляр данных в соответствующий метод Write. Компилятор выберет подходящую перегруженную версию метода в зависимости от типа данных.

При записи текстовой строки (string) в двоичный поток, она будет предварена префиксом, который содержит длину строки. Строка будет записана в кодировке ASCII.

Чтение двоичных данных. Для чтения данных из потока с использованием класса BinaryReader в C#, используются методы, такие как ReadInt32 для чтения 4-байтового целого числа, ReadDouble для чтения 8-байтового числа с плавающей точкой, ReadString, который позволяет считывать текстовые строки с префиксом, содержащим длину строки и т. д. Каждый метод предназначен для чтения определенного типа данных из потока.

Работа с текстовыми файлами. Хотя рассмотренные в предыдущем разделе потоки FileStream, BinaryWriter и BinaryReader можно использовать для записи в файлы и чтения из файлов текстовых строк, лучше прменять специально предназначенные для этого средства. Речь идет о потоках классов StreamWriter и StreamReader. Эти потоки чрезвычайно просты в использовании и удобны для работы с текстовыми файлами.

Основные приемы использования потоков StreamWriter и StreamReader демонстрируются в программе, исходный текст которой приведен ниже:

```
using System;
using System.IO;
namespace TextFile
{
    class TextFileApp
    {
   private const string testFile = "mydata.txt";
   static void Main(string[] args)
   {
       if (File.Exists(testFile)) //1
       {
     Console.WriteLine("Файл {0} уже существует", testFile);  //2
     Console.ReadLine();
     return;
      }
      StreamWriter sw = File.CreateText(testFile);    //3
```

```
      sw.WriteLine("Каждый охотник желает знать, где сидит фазан!");//4
      sw.WriteLine("Число \"Пи\" равно примерно {0}.", 3.1415926);//4
      sw.Close();    //5
       StreamReader sr = File.OpenText(testFile); //6
       while (true)        //7
       {String str = sr.ReadLine();
     if (str == null)
         break;
     Console.WriteLine(str);
       }
       sr.Close();   //8
       Console.WriteLine("Файл  успешно  создан"); Console.ReadLine();
}    } }
```

Важные шаги программы, отмеченные комментариями:

1) проверка существования файла и завершение программы, если файл уже существует;

2) создание и открытие потока StreamWriter для записи в файл;

3) запись строк в поток StreamWriter с использованием методов Write и WriteLine;

4) закрытие потока StreamWriter;

5) открытие потока StreamReader для чтения из файла;

6) чтение строк из потока StreamReader с использованием метода ReadLine в цикле;

7) отображение прочитанных строк на консоли;

8) закрытие потока StreamReader.

Задания

1 Работа с двоичными файлами.

Пример 1 – Создать файл и записать в него вещественные числа из диапазона от *a* до *b* с шагом *h*. Вывести на экран все компоненты файла с нечетными порядковыми номерами.

```
using System;
using System.Text;
using System.IO;
namespace MyProgram
{  class Program
   {
   static void Main()
   {  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());
   //Записываем в файл t.dat вещественные числа из заданного диапазона
      FileStream f=new FileStream("t.dat",FileMode.Open);
      BinaryWriter fOut=new BinaryWriter(f);
      for (double i=a; i<=b; i+=h)   fOut.Write(i);
      fOut.Close();
      //Объекты f и fIn связаны с одним и тем же файлом
      f=new FileStream("t.dat",FileMode.Open);
      BinaryReader fIn=new BinaryReader(f);
      long m=f.Length; //определяем количество байт в потоке
//Читаем данные из файла t.dat, начиная с элемента с номером 1, т.е с 8 байта,
//перемещая внутренний указатель на 16 байт, т.е. на два вещественных числа
```

```
      for (long i=8; i<m; i+=16)
      {  f.Seek(i,SeekOrigin.Begin);
         a=fIn.ReadDouble();
         Console.Write("{0:f2} ",a);    }
      fIn.Close();
      f.Close();
} } }
```

1.1 Создать файл и записать в него степени числа 3. Вывести на экран все компоненты файла с четным порядковым номером.

1.2 Создать файл и записать в него обратные натуральные числа $1, \frac{1}{2}, \ldots, \frac{1}{n}$. Вывести на экран все компоненты файла с порядковым номером, кратным 3.

1.3 Создать файл и записать в него *n* первых членов последовательности Фибоначчи. Вывести на экран все компоненты файла с порядковым номером, не кратным 3.

1.4 Дана последовательность из *n* целых чисел. Создать файл и записать в него все четные числа последовательности. Вывести содержимое файла на экран.

1.5 Дана последовательность из n целых чисел. Создать файл и записать в него все отрицательные числа последовательности. Вывести содержимое файла на экран.

1.6 Дана последовательность из *n* целых чисел. Создать файл и записать в него числа последовательности, попадающие в заданный интервал. Вывести содержимое файла на экран.

1.7 Дана последовательность из *n* целых чисел. Создать файл и записать в него числа последовательности, не кратные заданному числу. Вывести содержимое файла на экран.

1.8 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты, не попадающие в данный диапазон.

1.9 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты файла с нечетными номерами, большие заданного числа.

1.10 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все компоненты файла с четными номерами, меньшие заданного числа.

1.11 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Вывести на экран все положительные компоненты файла.

1.12 Дана последовательность из *n* вещественных чисел. Записать все эти числа в файл. Подсчитать среднее арифметическое компонентов файла, стоящих на четных позициях.

2 Работа с текстовым (символьным) файлом.

Пример 2 – Дан текстовый файл. Найти количество строк, которые начинаются с данной буквы.

```
using System;
using System.Text;
using System.IO;
```

```
namespace MyProgram
{  class Program
   {
   static void Main()
   {  Console.Write("Введите заданную букву: ");
   char a=char.Parse(Console.ReadLine());
   StreamReader fileIn = new StreamReader("text.txt");
   string text;
       int k=0;
       while((text=fileIn.ReadLine())!=null) //считываем из файла строку
   {  //разбиваем текст на слова
       string []newText=text.Split("[ ,.:;]+".ToCharArray());
      //подсчитываем количество слов, начинающихся на заданную букву
      foreach( string b in newText)
        if (b[0]==a) k++;
       }
      Console.WriteLine("k= "+k);
      fileIn.Close();
   } } }
```

2.1 Дан текстовый файл. Найти количество строк, которые начинаются и заканчиваются одной буквой.

2.2 Дан текстовый файл. Найти самую длинную строку и ее длину.

2.3 Дан текстовый файл. Найти самую короткую строку и ее длину.

2.4 Дан текстовый файл. Найти номер самой длинной строки.

2.5 Дан текстовый файл. Найти номер самой короткой строки.

2.6 Дан текстовый файл. Выяснить, имеется ли в нем строка, которая начинается с данной буквы. Если да, то напечатать ее.

2.7 Дан текстовый файл. Напечатать первый символ каждой строки.

2.8 Дан текстовый файл. Напечатать символы с $k1$ по $k2$ в каждой строке.

2.9 Дан текстовый файл. Напечатать все нечетные строки.

2.10 Дан текстовый файл. Напечатать все строки, в которых имеется хотя бы один пробел.

2.11 Дан текстовый файл. Напечатать все строки, длина которых равна данному числу.

2.12 Дан текстовый файл. Напечатать все строки, длина которых меньше заданного числа.

5 Лабораторная работа № 17. Структуры

Цель работы: получение навыков использования структур.

Постановка задачи. Решение задач с применением структур.

5.1 Основные сведения

Структура – это составной объект, в который входят элементы любых типов. В отличие от массива, все элементы которого однотипны, структура может содержать элементы разных типов.

Структуры объявляются с помощью ключевого слова *struct*. Объявление структуры (структурного типа) имеет следующий формат:

```
[спецификаторы] struct имя_структуры
{
[спецификатор] тип_1 элемент_1;
[спецификатор] тип_2 элемент_2;
…
[спецификаторы] тип_n элемент_n;
//методы
} [ список_описателей ];
тело структуры
```

где *struct* – служебное слово;

имя_структуры – конкретное имя структуры; спецификаторы – спецификаторы доступа, определяют статус доступа структурного типа.

Для использования доступа к полям снаружи структуры, члены структуры необходимо описывать с модификатором доступа *public*.

Элементы структуры называются *полями структуры* и могут иметь любой тип. Описание структуры определяет новый тип, имя которого можно использовать в дальнейшем наряду со стандартными типами, например:

```
// описание нового типа с именем Student
  struct Student
   {
       public string fio;
       public int num_zac;
       public double sr_bal;
   };
       // определение переменной типа Student и массива типа Student
       Student stud1;
       Student[] gr = new Student[30];
```

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Структуру можно передавать в функцию и возвращать в качестве значения функции.

Доступ к полям структуры выполняется с помощью операций выбора «.» (точка), например:

```
stud1.fio="Страусенко";
gr[8].sr_bal = 5;
```

Пример – Описать структуру с именем USER, содержащую следующие поля:

NAME – фамилия, имя;

AGE – возраст;

HEIGHT – рост.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из *n* элементов типа USER; записи должны быть упорядочены по росту;

– вывод на экран информации о людях, заданного возраста, если такого нет, выдать на дисплей соответствующее сообщение.

Текст программы:

```
using System;
namespace ConsoleApp6
{ class Program
  { //описание нового типа структуры
    struct User
```

```
{
  public string Name;
  public int Age;
  public double Height;
  //конструктор, вызываемый при использовании new
  public User(string N, int A, double H)
  { Name = N;
    Age = A;
    Height = H;
  }
  //метод для ввода информации User
  public void Input()
  { Console.WriteLine("Введите имя:");
    Name = Console.ReadLine();
    Console.WriteLine("Введите возраст:");
    Age = int.Parse(Console.ReadLine());
    Console.WriteLine("Введите рост:");
    Height = double.Parse(Console.ReadLine());
  }
  //метод для вывода информации User
  public void Output()
  {
    Console.WriteLine("Имя: {0}, возраст: {1}, рост: {2}", Name, Age, Height);
  }
}; //конец описания struct User
//метод сортировки
static void sort(User[] a)
{ int i, k;
  User temp;
  for (k = 1; k < a.Length; k++)
    for (i = 0; i < a.Length - k; i++)
      if (a[i].Height> a[i + 1].Height)
      { temp = a[i];
        a[i] = a[i + 1];
        a[i + 1] = temp;
      }    }
//метод поиска
static void poisk(User[] a)
{ bool f = false;
  Console.WriteLine("Введите возраст для поиска: ");
  int age=int.Parse(Console.ReadLine());
  for (int i = 0; i < a.Length; i++)
    if (a[i].Age == age)
    { a[i].Output();
      f = true;      }
  if (f == false) Console.WriteLine("Нет в списке.");
}
static void Main()
  { Console.WriteLine("Введите количество записей");
    int n = int.Parse(Console.ReadLine());
    User[]SPISOK = new User[n];
  //вызывается конструктор для первого элемента
  SPISOK[0] = new User("Иванов И.", 37, 184.0);
  //ввод с клавиатуры остальных элементов
  for (int i = 1; i < n; i++)
  { Console.WriteLine("Введите {0}-ую запись:",i+1);
    SPISOK[i].Input();       }
  //вывод всех элементов на экран
  Console.WriteLine("Исходный список:");
  for (int i =0; i < n; i++)
  { Console.Write((i + 1)+": ");
    SPISOK[i].Output();       }
```

```
      sort(SPISOK);//вызов метода сортировки
      //вывод отсортированного массива на экран
      Console.WriteLine("Отсортированный список:");
      for (int i = 0; i < n; i++)
      { Console.Write((i + 1) + ": ");
        SPISOK[i].Output();    }
      poisk(SPISOK);//вызов метода поиска
      Console.ReadKey();
      }
  }}}
```

Задания

1 Описать структуру с именем STUDENT, содержащую следующие поля:

- NAME – фамилия и инициалы;
- GROUP – номер группы;
- SES – успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив STUD1, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию содержимого поля GROUP;

– вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4,0;

– если таковых студентов нет, вывести соответствующее сообщение.

2 Описать структуру с именем STUDENT, содержащую следующие поля:

- NAME – фамилия и инициалы;
- GROUP – номер группы;
- SES – успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив STUD1, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по алфавиту;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;

– если таковых студентов нет, вывести соответствующее сообщение.

3 Описать структуру с именем AEROFLOT, содержащую следующие поля:

- NAZN – название пункта назначения рейса;
- NUMR – номер рейса;
- TIP – тип самолета.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив AIRPORT, состоящий из семи элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;

– вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;

– если таких рейсов нет, выдать на дисплей соответствующее сообщение.

4 Описать структуру с именем WORKER, содержащую следующие поля:

- NAME – фамилия и инициалы работника;
- POS – название занимаемой должности;

– YEAR – год поступления на работу.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив TABL, состоящий из десяти структур типа WORKER; записи должны быть размещены по алфавиту;

– вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;

– если таких работников нет, вывести на дисплей соответствующие сообщение.

5 Описать структуру с именем TRAIN, содержащую следующие поля:

– NAZN – название пункта назначения;

– NUMR – номер поезда;

– TIME – время отправления.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив RASP, состоящий из восьми элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

– вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;

– если таких поездов нет, выдать на дисплей соответствующее сообщение.

6 Описать структуру с именем MARSH, содержащую следующие поля:

– BEGST – название начального пункта маршрута;

– TERM – название конечного пункта маршрута;

– NUMER – номер маршрута.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив TRAFIC, состоящий из восьми элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;

– вывод на экран информации о маршруте, номер которого введен с клавиатуры;

– если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

7 Описать структуру с именем NOTE, содержащую следующие поля:

– NAME – фамилия, имя;

– TEL – номер телефона;

– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BLOCKNOTE, состоящий их восьми элементов типа NOTE; записи должны быть размещены по алфавиту;

– вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;

– если таких нет, выдать на дисплей соответствующее сообщение.

8 Описать структуру с именем NOTE, содержащую следующие поля:

– NAME – фамилия, имя;

– TEL – номер телефона;

– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BLOCKNOTE, состоящий их восьми элементов типа NOTE; записи должны быть упорядочены по датам дней рождения;

– вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;

– если такого нет, выдать на дисплей соответствующее сообщение.

9 Описать структуру с именем ZNAK, содержащую следующие поля:

– NAME – фамилия, имя;
– ZODIAC – знак зодиака;
– BDAY – день рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив BOOK, состоящий из восьми элементов типа ZNAK; записи должны быть упорядочены по датам дней рождения;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры;

– если такого нет, выдать на дисплей соответствующее сообщение.

10 Описать структуру с именем PRICE, содержащую следующие поля:

– TOVAR – название товара;
– MAG – название магазина, в котором продается товар;
– STOIM – стоимость товара в рублях.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из восьми элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;

– вывод на экран информации о товаре, название которого введено с клавиатуры;

– если такого товара нет, выдать на дисплей соответствующее сообщение.

11 Описать структуру с именем ORDER, содержащую следующие поля:

– PLAT – расчетный счет плательщика;
– POL – расчетный счет получателя;
– SUMMA – перечисляемая сумма в рублях.

Написать программу, выполняющую следующие действия:

– ввод с клавиатуры данных в массив SPISOK, состоящий из восьми элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;

– вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;

– если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

6 Лабораторная работа № 18. Создание приложений Windows Form

Цель работы: основные приемы использования средств создания, выполнения и отладки Windows приложений с графическим интерфейсом пользователя.

Постановка задачи. Решение задач с использованием Windows Form.

6.1 Основные сведения

Для создания проекта Windows-приложения: загружаем Visual Studio, далее выбираем **Файл (File) →Создать (New) →Проект (Project)**, затем выбираем пункт Visual C# и отмечаем **Приложение Windows Forms (Windows Forms Application**), даем имя проекта в поле **Имя: (Name:),** местоположение в поле **Расположение: (Location:)** и жмем **Ok**.

Рассмотрим на примере создание приложения для решения следующей задачи: *вычисление площади круга и длины окружности по вводимому пользователем значению радиуса.*

Исходные данные: радиус окружности.

Выходные данные: площадь круга; длина окружности.

Интерфейс пользователя. Приложение содержит одну форму. Форма имеет стандартный для Windows-приложений общий вид (заголовок, кнопки закрытия приложения, свертывания в пиктограмму, разворачивания в полное окно и т. п.). На форме расположены элементы интерфейса пользователя, обеспечивающие функциональность разрабатываемого приложения. Элементы интерфейса пользователя и их расположение на форме показаны на рисунке 6.1. На изображение формы в дополнительных белых полях нанесены те имена элементов интерфейса, которые будут использованы для их идентификации в коде программы (свойство Name).

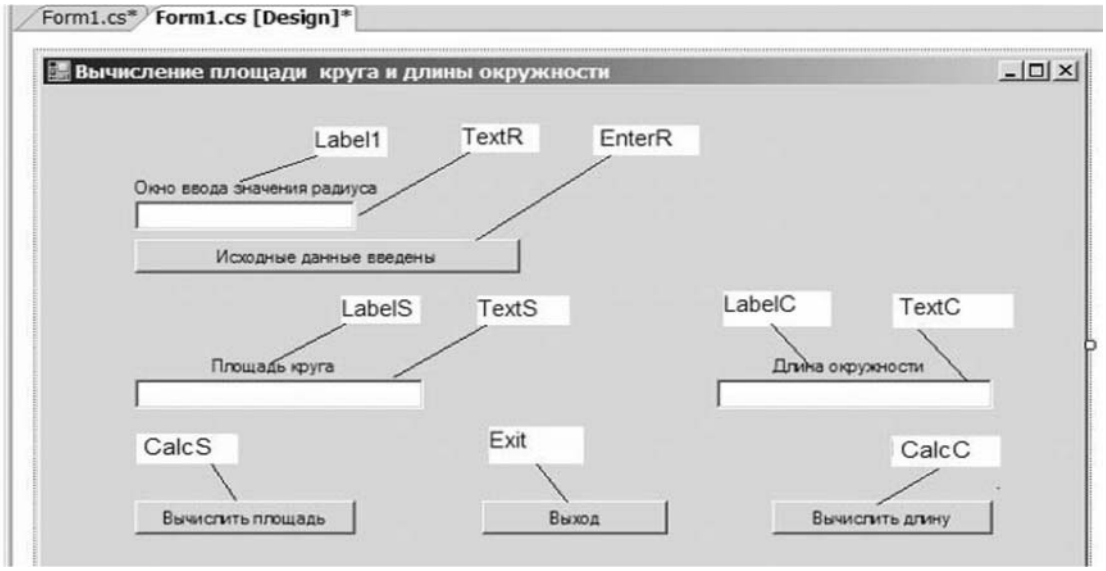


Рисунок 6.1 – Вид интерфейса пользователя

По функциональному назначению используемые в данной работе элементы интерфейса пользователя делятся на две группы:

1) элементы отображения (и ввода) текстовой информации;

2) элементы управления выполнением приложения.

Элементы первой группы предназначены для ввода исходных данных, вывода результатов вычислений и поясняющих надписей. К ним на рисунке 6.1 относятся:

– окно для ввода значения радиуса (TextR);

– текстовое поле с поясняющей надписью (LabelR);

– окно для вывода площади круга (TextS);

– текстовое поле с поясняющей надписью (LabelS);

– окно для вывода длины окружности (TextC);

– текстовое поле с поясняющей надписью (LabelC).

Элементы второй группы предназначены для ввода запросов («команд») пользователя. К ним относятся:

– кнопка подтверждения готовности исходных данных (EnterR);

– кнопка запроса на вычисление площади круга (CalcS);

– кнопка запроса на вычисление длины окружности (CalcC);

– кнопка завершения приложения (Exit).

Кнопки запросов («команд») на вычисления (CalcS, CalcC) должны в начале выполнения приложения находиться в пассивном состоянии и активизироваться только после нажатия кнопки подтверждения ввода исходных данных (EnterR).

Нажатие кнопки подтверждения ввода исходных данных (EnterR) сопровождается очисткой (стиранием) старых результатов в окнах вывода (TextS, TextC).

Подготовка экранной формы в режиме визуального программирования. Для изменения заголовка разрабатываемого приложения, например, «Вычисление площади круга и длины окружности» щелкните правой клавишей мыши на форме, выберите пункт Properties. Тем самым откроется или получит фокус ввода окно свойств формы. В этом окне найдите свойство Text и измените его значение.

Придайте форме требуемые размеры, перемещая мышью граничные рамки формы. Форма подготовлена и можно приступить к размещению на ней элементов интерфейса пользователя.

Рекомендуется следующая последовательность действий при разработке приложения:

– размещение на форме элементов интерфейса и коррекция их расположения;

– настройка свойства элементов;

– определение функциональности элементов интерфейса.

Элементы интерфейса добавляются в форму «перетаскиванием» значков нужных стандартных элементов из инструментальной панели Toolbox на изображение формы. Если панель инструментов не отображается, активизируйте ее через главное меню последовательностью **View -> ToolBox**.

Для разрабатываемого приложения разместите на форме:

– три окна для отображения текстовой информации (элемент типа **TextBox**);

– три панели для поясняющих надписей (элемент типа **Label**);

– четыре кнопки для управления приложением (элемент типа **Button**).

Выполните коррекцию размеров и взаимного положения элементов в соответствии с рисунком 6.2.

а) *б*)

Рисунок 6.2 – Изменение размера и перемещение элемента

При изменении размеров и перемещении элементов на форме отображаются горизонтальные и вертикальные линии индикации положения элемента относительно других элементов формы. На рисунке 6.2, *б* показаны вертикальные линии индикации положения элемента **button1** относительно элементов **label1** и **textbox1**.

Редактирование свойств элементов интерфейса пользователя позволяет изменить значения, установленные по умолчанию для этих элементов. Это можно сделать как при визуальном проектировании формы, так и программным способом во время выполнения приложения.

При визуальном программировании необходимо выбрать элемент, для которого нужно изменить свойства, и перейти в окно редактирования свойств этого элемента. Если окно свойств не отображается, его можно активизировать через меню **«View -> Properties Window»**. В окне свойств можно задать или изменить значения свойств элемента.

Одно из важных свойств элемента – его *имя*. По умолчанию элементу назначается *имя*, основанное на типе элемента и порядковом номере его размещения на форме. Однако, это имя по умолчанию не всегда отражает функциональное назначение элемента и может быть неудобным при чтении кода. Поэтому рекомендуется изменить стандартные имена элементов на более понятные имена, определенные разработчиком.

В приведенном примере, на рисунке формы, показаны белые поля, где мы планируем использовать определенные имена для элементов интерфейса. Чтобы присвоить элементам эти имена, необходимо для каждого элемента изменить значение свойства «*Name*». Например, для кнопки вычисления площади круга можно задать имя «*CalcS*».

Таким образом, редактирование свойств элементов интерфейса позволяет настраивать элементы по своим потребностям, включая изменение имен элементов для более понятного кода.

В соответствии с надписями на рисунке 6.1 измените стандартные имена элементов интерфейса (TextBox1 на TextR, Label1 на LabelR и т. д.).

Текстовая надпись. По умолчанию элементы интерфейса получают текстовую надпись, совпадающую со стандартным именем элемента. Например, для первой помещенной в форму кнопки (Button) текстовой надписью будет «**button1**». Для изменения текстовой надписи достаточно изменить значение свойства **Text**.

В нашем случае (в соответствии с приведенном на рисунке 6.1 изображением окна с интерфейсом пользователя) надпись должна иметь вид «**Окно ввода значения радиуса**». Аналогичным образом в соответствии с рисунком 6.1 задаем надписи для остальных элементов интерфейса:

– окно для ввода значения радиуса (элемент **TextBox** с именем **TextR**): пустая строка;

– поясняющая надпись (элемент **Label** с именем **LabelR**): «**Окно ввода значения радиуса**» и т.д. в соответствии с рисунком 6.1.

Свойство **Font** позволяет устанавливать шрифт надписи или текста в поле ввода-вывода. По умолчанию используется шрифт **MS Sans Serif 8**. Вы можете установить для каждого элемента тот шрифт, который кажется вам более подходящим. Для установки шрифта необходимо выбрать свойство **Font.**

Состояние элемента. По умолчанию элементы интерфейса находятся в активном состоянии, т. е. способны реагировать на события, к ним относящиеся. Для изменения состояния элемента необходимо изменить свойство **Enabled.** Значение этого свойства **true** соответствует активному состоянию элемента, а значение **false** – пассивному состоянию.

Согласно требованиям к интерфейсу пользователя для разрабатываемого приложения кнопки запроса на вычисление площади круга (**CalcS**) и запроса на вычисление длины окружности (**CalcC**) должны находиться в пассивном состоянии до нажатия кнопки подтверждения готовности исходных данных (элемент с именем **EnterR**).

Для установки желаемого состояния элемента необходимо в окне его свойств выделить строку **Enabled** и выполнить щелчок левой клавишей мыши на этой строке. Из списка возможных значений необходимо выбрать строку с нужным значением и подтвердить свой выбор щелчком левой клавиши мыши.

Аналогичным образом нужно установить в пассивное состояние кнопку запроса на вычисление длины окружности (элемент с именем **EnterC**).

Подготовка и редактирование исходного кода обработчиков событий. После включения в изображение формы элементов интерфейса и редактирования их свойств, исходный код программы будет размещен в файлах **Form1.Designer.cs, Program.cs** и **Form1.cs**.

Вначале файл **Form1.cs** будет содержать только следующий код, реализующий инициализацию формы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Windows.Forms;
namespace winPro {
public partial class Form1 : Form
```

```
{    public Form1()
      {InitializeComponent(); }    }}
```

Для вычисления площади круга и длины окружности по заданному радиусу необходимо добавить в код файла **Form1.cs** средства для обработки событий, связанных с активными элементами управления, размещенными в форме.

Определение функциональности приложения производится выявлением событий, на которые должны реагировать элементы интерфейса, и размещением в файле **Form1.cs** кодов обработчиков этих событий.

В разрабатываемом приложении событиями будут:

– щелчок левой клавиши мыши на кнопке подтверждения готовности исходных данных (**EnterR**);

– щелчок левой клавиши мыши на кнопке вычисления площади круга (**CalcS**);

– щелчок левой клавиши мыши на кнопке вычисления длины окружности (**CalcC**);

– щелчок левой клавиши мыши на кнопке завершения приложения (**Exit**).

Для обработки событий в коде должны присутствовать соответствующие методы. Заготовки этих методов добавляются в текст файла **Form1.cs** с помощью инструментария Microsoft Visual Studio.

Перейдите в режим редактирования формы командой **View -> Designer** или щелчком левой клавишей мыши на закладке **Form1.cs[Design].** Выполните двойной щелчок левой клавишей мыши на соответствующей кнопке в визуальном представлении формы. В исходный код будет добавлен код заготовки метода, реализующего обработчик события, при этом автоматически произойдет переход в режим редактирования кода.

Заготовка обработчика, которая будет добавлена в исходный код после двойного щелчка на кнопке подтверждения готовности исходных данных (EnterR):

```
private void EnterR_Click(object sender, EventArgs e)
{    }
```

Имя метода, реализующего обработку события, будет состоять из имени элемента интерфейса (EnterR) и имени события (Click). Имя элемента и имя события разделяются (соединяются) знаком подчеркивания. Тело метода пусто – сюда необходимо вручную добавить нужные операторы, снабдив их комментариями.

Обратившись к постановке задачи, видим, что при нажатии на кнопку готовности исходных данных необходимо перевести кнопки «вычислить площадь круга» и «вычислить длину окружности» в активное состояние и очистить окна вывода предыдущих результатов (TextS, TextC). Метод обработки события «щелчок левой клавиши мыши на кнопке подтверждения готовности исходных данных» примет вид:

```
// Подтверждение готовности исходных данных
private void EnterR_Click(object sender, System.EventArgs e)
{    CalcS.Enabled = true; // Перевод кнопок ВЫЧИСЛИТЬ
    CalcC.Enabled = true; // в активное состояние
    TextS.Text = ""; // Очистка окон
    TextC.Text = ""; // вывода результатов     }
```

Обратите внимание, что в тексте обработчика события программно изменяются те самые свойства элементов интерфейса, которые доступны в режиме визуального проектирования формы. Для обращения к свойству используется конструкция

```
имя_элемента.название_свойства
```

Поэтому для программирования необходимо знать имена элементов (значение их свойства Name), названия нужных свойств и допустимые для этих свойств значения.

Аналогичным образом определяются методы обработки других событий.

```
//Вычисление площади круга:
private void CalcS_Click(object sender, System.EventArgs e)
{       double r,s;
       r = double.Parse(TextR.Text); // Преобразуем строку символов
       // из окна ввода в значение радиуса
       s = Math.PI * r * r; // Вычисляем площадь
       TextS.Text = s.ToString(); // Преобразуем площадь в строку
       // и отображаем ее в окне вывода
       CalcS.Enabled = false; // Переводим кнопку ВЫЧИСЛИТЬ в пассивное состояние
       }
//Вычисление длины окружности:
private void CalcC_Click(object sender, System.EventArgs e)
{       double r,c;
        r = double.Parse(TextR.Text); //Преобразуем строку символов
        //из окна ввода в значение радиуса
        c = 2.0 * Math.PI * r; //Вычисляем длину
        TextC.Text = c.ToString(); //Преобразуем длину в строку
        //и отображаем ее в окне вывода
        CalcC.Enabled = false; //Переводим кнопку ВЫЧИСЛИТЬ в пассивное состояние
        }
//Выход из приложения:
private void Exit_Click(object sender, System.EventArgs e)
{       Close();}  //Завершить приложение
```

Прежде чем переходить к трансляции и выполнению приложения, просмотрите полученный исходный код. Возможно, что в исходном коде появились заготовки обработчиков событий, которые вы определять не собирались. Это может произойти при случайном двойном щелчке на элементе интерфейса или на форме. Например, при случайном двойном щелчке на форме в исходный код будет добавлена заготовка обработчика события – «загрузка формы»:

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

Ссылка на все обработчики занесена в файл **Form1.Designer.cs .** После ручного удаления заготовки из файла **Form1.cs,** ссылка на нее останется, что приведет к ошибке при запуске приложения.

Для корректного удаления обработчика используйте специальные средства интегрированной среды. Для этого откройте дизайнер форм и на изображении формы выполните одинарный щелчок левой кнопкой мыши. Затем откройте окно свойств **View -> Properties Window** (иначе – щелчок правой кнопкой мыши на форме и затем выбор пункта **Properties**). В окне свойств выполните щелчок левой клавиши мыши на кнопке событий (с изображением «молнии»), как показано на рисунке 6.3.

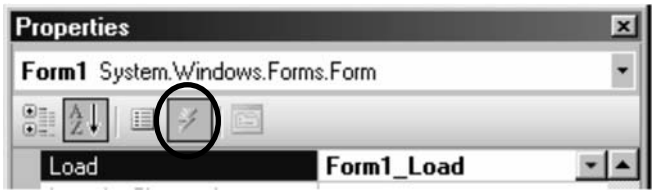


Рисунок 6.3 – Активизация списка событий с лишним обработчиком

В списке событий найдите строку с именем удаляемого обработчика (в данном случае это **Form1_Load**) и сотрите имя обработчика. Заготовка обработчика будет удалена из кода в файле **Form1.cs**, кроме того, будут удалены ссылки на этот обработчик в других файлах.

Выполнение приложения в интегрированной среде. Для запуска приложения выполните команду **Debug -> Start Without Debugging** (или сочетание клавиш **Ctrl+F5**). По этой команде последовательно реализуются два процесса.

1 *Построение решения*. Исходный код программы транслируется в промежуточный код, который сохраняется в файле **winPro.exe.**

2 *Выполнение решения*. Автоматически запускается среда исполнения, в которой выполняется код, содержащийся в файле **winPro.exe.**

Задания

1 Ввести длины катетов a и b прямоугольного треугольника. Найти его периметр и площадь.

2 Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

3 Найти длину окружности и площадь круга заданного радиуса R.

4 Найти площадь кольца, внутренний радиус которого равен $R1$, а внешний радиус равен $R2$ ($R1 < R2$).

5 Дана сторона равностороннего треугольника. Найти площадь этого треугольника и радиусы вписанной и описанной окружностей.

6 Дана длина окружности. Найти площадь круга, ограниченного этой окружностью.

7 Найти расстояние между двумя точками с заданными координатами ($x1$, $y1$) и ($x2$, $y2$).

8 Ввести с клавиатуры координаты трех вершин треугольника ($x1$, $y1$), ($x2$, $y2$), ($x3$, $y3$). Найти его периметр и площадь.

9 Найти действительные корни квадратного уравнения $A \cdot x^2 + B \cdot x + C = 0$, заданного своими коэффициентами A, B, C (коэффициент $A \neq 0$).

10 Дано целое четырехзначное число. Используя операции деления / и нахождения остатка от деления %, найти сумму и произведение его цифр.

11 Скорость лодки в стоячей воде V км/ч, скорость течения реки U км/ч ($U < V$). Время движения лодки по озеру $T1$ ч, а по реке (против течения) – $T2$ ч. Определить путь S, пройденный лодкой.

12 Скорость первого автомобиля $V1$ км/ч, второго – $V2$ км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили удаляются друг от друга.

7 Лабораторная работа № 19. Дополнительные компоненты Windows Form

Цель работы:
– создание приложения с несколькими формами;
– изучение дополнительных визуальных компонент.

Постановка задачи. Решение задач с использованием Windows Form.

7.1 Основные сведения

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выберем Add(Добавить)->Windows Form...

Для вызова второй формы (например, вторая форма называется Form2), сначала создается объект данного класса, а потом для его отображения на экране вызываем метод Show или ShowDialog:

```
private void button1_Click(object sender, EventArgs e)
{    Form2 newForm = new Form2();
    newForm.Show();  }
```

Теперь сделаем чтобы вторая форма воздействовала на первую. Для этого нужно второй форме передать сведения о первой форме. Для этого воспользуемся передачей ссылки на форму в конструкторе. Поскольку C# поддерживает перегрузку методов, то можно создать несколько методов и конструкторов с разными параметрами и в зависимости от ситуации вызывать один из них. Итак, изменим файл кода второй формы на следующий:

```
public partial class Form2 : Form
    {
        public Form2()
        {            InitializeComponent();        }
// добавим конструктор с параметром
        public Form2(Form1 f)
        {   InitializeComponent();
            f.BackColor = Color.Yellow;
        }     }
```

Теперь в коде первой формы, где вызвана вторая форма, изменив вызов:

```
private void button1_Click(object sender, EventArgs e)
{   Form2 newForm = new Form2(this);
    newForm.Show();    }
```

Поскольку в данном случае ключевое слово this представляет ссылку на текущий объект – объект Form1, то при создании второй формы она будет получать ее (ссылку) и через нее управлять первой формой.

Для создания меню в Windows Forms применяется элемент MenuStrip. Данный класс унаследован от ToolStrip и поэтому наследует его функциональность.

Наиболее важные свойства компонента MenuStrip.

Dock: прикрепляет меню к одной из сторон формы

LayoutStyle: Задает ориентацию панели меню на форме. Может принимать значения: ***HorizontalStackWithOverflow***: расположение по горизонтали с переполнением, если длина меню превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, т. е. панель переполняется элементами; ***StackWithOverflow***: элементы располагаются автоматически с переполнением; ***VerticalStackWithOverflow***: элементы располагаются вертикально с переполнением; ***Flow***: элементы размещаются автоматически, но без переполнени, если длина панели меню меньше длины контейнера, то выходящие за границы элементы переносятся; ***Table***: элементы позиционируются в виде таблицы.

ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню.

Stretch: позволяет растянуть панель по всей длине контейнера.

TextDirection: задает направление текста в пунктах меню.

Добавить новые элементы в меню можно в режиме дизайнера. MenuStrip выступает своего рода контейнером для отдельных пунктов меню, которые представлены объектом ***ToolStripMenuItem***.

Назначив обработчики для события ***Click***, мы можем обработать нажатия на пункты меню.

Если надо быстро обратиться к какому-то пункту меню, то могут использоваться клавиши быстрого доступа. Для задания клавиш быстрого доступа используется свойство ***ShortcutKeys***:

```
saveItem.ShortcutKeys = Keys.Control | Keys.P;
```

Пример – Создать меню с командами **Ввод данных**, **Изменение размеров**, **Выход**. Команда **Изменение размеров** при запуске приложения недоступна.

При выборе команды **Выход** приложение завершает работу. При выборе команды **Ввод данных** открывается диалоговое окно, содержащее:

– два поля ввода типа **TextBox** с метками **По горизонтали**, **По вертикали**;

– группу из двух переключателей **Уменьшение**, **Увеличение** типа RadioButton;

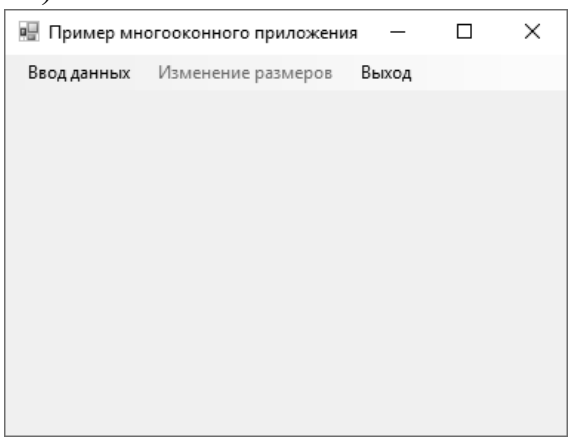
– две кнопки типа Button.

Обеспечить возможность ввода значений в поля **Размер по горизонтали** и **Размер по вертикали**. Значения интерпретируются как количество пикселей, на которое нужно изменить размеры главного окна: увеличить или уменьшить в зависимости от положения переключателей.

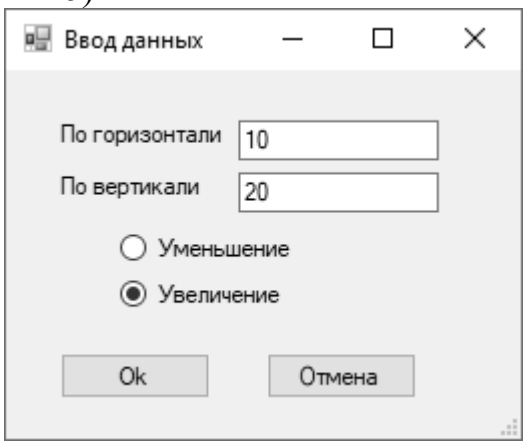
После ввода значений команда **Изменение размеров** становится доступной. При выборе этой команды размеры главного окна увеличиваются или уменьшаются на введенное в диалоговом окне количество пикселей.

Вид главного окна и диалогового окна работающего приложения показан на рисунке 7.1.

а)



б)



а – главное окно; *б* – диалоговое окно

Рисунок 7.1 – Демонстрация работы приложения

Код файла **Form1.cs** главной формы приложения.

```
public partial class Form1 : Form
 {
  // horiz – величина изменения размера по горизонтали
  // vert– величина изменения размера по вертикали
  int horiz, vert;
  // dec – признак уменьшения размера
  // inc– признак увеличения размера
  bool dec, inc;
  public Form1()
  {
InitializeComponent();
  }
  // Выбор пункта меню Ввод данных
  private void вводДанныхToolStripMenuItem_Click(object sender, EventArgs e)
  {
Form2 f2 = new Form2(); // Создать объект второй формы
// Загружаем 2-ю форму и если пользователь нажал ввел данные в ней и нажал Ok
if (f2.ShowDialog() == DialogResult.OK)
{
    // получить из второй формы величину изменения по горизонтали
    horiz = f2.H;
    // получить из второй формы величину изменения по вертикали
    vert = f2.W;
    // получить из второй формы направление изменения «уменьшение»
    dec = f2.R1;
    // получить из второй формы направление изменения «увеличение»
    inc = f2.R2;
   // Сделать доступным пункт меню для изменения размеров окна
   измененияРазмеровToolStripMenuItem.Enabled = true;
}
  }
  // Выбор пункта меню Изменение размеров
  private void измененияРазмеровToolStripMenuItem_Click(object sender, EventArgs e)
  {
if (inc == true)
{
    Height += vert;
    Width += horiz;
}
if (dec == true)
```

```
{
    Height -= vert;
    Width -= horiz;
}
измененииРазмеровToolStripMenuItem.Enabled = false;
  }
  //Выбор пункта меню Выход
  private void выходToolStripMenuItem_Click(object sender, EventArgs e)
  {
Close();  }
    }
```

Код файла **Form2.cs** диалогового окна:

```
public partial class Form2 : Form
    {
  int h, w;
  public Form2()
  {
InitializeComponent();
//устанавливаем свойства кнопок для выхода из модального окна
button1.DialogResult = DialogResult.OK;
button2.DialogResult = DialogResult.Cancel;
  }
  // Метод-свойство для получения размера изменения по горизонтали
  public int H
  {    get { return h; }
  }
  // Метод-свойство для получения размера изменения по вертикали
  public int W
  {    get { return w;}
  }

  // Метод-свойство для получения значения состояния первой радиокнопки
  public bool R1
  {    get { return radioButton1.Checked; }
  }
  // Метод-свойство для получения значения состояния второй радиокнопки
  public bool R2
  {    get { return radioButton2.Checked; }
  }
  // Нажатие кнопки Ok
  private void button1_Click(object sender, EventArgs e)
  {
    try
    {    h = Convert.ToInt32(textBox1.Text);
    }
    catch (FormatException)
    {    MessageBox.Show("Ошибка ввода размера по горизонтали!");
        return;
    }
    try
    {     w = Convert.ToInt32(textBox2.Text); }
    catch (FormatException)
    {    MessageBox.Show("Ошибка ввода размера по вертикали!");
        return;
    }
    if (radioButton1.Checked == false && radioButton2.Checked ==false)
    {    MessageBox.Show("Вы не задали направление изменения!");
        return;}  }  }
```

Задания

Написать Windows-приложение, заголовок главного окна которого содержит ФИО, группу и номер варианта. В программе должна быть предусмотрена обработка исключений, возникающих из-за ошибочного ввода.

1 Создать меню с командами **Ввод**, **Вычисления** и **Выход**.

Команда **Вычисления** после запуска программы недоступна.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– три поля типа **TextBox** для ввода длин трех сторон треугольника;

– группу из двух флажков **Периметр** и **Площадь** типа **CheckBox**;

– кнопку типа **Button**.

Необходимо обеспечить возможность:

– ввода длин трех сторон треугольника;

– выбора режима с помощью флажков: подсчет периметра и/или площади треугольника.

После ввода значений команда **Вычисления** становится доступной. При выборе команды **Вычисления** должно открываться диалоговое окно с результатами. При выборе команды **Выход** работа приложения должна завершиться.

2 Создать меню с командами **Ввод**, **Вычисления** и **Выход**.

Команда **Вычисления** после запуска программы недоступна.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– три поля ввода типа **TextBox** с метками **Радиус**, **Высота**, **Плотность**;

– группу из двух флажков **Объем** и **Масса** типа **CheckBox**;

– кнопку типа **Button**.

Необходимо обеспечить возможность:

– ввода радиуса, высоты и плотности конуса;

– выбора режима с помощью флажков: подсчет объема и/или массы конуса.

При выборе команды **Вычисления** должно открываться окно сообщений с результатами. При выборе команды **Выход** приложение должно завершить работу.

3 Создать меню с командами **Ввод**, **Результат**, **О программе**.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– поле ввода типа **TextBox** с меткой **Ввод word**;

– группу из двух переключателей **Upper case** и **Lower case** типа **RadioButton**;

– кнопку типа **Button**.

Необходимо обеспечить возможность ввода слова и выбора режима перевода в верхний или нижний регистр в зависимости от положения переключателей.

При выборе команды **Результат** должно открываться диалоговое окно с результатом перевода. При выборе команды **О программе** должно открываться окно с информацией о разработчике.

4 Написать приложение, которое по заданным в файле исходным данным выводит информацию о компьютерах. Создать меню с командами **Choose**, **Результат**, **Выход**. Команда **Результат** недоступна.

При запуске приложения из файла должны читаться исходные данные. Файл необходимо сформировать самостоятельно. Каждая строка файла должна содержать тип компьютера, цену (**price**) и емкость жесткого диска (**hard drive**).

При выборе команды **Choose** должно открываться диалоговое окно, содержащее:

– поле типа **TextBox** для ввода минимальной емкости диска;

– поле типа **TextBox** для ввода максимальной приемлемой цены;

– группу из двух переключателей **Hard drive** и **Price** типа **RadioButton**;

– кнопки **OK** и **Cancel** типа **Button**.

После ввода всех данных команда меню **Результат** становится доступной. Эта команда должна открывать диалоговое окно, содержащее список компьютеров, удовлетворяющий введенным ограничениям и упорядоченный по отмеченной характеристике. Команда **Выход** должна завершать работу приложения.

5 Создать меню с командами **Ввод**, **Вычислить**, **О программе**.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– три поля ввода типа **TextBox** с метками **Number1**, **Number2**, **Number3**;

– группу из двух флажков **Summ** и **Multiply** типа **CheckBox**;

– кнопку типа **Button**.

Необходимо обеспечить возможность ввода трех чисел и выбора режима вычислений с помощью флажков: подсчет суммы трех чисел (**Summ**) и/или произведения двух первых чисел (**Multiply**).

При выборе команды **Вычислить** должно открываться диалоговое окно с результатами. При выборе команды **О программе** должно открываться окно с информацией о разработчике.

6 Создать меню с командами **Ввод**, **Вычислить**, **Выход**.

Команда **Вычислить** после загрузки приложения недоступна.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– два поля ввода типа **TextBox** с метками **Number 1** и **Number 2**;

– группу из двух флажков **Min** и **Max** типа **CheckBox**;

– кнопку типа **Button**.

Необходимо обеспечить возможность:

– ввода двух чисел;

– выбора режима вычислений с помощью флажков вычисления минимального или максимального значения двух чисел.

При выборе команды **Вычислить** должно открываться окно сообщений с результатами. При выборе команды **Выход** работа приложения должна завершиться.

7 Создать меню с командами **Ввод цвета**, **Изменить, Выход**, **Помощь**.

При выборе команды **Ввод цвета** должно открыться диалоговое окно, содержащее:

– три поля ввода типа **TextBox** с метками **Red**, **Green**, **Blue**;

– группу из двух флажков **Left** и **Right** типа **CheckBox**;

– кнопку типа **Button**.

Необходимо обеспечить возможность ввода RGB-составляющих цвета.

При выборе команды **Изменить** цвет главного окна (левой, правой или обеих половин окна в зависимости от установки флажков) должен изменяться на заданный. При выборе команды **Выход** приложение должно завершить работу.

8 Создать меню с командами **Ввод цвета**, **Изменение цвета**, **Очистка**.

Команда **Изменение цвета** после запуска программы недоступна.

При выборе команды **Ввод цвета** должно открываться диалоговое окно, содержащее:

– группу из двух флажков **Up** и **Down** типа **CheckBox**;

– группу из трех переключателей **Red**, **Green**, **Blue** типа **RadioButton**;

– кнопку типа **Button**.

Необходимо обеспечить возможность:

– выбора цвета с помощью переключателей;

– ввода режима, определяющего, какая область закрашивается: все окно, его верхняя или нижняя половина.

После закрытия окна ввода данных команда **Изменение цвета** становится доступной. При выборе команды **Изменение цвета** цвет главного окна (верхней, нижней или обеих половин в зависимости от введенного режима) должен изменяться на заданный. При выборе команды **Очистка** должен восстанавливаться первоначальный цвет окна.

9 Создать меню с командами **Перевод**, **Результат**, **О программе**, **Выход**. Команда **Результат** после загрузки приложения недоступна.

При выборе команды **Перевод** должно открываться диалоговое окно, содержащее:

– поле ввода типа **TextBox** с меткой **Binary number**;

– поле ввода типа **TextBox** для вывода результата (**read-only**);

– группу из трех переключателей **8**, **10**, **16** типа **RadioButton**;

– кнопку **Do** типа **Button**.

Необходимо обеспечить возможность:

– ввода числа в двоичной системе в поле **Binary number**;

– выбора режима преобразования с помощью переключателей: перевод в восьмеричную, десятичную или шестнадцатеричную систему счисления. При выборе команды **Результат** должно открываться окно сообщений с результатами. должен появляться результат перевода. При выборе команды **Выход** приложение должно завершить работу.

10 Создать меню с командами **Ввод**, **Результат**, **Выход**.

При выборе команды **Ввод** должно открываться диалоговое окно, в котором обеспечиваться возможность ввода координат двух точек и выбора режима с помощью флажков **length** и **koef**: подсчет длины отрезка, соединяющего эти точки, и/или углового коэффициента.

При выборе команды **Результат** должно открываться окно сообщений с результатами подсчета. При выборе команды **Выход** приложение должно

завершить работу.

11 Создать меню с командами **Ввод**, **Результат**, **О программе**.

При выборе команды **Ввод** должно открываться диалоговое окно, содержащее:

– два поля ввода типа **TextBox**;

– группу из двух переключателей **First tetter** и **All letters** типа **RadioButton**;

– кнопку типа **Button**.

Необходимо обеспечить возможность ввода предложения и выбора режима его преобразования: либо начинать с прописной буквы каждое слово (**First letter**), либо перевести все буквы в верхний регистр (**All letters**).

При выборе команды **Результат** должно открываться диалоговое окно с результатом преобразования. При выборе команды **О программе** должно открываться окно с информацией о разработчике.

12 Написать анализатор текстовых файлов, выводящий информацию о количестве слов в тексте, а также статистическую информацию о введенной пользователем букве.

Создать следующую систему меню:

– **Файл (Загрузить текст**; **Выход)**;

– **Анализ (Количество слов**; **Повторяемость буквы)**.

При выборе файла для загрузки необходимо использовать объект типа **OpenFileDialog**. При выборе команды **Количество слов** программа должна вывести в окно сообщений количество слов в тексте. При выборе команды **Повторяемость буквы** программа должна предложить пользователю ввести букву, а затем она должна вывести в окно сообщений количество ее повторений без учета регистра.

8 Лабораторная работа № 20. Построение графиков

Цель работы:

– создание приложения с несколькими формами;

– изучение дополнительных визуальных компонент.

Постановка задачи. Построение графиков функций с использованием Windows Form.

8.1 Основные сведения

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации. Элемент управления **Chart** находится на **Панеле элементов (ToolBox)** в категории **Данные (Data).**

Построение графиков с помощью компонента Chart.

Пример – Составить программу, отображающую графики функций sin(x) и cos(x) на интервале [*Xmin*, *Xmax*]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего следует определить в коде класса все необходимые переменные и константы.

```
        private double XMin = -Math.PI; // Левая граница графика
        private double XMax = Math.PI; // Правая граница графика
        private double Step = (Math.PI * 2) / 10; // Шаг графика
        private double[] x;// Массив значений X - общий для обоих графиков
        private double[] y1; // Два массива Y - по одному для каждого графика
        private double[] y2;
```

Далее следует определить метод, который будет рассчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы *x*, *y*1 и *y*2.

```
// Расчёт значений графика
private void CalcFunction()
        {   // Количество точек графика
            int count = (int)Math.Ceiling((XMax - XMin) / Step)+ 1;
            // Создаём массивы нужных размеров
            x = new double[count];
            y1 = new double[count];
            y2 = new double[count];
            // Расчитываем точки для графиков функции
            for (int i = 0; i < count; i++)
            {   // Вычисляем значение X
                x[i] = XMin + Step * i;
                // Вычисляем значение функций в точке X
                y1[i] = Math.Sin(x[i]);
                y2[i] = Math.Cos(x[i]);
            }       }
```

После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Для этого нужно добавить вторую диаграмму и настроить внешний вид осей.

```
/// Настроим свойства элемента управления chart1
        private void CreateChart()
        {   //Многие свойства можно задать через оно "Свойства" объекта Chart
            // Задаём размеры элемента
            chart1.SetBounds(10, 10, this.Width - 20, this.Height - 20);
            // Задаём левую и правую границы оси X
            chart1.ChartAreas[0].AxisX.Minimum = XMin;
            chart1.ChartAreas[0].AxisX.Maximum = XMax;
            // Определяем шаг сетки
            chart1.ChartAreas[0].AxisX.MajorGrid.Interval = Step;
            // Задаём тип графика - сплайны
            chart1.Series[0].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;
            // Указываем ширину линии графика
            chart1.Series[0].BorderWidth = 3;
            // Название графика для отображения в легенде
            chart1.Series[0].LegendText = "sin(x)";
            // Добавляем в список графиков вторую диаграмму
            chart1.Series.Add("Series2");
            // Аналогичные действия для второго графика
            chart1.Series[1].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;
            chart1.Series[1].BorderWidth = 3;
            chart1.Series[1].LegendText = "cos(x)";        }
```

Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись, например, сразу после запуска программы, надо вызывать их в обработчике события Load формы.

```
private void Form1_Load(object sender, EventArgs e)
{
    // Создаём элемент управления
    CreateChart();
    // Расчитываем значения точек графиков функций
    CalcFunction();
    // Добавляем вычисленные значения в графики
    chart1.Series[0].Points.DataBindXY(x, y1);
    chart1.Series[1].Points.DataBindXY(x, y2);        }
```

Задание

Составить программу для табулирования функции *f*(*x*) на интервале [*Хнач*, *Хкон*] с шагом dx.

Значения *Хнач*, *Хкон* и коэффициенты функции вводятся в текстовые поля. Предусмотреть проверку на корректность ввода, используя различные обработчики событий и элемент управления ErrorProvider.

Значение *dx* выбирается из списка.

Табулируемая функция выбирается при помощи радиопереключателей.

Вывести в список значения *x* и *f*(*x*).

В каждый момент работы программы на форме должны располагаться только необходимые элементы управления. При изменении размеров окна взаимное расположение элементов управления и положение относительно границ окна должно сохраняться.

Построить графики одной из приведенных функций:

1) $f(x) = -\cos^k(\pi x^m / 2) + 2x^j$;
2) $f(x) = \sin^k(\pi x^m / 2) - (1-x)^j$;
3) $f(x) = 1 - x^j - \mathrm{tg}^k(\pi x^m / 4)$;
4) $f(x) = (1-x)^j - \mathrm{tg}^k(\pi x^m / 4)$;
5) $f(x) = (1-x)^{1/j} - \mathrm{tg}^k(\pi x^m / 4)$;
6) $f(x) = e^{-x^j} - \sin(\pi x^m / 2)$;
7) $f(x) = e^{-x^j} - x^k \sin(\pi x^m / 2)$;
8) $f(x) = (1-x)^j \,\mathrm{ch}^k\, x^m - x^l$;
9) $f(x) = (1-x)^j \cos^k(\pi x^m / 2) - x^l$;
10) $f(x) = (1-x)^j - \mathrm{sh}^k\, x^m$;
11) $f(x) = (1-x)^j \,\mathrm{ch}^k\, x^m - x^l$;
12) $f(x) = \arcsin x^m - (1-x^j)^k$.

В функциях параметры *l*, *k*, *i*, *m* принимают значения от 1 до 4.

Список литературы

1 **Гуриков, С. Р.** Введение в программирование на языке Visual C#: учебное пособие / С. Р. Гуриков. – Москва: ФОРУМ; ИНФРА-М, 2020. – 447 с.

2 **Дадян, Э. Г.** Современные технологии программирования. Язык C#: учебник: в 2 т. / Э. Г. Дадян. – Москва : ИНФРА-М, 2021. – Т. 1. – 312 с.

3 **Гагарина, Л. Г.** Технология разработки программного обеспечения: учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул; под ред. Л. Г. Гагариной. – Москва: ФОРУМ; ИНФРА-М, 2019. – 400 с.

4 **Немцова, Т. И.** Программирование на языке высокого уровня. Программирование на языке C++: учебное пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев; под ред. Л. Г. Гагариной. – Москва: ФОРУМ; ИНФРА-М, 2021. – 512 с.