ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

СБОРНИК УПРАЖНЕНИЙ

по дисциплине

«Языки программирования»

Авторы
к.т.н., доцент Галушка В.В.
к.т.н., доцент Айдинян А.Р.

Ростов-на-Дону, 2014

Оглавление

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: Среда программирования Visual Studio. Линейные алгоритмы.

Цель: Изучение среды программирования Visual Studio и основных видов проектов. Знакомство с языком программирования C#.

Теоретические сведения

1. Создание проекта.

Для выполнения лабораторных работ рекомендуется использовать среду программирования Visual Studio версии не ниже 2008.

Интерфейс среды программирования представлен на рис. 1. Для того чтобы начать писать программу необходимо создать проект, для этого нажать на надпись «Create: Project…» либо выбрать пункт меню «File» → «New» → «Project…». Откроется окно, в котором можно выбрать тип проекта и указать его название (рис. 2). Для начала изучения языка C# лучше выбирать тип проекта «Console Application».

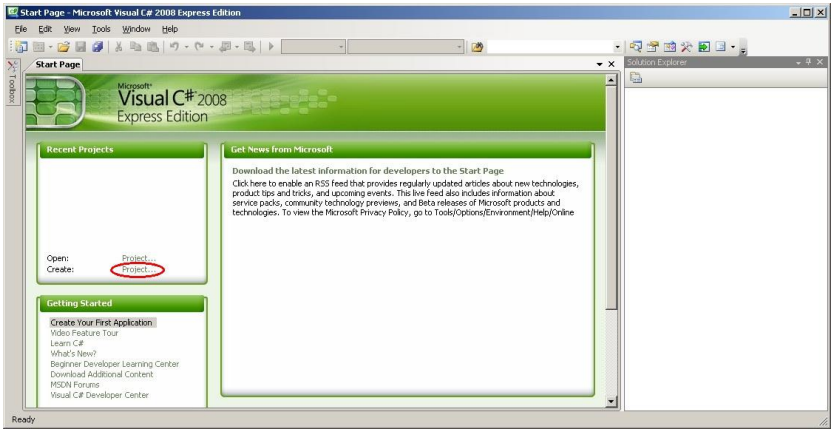


Рис. 1 — Стартовая страница среды программирования.

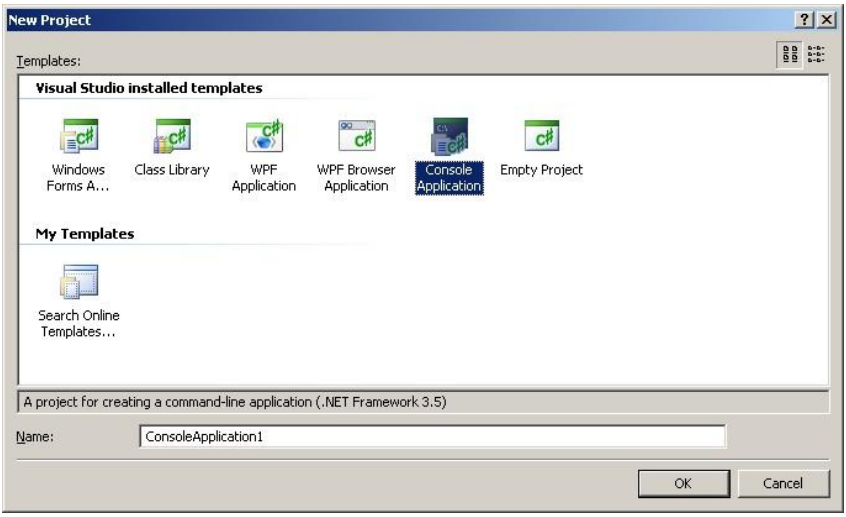


Рис. 2 — Окно создания проекта.

В поле «Name:» можно ввести название проекта. После нажатия кнопки «Ok» средой программирования будет автоматически сгенерированы несколько файлов, входящих в проект, в том числе файла с настройками и каркас файла с исходным кодом программы, который откроется в рабочей области среды программирования (рис. 3).

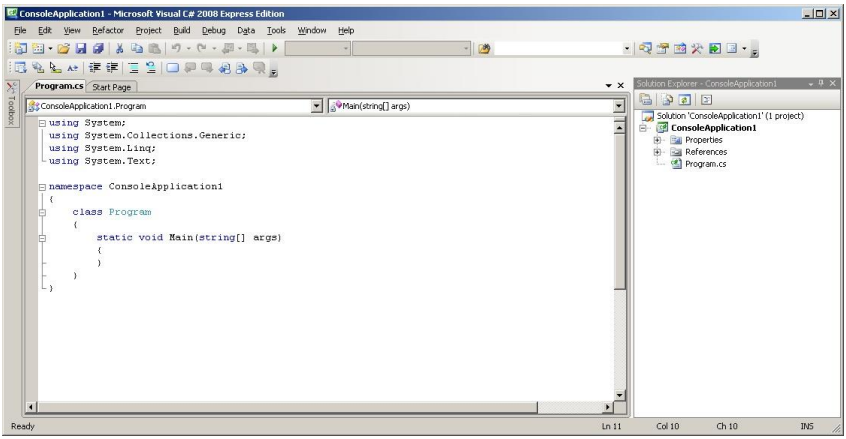


Рис. 3 — Исходный код и структура проекта.

Несмотря на то, что никаких действий в данном коде пока не описано, его уже можно скомпилировать и запустить. Для это-

го нужно выбрать пункт меню «Debug» → «Start Without Debugging» или нажать сочетание клавиш Ctrl F5.

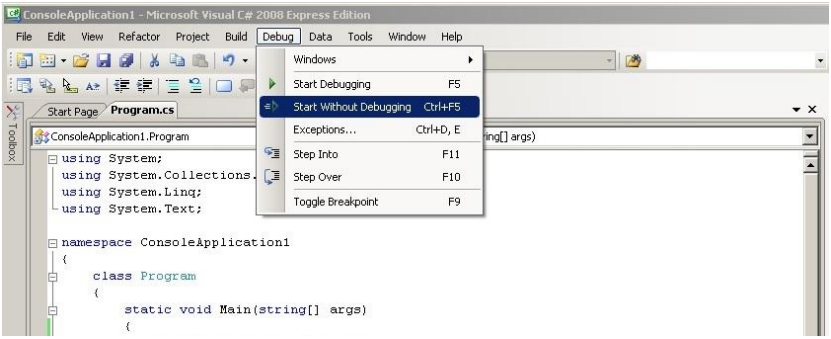


Рис. 4 — Запуск проекта.

2. Операторы ввода-вывода

Для ввода данных в программу используется функция Console.ReadLine(). Для вывода результатов работы функция Console.WriteLine(значение).

Рассмотрим простой пример программного кода.

```
Console.WriteLine("Введите имя");
string name = "";
name = Console.ReadLine();
Console.WriteLine("Привет, " + name);
```

Данная программа сначала выведет на экран сообщение «Введите имя» и будет ждать пользовательского ввода. После нажатия «Enter» программа выведет приветствие того, чьё имя было введено.

Описанным выше образом можно вводить в программу строковые данные. Для ввода числовых данных, их необходимо преобразовывать к соответствующему типу. Для этого используется следующее выражение:

`int.Parse(Console.ReadLine())` , если необходимо вводить целые числа

и

`double.Parse(Console.ReadLine())` , если необходимо вводить числа с дробной частью.

Например:

```
int a = int.Parse(Console.ReadLine());
```

3. Математические функции

Для вычисления математических и тригонометрических функций, таких как синус, косинус, тангенс, возведения в степень и т.д. используются следующие выражения:

`Math.Sin(a)` — синус a.
`Math.Asin(a)` — арксинус a.
`Math.Cos(a)` — косинус a.
`Math.Acos(a)` — арккосинус a.
`Math.Tan(a)` — тангенс a.
`Math.Atan(a)` — арктангенс a.
`Math.Pow(a,b)` — возведение числа a в степень b.
`Math.Log(a,b)` — логарифм числа a по основанию b.
`Math.Log(a)` — натуральный логарифм числа a.

Задание

Написать программу для выполнения расчётов по двум формулам, приведённым в задании. Результат вычисления по первой формуле должен совпадать с результатом, полученным по второй.

Вариант 1

$$z_1 = 2\sin^2(3\pi - 2\alpha)\cos^2(5\pi + 2\alpha)$$

$$z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$$

Вариант 2

$$z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$$

$$z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$$

Вариант 3

$$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$$

$$z_2 = 2\sin\alpha$$

Вариант 4

$$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha}$$

$$z_2 = \operatorname{tg} 3\alpha$$

Вариант 5

$$z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$$

$$z_2 = \cos^2\alpha + \cos^4\alpha$$

Вариант 6

$$z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$$

$$z_2 = 4\cos\frac{\alpha}{2}\cdot\cos\frac{5}{2}\alpha\cdot\cos 4\alpha$$

Вариант 7

$$z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$$

$$z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$$

Вариант 8

$$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$$

$$z_2 = \sin(y + x)\cdot\sin(y - x)$$

Вариант 9

$$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$$

$$z_2 = -4\sin^2\frac{\alpha - \beta}{2}\cdot\cos(\alpha + \beta)$$

Вариант 10

$$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$$

$$z_2 = \operatorname{ctg}\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$$

Вариант 11

$$z_1 = \frac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$$

$$z_2 = \frac{1 - \operatorname{tg}\alpha}{1 + \operatorname{tg}\alpha}$$

Вариант 12

$$z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha}$$

$$z_2 = \operatorname{ctg}\left(\frac{3}{2}\pi - \alpha\right)$$

Вариант 13

$$z_1 = \frac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$$

$$z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$$

Вариант 14

$$z_1 = \frac{\cos\alpha + \sin\alpha}{\cos\alpha - \sin\alpha}$$

$$z_2 = \operatorname{tg} 2\alpha + \sec 2\alpha$$

Вариант 15

$$z_1 = \frac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2}$$

$$z_2 = \frac{1}{\sqrt{b+2}}$$

Вариант 16

$$z_1 = \frac{x^2 + 2x - 3 + (x+1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x-1)\sqrt{x^2 - 9}}$$

$$z_2 = \sqrt{\frac{x+3}{x-3}}$$

Вариант 17

$$z_1 = \frac{\sqrt{(3m+2)^2 - 24m}}{3\sqrt{m} - \frac{2}{\sqrt{m}}}$$

$$z_2 = -\sqrt{m}$$

Вариант 18

$$z_1 = \left(\frac{a+2}{\sqrt{2a}} - \frac{a}{\sqrt{2a}+2} + \frac{2}{a - \sqrt{2a}} \right) \cdot \frac{\sqrt{a} - \sqrt{2}}{a+2}$$

$$z_2 = \frac{1}{\sqrt{a} + \sqrt{2}}$$

Вариант 19

$$z_1 = \left(\frac{1 + a + a^2}{2a + a^2} + 2 - \frac{1 - a + a^2}{2a - a^2} \right)^{-1} (5 - 2a^2)$$

$$z_2 = \frac{4 - a^2}{2}$$

Вариант 20

$$z_1 = \frac{(m-1)\sqrt{m} - (n-1)\sqrt{n}}{\sqrt{m^3 n} + nm + m^2 - m}$$

$$z_2 = \frac{\sqrt{m} - \sqrt{n}}{m}$$

Контрольные вопросы

1. Отличие языка программирования от среды программирования.
2. Операторы ввода-вывода в C#.
3. Основные типы данных языка C#.
4. Способы объявления переменных в языке C#.
5. Вычисление математических функций в C#.
6. Операторы языка C# и их приоритеты.

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задания в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: Разветвляющиеся алгоритмы. Циклы.

Цель: Изучение основных операторов структурного программирования в языке C# и способов их применения.

Теоретические сведения

Ветвлением называется процесс управления тем, какая строка кода должна выполняться следующей. За то, на какую строку должен осуществляться переход, отвечает определенный вид условного оператора. Действие этого условного оператора основано на сравнении проверочного значения с одним или более возможными значениями с применением булевской логики.

```
if (<условие>) <код, выполняемый, если <условие> равно <true>;
```

После выполнения этого кода, или невыполнения из-за того, что в результате вычисления выражения <условие> было получено false, выполнение программы снова возобновляется со следующей строки кода.

Вместе с оператором if также может указываться и дополнительный код с помощью оператора else. Этот оператор выполняется в случае, если при вычислении выражения <условие> получается false:

```
if (<условие>) <код, выполняемый, если <условие> равно true>;
else <код, выполняемый, если <условие> равно false>;
```

Оба раздела кода могут занимать несколько строк и представлять собой заключенные в фигурные скобки блоки:

```
if (<условие>)
{
код, выполняемый, если <условие> равно true;
}
else
{
код, выполняемый, если <условие> равно false;
}
```

Для проверки нескольких условий:

```
if (<условие>)
{
код, выполняемый, если <условие> равно true;
}
else if (<условие2>)
{
код, выполняемый, если <условие2> равно true;
}

else
{
код, выполняемый, если ни одно из условий не равно true;
}
```

Организация циклов подразумевает повторяющееся выполнение операторов. Эта методика является очень полезной, поскольку позволяет делать так, чтобы необходимые операции выполнялись повторно столько, сколько требуется раз, без повторного написания одного и того же кода снова и снова.

Для описания цикла требуется следующая информация:

—начальное значение для инициализации переменной, играющей роль счетчика;

—условие для продолжения цикла, в котором участвует переменная-счетчик;

—операция, которая должна выполняться над переменной-счетчиком в конце каждого цикла.

```
for {<инициализация>; <условие>; <операция>)
{
код, подлежащий выполнению в цикле;
}
```

Например, приведённый ниже фрагмент кода выведет на экран числа от 1 до 10.

```
for (int i = 1; i <= 10; ++i)
{
Console.WriteLine(i);
}
```

Задание

Написать программу, вычисляющую и выводящую на экран таблицу значений функции F в диапазоне от $x_{нач}$ до $x_{кон}$ с шагом dx. Значения a, b, c задаются константой в коде программы, значения $x_{нач}$, $x_{кон}$ и dx вводятся с клавиатуры.

Вариант 1

$$F = \begin{cases} ax^2 + b & при\ x < 0\ и\ b \neq 0 \\ \dfrac{x-a}{x-c} & при\ x > 0\ и\ b = 0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 2

$$F = \begin{cases} \dfrac{1}{ax} - b & при\ x+5 < 0\ и\ c = 0 \\ \dfrac{x-a}{x} & при\ x+5 > 0\ и\ c \neq 0 \\ \dfrac{10x}{c-4} & в\ остальных\ случаях \end{cases}$$

Вариант 3

$$F = \begin{cases} ax^2 + bx + c & при\ a < 0\ и\ c \neq 0 \\ \dfrac{-a}{x-c} & при\ a > 0\ и\ c = 0 \\ a(x+c) & в\ остальных\ случаях \end{cases}$$

Вариант 4

$$F = \begin{cases} -ax - c & при\ c < 0\ и\ x \neq 0 \\ \dfrac{x-a}{-c} & при\ c > 0\ и\ x = 0 \\ \dfrac{bx}{c-a} & в\ остальных\ случаях \end{cases}$$

Вариант 5

$$F = \begin{cases} a - \dfrac{x}{10+x} & при\ x < 0\ и\ b \neq 0 \\ \dfrac{x-a}{x-c} & при\ x > 0\ и\ b = 0 \\ 3x + \dfrac{2}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 6

$$F = \begin{cases} ax^2 + b^2 x & при\ c < 0\ и\ b \neq 0 \\ \dfrac{x+a}{x+c} & при\ c > 0\ и\ b = 0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 7

$$F = \begin{cases} -ax^2 - b & при\ x < 5\ и\ c \neq 0 \\ \dfrac{x-a}{x} & при\ x > 5\ и\ c = 0 \\ \dfrac{-x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 8

$$F = \begin{cases} -ax^2 & при\ c < 0\ и\ a \neq 0 \\ \dfrac{a-x}{cx} & при\ c > 0\ и\ a = 0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 9

$$F = \begin{cases} ax^2 + b^2 x & при\ a < 0\ и\ x \neq 0 \\ x - \dfrac{a}{x-c} & при\ a > 0\ и\ x = 0 \\ 1 + \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 10

$$F = \begin{cases} ax^2 - bx + c & при\ x < 3\ и\ b \neq 0 \\ \dfrac{x-a}{x-c} & при\ x > 3\ и\ b = 0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 11

$$F = \begin{cases} ax^2 + \dfrac{b}{c} & при\ x < 1\ и\ c \neq 0 \\ \dfrac{x-a}{(x-c)^2} & при\ x > 1{,}5\ и\ c = 0 \\ \dfrac{x^2}{c^2} & в\ остальных\ случаях \end{cases}$$

Вариант 12

$$F = \begin{cases} ax^3 + b^2 + c & \text{при } x < 0{,}6 \text{ и } b + c \neq 0 \\ \dfrac{x-a}{x-c} & \text{при } x > 0{,}6 \text{ и } b + c = 0 \\ \dfrac{x}{c} + \dfrac{x}{a} & \text{в остальных случаях} \end{cases}$$

Вариант 13

$$F = \begin{cases} ax^2 + b & \text{при } x - 1 < 0 \text{ и } b - c \neq 0 \\ \dfrac{x-a}{x} & \text{при } x - 1 > 0 \text{ и } b + x = 0 \\ \dfrac{x}{c} & \text{в остальных случаях} \end{cases}$$

Вариант 14

$$F = \begin{cases} -ax^3 - b & \text{при } x + c < 0 \text{ и } a \neq 0 \\ \dfrac{x-a}{x-c} & \text{при } x + c > 0 \text{ и } a = 0 \\ \dfrac{x}{c} + \dfrac{c}{x} & \text{в остальных случаях} \end{cases}$$

Вариант 15

$$F = \begin{cases} -ax^2 + b & \text{при } x < 0 \text{ и } b \neq 0 \\ \dfrac{x}{x-c} + 5{,}5 & \text{при } x > 0 \text{ и } b = 0 \\ \dfrac{x}{-c} & \text{в остальных случаях} \end{cases}$$

Вариант 16

$$F = \begin{cases} a(x+c)^2 - b & при\ x = 0\ и\ b \neq 0 \\ \dfrac{x-a}{-c} & при\ x = 0\ и\ b = 0 \\ a + \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 17

$$F = \begin{cases} ax^2 - cx + b & при\ x+10 < 0\ и\ b \neq 0 \\ \dfrac{x-a}{x-c} & при\ x+10 > 0\ и\ b = 0 \\ \dfrac{-x}{a-c} & в\ остальных\ случаях \end{cases}$$

Вариант 18

$$F = \begin{cases} ax^3 + bx^2 & при\ x < 0\ и\ b \neq 0 \\ \dfrac{x-a}{x-c} & при\ x > 0\ и\ b = 0 \\ \dfrac{x+5}{c(x-10)} & в\ остальных\ случаях \end{cases}$$

Вариант 19

$$F = \begin{cases} a(x+7)^2 - b & при\ x < 5\ и\ b \neq 0 \\ \dfrac{x-cd}{ax} & при\ x > 5\ и\ b = 0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$$

Вариант 20

$$F = \begin{cases} -\dfrac{2x-c}{cx-a} & при\ x<0\ и\ b\neq 0 \\ \dfrac{x-a}{x-c} & при\ x>0\ и\ b=0 \\ -\dfrac{x}{c}+\dfrac{-c}{2x} & в\ остальных\ случаях \end{cases}$$

Контрольные вопросы

1. Операторы ветвления в языке C#.
2. Операторы множественного выбора.
3. Операторы цикла в C#.
4. Циклы с предусловием и постусловием.
5. Цикл с параметром.

Требования к отчёту

Отчёт по лабораторной работе должен содержать:
— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задания в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: Одномерные массивы.

Цель: Изучить способы описания одномерных массивов в языке программирования C# и методы работы с ними.

Теоретические сведения

Массив — набор однотипных элементов, с общим для обращения к ним именем, доступ к которым осуществляется по индексу (номеру).

Для объявления массива используется следующая форма:

```
тип[] имя_массива = new тип[размер];
```

Например, массив из пяти элементов типа double может быть объявлен одним из следующих способов:

```
double[] mas = new double[5];
double[] mas = {2,4,6,8,0};
double[] mas = new double[5] {2,4,6,8,0};
double[] mas = new double[] {2,4,6,8,0};
```

Количество элементов, которые могут содержаться в массиве, называется размерностью массива.

Доступ к элементам массива осуществляется по имени и индексу. Например, строка

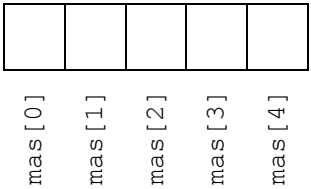
```
mas[2] = 3;
```

означает, что в элемент массива с номером 2 будет записано число 3, а строка

```
Console.WriteLine(mas[4]);
```

выведет на экран значение четвёртого элемента массива.

При этом следует помнить, что нумерация элементов массива начинается с 0, а последний элемент имеет номер на 1 меньший, чем размерность массива.



Для работы с массивами удобно использовать циклы, так как они позволяют многократно проводить однотипные операции. Например, ввод элементов массива можно реализовать в цикле

следующим образом:

```
int n = 5;
double[] mas = new double[5];
for(int i=0; i<n; i++)
{
    Console.Write("Введите " + i + "-ый элемент
массива: ");
    mas[i] = double.Parse(Console.ReadLine());
}
```

Вывод массива соответственно будет выглядеть следующим образом:

```
for (int i = 0; i < n; i++)
{
    Console.Write(mas[i] + " ");
}
```

Задание

Написать программу, позволяющую ввести одномерный массив из *n* вещественных чисел и выполняющую действия над элементами массива в соответствии с вариантом задания.

Вариант 1

a) Вычислить сумму отрицательных элементов массива.

b) Упорядочить элементы массива по возрастанию.

Вариант 2

a) Вычислить сумму положительных элементов массива.

b) Упорядочить элементы массива по убыванию.

Вариант 3

a) Вычислить произведение элементов массива с четными номерами.

b) Преобразовать массив таким образом, чтобы сначала располагались элементы большие 0, а потом — все остальные.

Вариант 4

a) Вычислить сумму элементов массива с нечетными номерами.

b) Сжать массив, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 5

a) Найти максимальный элемент массива.

b) Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 6

a) Найти минимальный элемент массива.

b) Поменять порядок следования элементов массива на обратный.

Вариант 7

a) Определить номер максимального элемента массива.

b) Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине — элементы, стоявшие в четных позициях.

Вариант 8

a) Определить номер минимального элемента массива.

b) Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом — все остальные.

Вариант 9

a) Найти максимальный по модулю элемент массива.

b) Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

Вариант 10

a) Найти минимальный по модулю элемент массива.

b) Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине — элементы, стоявшие в нечетных позициях.

Вариант 11

a) Определить номер минимального по модулю элемента массива.

b) Сжать массив, удалив из него все элементы, величина которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.

Вариант 12

a) Определить номер максимального по модулю элемента массива.

b) Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a, b], а потом — все остальные.

Вариант 13

a) Посчитать количество элементов массива, лежащих в диапазоне от A до B.

b) Упорядочить элементы массива по убыванию модулей элементов.

Вариант 14

a) Посчитать количество элементов массива, равных 0.

b) Упорядочить элементы массива по возрастанию модулей элементов.

Вариант 15

a) Посчитать количество элементов массива, больших C.

b) Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом — все положительные (элементы, равные 0, считать положительными).

Вариант 16

a) Посчитать количество отрицательных элементов массива.

b) Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.

Вариант 17

a) Посчитать количество положительных элементов массива.

b) Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом — все остальные,

Вариант 18

a) Посчитать количество элементов массива, меньших C.

b) Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом — все остальные.

Вариант 19

a) Вычислить произведение отрицательных элементов массива.

b) Изменить порядок следования элементов в массиве на обратный.

Вариант 20

a) Вычислить произведение положительных элементов массива.

b) Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

Контрольные вопросы

1. Что такое массив?
2. Объявление массивов в C#.
3. Что такое размерность массива?
4. Как осуществляется обращение к элементам массива?
5. Как производится нумерация элементов массива?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:
—титульный лист,
—цель работы,
—краткие теоретические сведения,
—задания в соответствии с вариантом,
—листинг программы,
—пример работы программы в виде скриншотов,
—вывод.

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: Двумерные массивы.

Цель: Изучить способы задания двумерных массивов и методы работы с ними.

Теоретические сведения

В программировании, на ряду с одномерными, часто используются многомерные массивы. Многомерным называется такой массив, который отличается двумя или более измерениями, причем доступ к каждому элементу такого массива осуществляется с помощью определенной комбинации двух или более индексов.

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно представить в виде таблицы, на строки которой указывает один индекс, а на столбцы — другой.

В следующей строке кода объявляется двумерный массив целых чисел размерами 4×5.

```
int[,] table = new int[4, 5];
```

Первым при этом указывается число строк, вторым — число столбцов.

Возможны и другие варианты объявления двумерного массива, включающие задание значений его элементов:

```
int[,] table = {
                {1,2,3},
                {4,5,6},
                {7,8,9}
                };
```

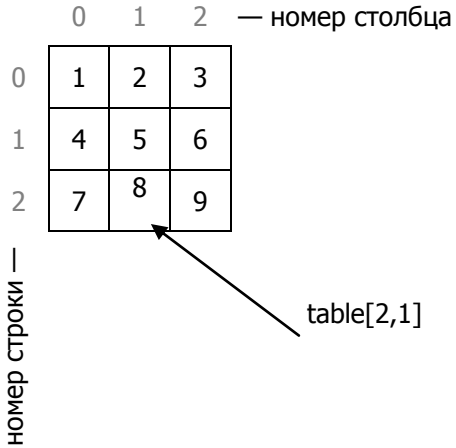
или

```
int[,] table = new int[3,3] {
                {1,2,3},
                {4,5,6},
                {7,8,9}
                };
```

или

```
int[,] table = new int[,] {
                    {1,2,3},
                    {4,5,6},
                    {7,8,9}
                    };
```

В результате выполнения описанных выше команд будет создан массив следующего вида:



Для доступа к элементу двумерного массива следует указать оба индекса, разделив их запятой. Например, в следующей строке кода элементу массива table с координатами местоположения (2,1) присваивается значение 10.

```
table[2, 1] = 10;
```

Для ввода-вывода двумерных массивов, по аналогии с одномерными, можно использовать циклы. Так как двумерный массив имеет два измерения, то для доступа к каждому его элементу используются два цикла — счётчик первого цикла определяет номер строки, счётчик второго цикла, вложенного в первый, определяет номер столбца (или номер ячейки в строке).

```
// ввод
```

```
int n = 3;
int m = 4;
double[,] matrix = new double[n, m];

for (int i = 0; i < n; i++)
{
  for (int j = 0; j < m; j++)
   {
    Console.Write("Введите элемент [" + i + "," + j
+ "] ");
   matrix[i, j] = double.Parse(Console.ReadLine());
    }
}


// вывод
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < m; j++)
   {
      Console.Write(matrix[i,j]);
    }
   Console.WriteLine();
}
```

В C# можно также создавать специальный тип двумерного массива, называемый ступенчатым массивом. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной. Следовательно, ступенчатый массив может быть использован для составления таблицы из строк разной длины.

Объявляется ступенчатый массив следующим образом:

```
тип[][] имя_массива = new тип [размер][];
```

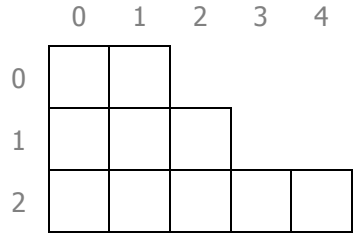
где размер обозначает число строк в массиве.

Задание количества ячеек в каждой строке производится отдельно. Например:

```
int[][] massiv = new int[3][];
```

```
massiv[0] = new int[2];
massiv[1] = new int[3];
massiv[2] = new int[5];
```

Данный программный код создаст массив следующего вида:



После создания ступенчатого массива доступ к его элементам осуществляется по индексу, указываемому в отдельных квадратных скобках. Например, занести число 100 в ячейку ступенчатого массива можно следующим образом:

```
massiv[1][2] = 100;
```

С каждым массивом связано свойство Length, содержащее число элементов, из которых может состоять массив и позволяющее определить его длину.

Задание

Написать программу для выполнения над двумерным массивом действий, указанных в варианте задания.

Вариант 1

Определить количество строк, не содержащих ни одного нулевого элемента.

Создать ступенчатый массив, скопировав в него только элементы из начального массива, лежащие в диапазоне [a, b].

Вариант 2

Определить количество столбцов, не содержащих ни одного нулевого элемента.

Создать ступенчатый массив, скопировав в него только положительные элементы из начального массива.

Вариант 3

Определить количество столбцов, содержащих хотя бы один нулевой элемент.

Создать ступенчатый массив, скопировав в него только элементы, лежащие выше главной диагонали матрицы.

Вариант 4

Определить произведение элементов в тех строках, которые не содержат отрицательных элементов.

Создать ступенчатый массив, скопировав в него только отрицательные элементы из начального массива.

Вариант 5

Определить сумму элементов в тех столбцах, которые не содержат отрицательных элементов.

Создать ступенчатый массив, скопировав в него элементы начального массива, являющиеся целыми степенями числа 2.

Вариант 6

Определить сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Создать ступенчатый массив, скопировав в него только элементы, лежащие ниже главной диагонали матрицы.

Вариант 7

Найти такие *k*, что *k*-я строка матрицы совпадает с *k*-м столбцом.

Создать ступенчатый массив, скопировав в него только элементы из начального массива, не лежащие в диапазоне [a, b].

Вариант 8

Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Создать ступенчатый массив, скопировав в него только элементы, лежащие выше побочной диагонали матрицы.

Вариант 9

Найти сумму элементов, расположенных выше главной диагонали.

Создать ступенчатый массив, скопировав в него только чётные элементы начального массива.

Вариант 10

Подсчитать количество локальных минимумов в массиве. Элемент называется локальным минимумом, если он строго меньше всех имеющихся у него соседей.

Создать ступенчатый массив, скопировав в него только элементы, лежащие ниже побочной диагонали матрицы.

Вариант 11

Определить максимум среди сумм элементов диагоналей, параллельных побочной диагонали матрицы.

Создать ступенчатый массив, скопировав в него элементы из начального массива, делящиеся без остатка на 3.

Вариант 12

Найти номер первой из строк, содержащих хотя бы один положительный элемент.

Создать ступенчатый массив, скопировав в него только те элементы каждой строки начального массива, которые стоят до максимального элемента строки.

Вариант 13

Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.

Создать ступенчатый массив, скопировав в него только те элементы каждой строки начального массива, которые стоят после минимального элемента строки.

Вариант 14

Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.

Создать ступенчатый массив, скопировав в него только те элементы из начального массива, которые отличаются от среднего арифметического всех элементов массива не более чем на 30%.

Вариант 15

Определить минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Создать ступенчатый массив, скопировав в него только нечётные элементы начального массива.

Вариант 16

Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.

Создать ступенчатый массив, скопировав в него только те элементы из начального массива, которые отличаются от максимального элемента не более чем на 20%.

Вариант 17

Найти номер первой из строк, не содержащих ни одного

положительного элемента.

Создать ступенчатый массив, скопировав в него элементы из начального массива, которые больше стоящих выше (в предыдущей строке) элементов. Самую верхнюю строку скопировать полностью.

Вариант 18

Определить номер столбца, в котором находится самая длинная серия одинаковых элементов.

Создать ступенчатый массив, скопировав в него элементы из начального массива, которые меньше элементов, стоящего сразу после них.

Вариант 19

Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду.

Создать ступенчатый массив, скопировав в него только модули отрицательных элементов из начального массива.

Вариант 20

Определить количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент.

Создать ступенчатый массив, скопировав в него начальный массив. Если в строке какое-либо значение встречается более одного раза, то скопировать его необходимо только один раз, удалив таким образом повторяющиеся элементы.

Контрольные вопросы

1. Что такое двумерный массив?
2. Как происходит обращение к элементу двумерного массива?
3. Какие параметры необходимо указать при объявлении двумерного массива?
4. Как осуществить ввод-вывод двумерного массива?
5. Что такое ступенчатый массива?
6. Как осуществляется доступ к элементам ступенчатого массива?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задания в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Классы.

Цель: Изучить основы объектно-ориентированного программирования на языке C#, получить практические навыки описания классов и их использования.

Теоретические сведения

Объектно-ориентированное (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В спецификация класса C# используется для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.

В общем виде объявление класса на языке C# выглядит следующим образом:

```
class имя_класса {
// Объявление переменных.
доступ тип переменная1;
доступ тип переменная2;
//...
доступ тип переменнаяN;
// Объявление методов.
доступ возвращаемый_тип метод1 (параметры) {
// тело метода
}
доступ возвращаемый_тип метод2 (параметры). {
// тело метода
}
//...
доступ возвращаемый_тип методы (параметры) {
// тело метода
}
}
```

Доступ определяется одним из следующих ключевых слов:

— `private` — поле или метод могут быть использован только внутри тела класса в котором объявлены;

— `public` — поле или метод могут быть использован из любого места в программе.

Для создания объекты из класса используется оператор new:

```
Имя_класса название_объекта = new Имя_класса();
```

Например для класса `Student`:

```
Student s1 = new Student();
```

Эта строка объявления выполняет три функции. Во-первых, объявляется переменная s1, относящаяся к классу Student. Сама эта переменная не является объектом, а лишь переменной, которая может ссылаться на объект. Во-вторых, создается конкретная, физическая, копия объекта. Это делается с помощью оператора new. И наконец, переменной s1 присваивается ссылка на данный объект. Таким образом, после выполнения анализируемой строки объявленная переменная s1 ссылается на объект типа Student.

Класс представляет собой объявленный программистом составной тип данных, имеющий в составе:

—Поля данных — параметры объекта, задающие его состояние (свойства объекта предметной области). Иногда поля данных объекта называют свойствами объекта, из-за чего возможна путаница. Физически поля представляют собой значения (переменные, константы), объявленные как принадлежащие классу.

—Методы — процедуры и функции, связанные с классом. Они определяют действия, которые можно выполнять над объектом такого типа, и которые сам объект может выполнять.

Методы представляют собой подпрограммы, которые манипулируют данными, определенными в классе, а во многих случаях они предоставляют доступ к этим данным. Как правило, другие части программы взаимодействуют с классом посредством его методов.

Метод состоит из одного или нескольких операторов. В грамотно написанном коде C# каждый метод выполняет только одну функцию. У каждого метода имеется свое имя, по которому он вызывается. В общем случае, методу в качестве имени можно присвоить любой действительный идентификатор. Следует, одна-

ко, иметь в виду, что идентификатор `Main()` зарезервирован для метода, с которого начинается выполнение программы. Кроме того, в качестве имен методов нельзя использовать ключевые слова C#.

Общая форма определения метода:

```
доступ тип имя(тип имя_параметра, тип
имя_параметра, …)
{
// тело метода
}
```

Для возврата значения из метода в вызывающую часть программы служит следующая форма оператора return:

```
return значение;
```

где значение — это конкретное возвращаемое значение.

Если метод не возвращает никакого значения, то указывается тип `void`.

Задание

1. Разработать класс на языке C# в соответствии с вариантом задания.
2. Написать программу, демонстрирующую работу указанных выше элементов класса. В программе должен быть создан массив объектов разработанного класса, реализован ввод и вывод объектов, содержащихся в массиве.

Вариант 1

Разработать класс Worker (сотрудник), содержащий следующие поля:

personID (табельный номер сотрудника),
family (фамилия сотрудника),
name (имя сотрудника),
dolgnost (должность),
date (дата приёма на работу);
методы:
CalculateSalary() — посчитать зарплату (зарплата зависит от должности).

Вариант 2

Разработать класс Worker (сотрудник), содержащий следующие поля:

personID (табельный номер сотрудника),
family (фамилия сотрудника),

name (имя сотрудника),
dolgnost (должность),
date (дата приёма на работу);
методы:
CalculateSalary() — посчитать зарплату (зарплата зависит от должности).

Вариант 3

Разработать класс Jobless (безработный), содержащий следующие поля:

joblessID (регистрационный номер безработного),
lastName (фамилия безработного),
firstName (имя безработного),
birth (год рождения);
методы:
GetAge() — получить возраст.

Вариант 4

Разработать класс Jobless (безработный), содержащий следующие поля:

joblessID (регистрационный номер безработного),
lastName (фамилия безработного),
firstName (имя безработного),
birth (год рождения);
методы:
GetAge() — получить возраст.

Вариант 5

Разработать класс Passport (паспорт), содержащий следующие поля:

serial (серия документа),
number (номер документа),
date (дата выдачи),
who (кем выдан);
методы:
GetChangeTime() — получить время следующего обмена.

Вариант 6

Разработать класс Passport (паспорт), содержащий следующие поля:

serial (серия документа),
number (номер документа),
date (дата выдачи),
who (кем выдан);

методы:
GetChangeTime() — получить время следующего обмена.

Вариант 7

Разработать класс Route (туристический маршрут), содержащий следующие поля:
routeID (идентификатор маршрута),
name (название маршрута),
period (время в пути),
cost (стоимость одного дня);
методы:
GetPrice() — получить цену за весь срок прохождения маршрута.

Вариант 8

Разработать класс Route (туристический маршрут), содержащий следующие поля:
routeID (идентификатор маршрута),
name (название маршрута),
period (время в пути),
cost (стоимость одного дня);
методы:
GetPrice() — получить цену за весь срок прохождения маршрута.

Вариант 9

Разработать класс Cust (арендатор), содержащий следующие поля:
INN (ИНН арендатора),
name (название арендатора),
address (адрес арендатора),
room (номер кабинета),
chief (фамилия руководителя);
методы:
GetFloor() — получить номер этажа (зависит от номера кабинета, кабинеты, номера которых начинаются на 1 находятся на 1-м этаже, номера которых начинаются на 2 находятся на 2-м этаже и т.д.).

Вариант 10

Разработать класс Cust (арендатор), содержащий следующие поля:
INN (ИНН арендатора),
name (название арендатора),
address (адрес арендатора),
room (номер кабинета),
chief (фамилия руководителя);

методы:

GetFloor() — получить номер этажа (зависит от номера кабинета, кабинеты, номера которых начинаются на 1 находятся на 1-м этаже, номера которых начинаются на 2 находятся на 2-м этаже и т.д.).

Вариант 11

Разработать класс TelephonNumber (телефонный номер), содержащий следующие поля:

id (идентификатор клиента),
family (фамилия клиента),
phoneAddress (адрес клиента),
phoneNumber (номер телефона),
tariffType (вид тарифа — безлимитный, поминутный и т.д.);
методы:
GetCost() — получить плату за телефон (зависит от вида тарифа).

Вариант 12

Разработать класс TelephonNumber (телефонный номер), содержащий следующие поля:

id (идентификатор клиента),
family (фамилия клиента),
phoneAddress (адрес клиента),
phoneNumber (номер телефона),
tariffType (вид тарифа — безлимитный, поминутный и т.д.);
методы:
GetCost() — получить плату за телефон (зависит от вида тарифа).

Вариант 13

Разработать класс Count (счет), содержащий следующие поля:

countNumber (номер счета),
date (дата выписки счета),
value (сумма к уплате),
count (количество товара);
методы:
GetAverage() — получить среднюю стоимость товара в счёте.

Вариант 14

Разработать класс Count (счет), содержащий следующие поля:

countNumber (номер счет-фактуры),
date (дата выписки счет-фактуры),
value (сумма к уплате),
count (количество товара);

методы:
GetAverage() — получить среднюю стоимость товара в счёте.

Вариант 15

Разработать класс Buy (покупка), содержащий следующие поля:
number (номер покупки),
sum (сумма покупки),
otdel (название отдела магазина, где совершена покупка),
otdel (название отдела магазина, где совершена покупка);
методы:
GetSumSoSkidkoi(double skidka) — посчитать сумму со скидкой.

Вариант 16

Разработать класс Buy (покупка), содержащий следующие поля:
number (номер покупателя),
sum (сумма покупки),
otdel (название отдела магазина, где совершена покупка);
методы:
GetSumSoSkidkoi(double skidka) — посчитать сумму со скидкой.

Вариант 17

Разработать класс Spec (специальность), содержащий следующие поля:
kod (код специальности),
name (название специальности),
balls (количество баллов, необходимых для поступления);
методы:
Check(int mathBall, int phisicsBall, int rusBall) — проверить достаточно ли баллов для поступления на данную специальность.

Вариант 18

Разработать класс Spec (специальность), содержащий следующие поля:
kod (код специальности),
name (название специальности),
balls (количество баллов, необходимых для поступления);
методы:
Check(int mathBall, int phisicsBall, int rusBall) — проверить достаточно ли баллов для поступления на данную специальность.

Вариант 19

Разработать класс Bilet (билет), содержащий следующие

поля:

time1 (время вылета),
time2 (время прилета),
cost (цена билета),
name (владелец билета),
nomer (номер места);
методы:
GetDuration() — посчитать время в пути.

Вариант 20

Разработать класс Bilet (билет), содержащий следующие поля:

time1 (время вылета),
time2 (время прилета),
cost (цена билета),
name (владелец билета),
nomer (номер места);
методы:
GetDuration() — посчитать время в пути.

Контрольные вопросы

1. Что такое класс?
2. Что такое объект?
3. Что такое поля класса?
4. Что такое методы класса?
5. Как описать класс на языке C#?
6. Какие бывают модификаторы доступа к членам класса?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задания в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема: Методы класса. Конструкторы. Операторы класса.

Цель: Изучить синтаксис языки C# для описания методов класса и конструкторов, получить практические навыки перегрузки операторов класса.

Теоретические сведения

Конструктор — метод, вызываемый при создании объекта для инициализации значений его полей. У конструктора такое же имя, как и у его класса. Тип значения у конструктора не указывается.

Перегрузка операторов позволяет описывать и применять к созданным программистом типам данных (классам) операции, по смыслу эквивалентные уже имеющимся в языке.

Как только для класса определяются операторы, появляется возможность оперировать объектами этого класса, используя обычный синтаксис выражений в C#.

Общая форма перегрузки унарного оператора:

```
public static возвращаемый_тип operator op(тип_параметра операнд)
{
// операции
}
```

Общая форма перегрузки бинарного оператора:

```
public static возвращаемый_тип operator op(тип_параметра1 операнд1, тип_параметра1 операнд2)
{
// операции
}
```

Возвращаемый_тип может быть любым типом, но обычно он соответствует типу класса, для которого перегружается оператор.

Тип_параметра для унарных операторов должен быть таким же, как и у класса, для которого перегружается оператор. А в бинарных операторах хотя бы один из операндов должен быть такого же типа, как и у его класса.

Задание

Изменить класс, разработанный в предыдущей лабораторной работе следующим образом:

1. Все поля класса, на значения которых имеются ограничения, сделать скрытыми (объявленными как private).

2. Для доступа к каждому такому полю реализовать соответствующие методы — один для присваивания значения полю, другой — для получения значения. В методе присваивания необходимо выполнять проверку введённых значений на соответствие имеющимся ограничениям.

3. Добавить в класс конструктор по умолчанию, конструктор с параметрами и конструктор копирования.

4. Реализовать операторы сравнения (> и <) для класса.

5. В программе выполнить сортировку массива объектов.

Контрольные вопросы

1. Что такое модификатор доступа? Какие бывают модификаторы доступа?
2. Что такое конструктор?
3. Как на языке C# создать конструктор для класса?
4. Какое действие выполняет оператор return?
5. Что такое тип void и в каких случаях он используется?
6. Что такое перегрузка операторов?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задание,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 7

Тема: Наследование.

Цель: Изучить способы описания наследования классов.

Теоретические сведения

Наследование — механизм объектно-ориентированного програм-мирования, позволяющий описать новый класс на основе уже существующего. Оно является одним из основополагающих принципов объектно-ориентированного программирования. Благодаря наследованию можно создать общий класс, в котором определяются характерные особенности, присущие множеству связанных элементов. От этого класса могут затем наследовать другие, более конкретные классы, добавляя в него свои индивидуальные особенности.

В языке C# класс, который наследуется, называется базовым, а класс, который наследует — производным. Следовательно, производный класс представляет собой специализированный вариант базового класса. Он наследует все поля, методы и свойства, определяемые в базовом классе, добавляя к ним свои собственные элементы.

Для любого производного класса можно указать только один базовый класс. Если `ClassC` является производным от ClassB, и ClassB является производным от `ClassA`, `ClassC` наследует члены, объявленные в `ClassB и ClassA`.

Поддержка наследования в C# состоит в том, что в объявление одного класса разрешается вводить другой класс. Для этого при объявлении производного класса через двоеточие указывается базовый класс:

```
    class имя_производного_класса :
имя_базового_класса {
    // тело класса
    }
```

Для любого производного класса можно указать только один базовый класс.

Наследование класса не отменяет ограничения, накладываемые на доступ к закрытым членам класса. Поэтому если в производный класс и входят все члены его базового класса, в нем все равно оказываются недоступными те члены базового класса, которые являются закрытыми. Для того чтобы доступ к полю или методы класса можно было получать из объектов этого класса и

из объектов классов-наследников, но не из каких-либо других объектов или классов, используется модификатор доступа protected.

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные конструкторы, при этом конструктор базового класса конструирует базовую часть объекта, а конструктор производного класса — производную часть этого объекта.

С помощью формы расширенного объявления конструктора производного класса и ключевого слова base в производном классе может быть вызван конструктор, определенный в его базовом классе. Ниже приведена общая форма этого расширенного объявления:

```
конструктор_производного_класса(параметры) : base (аргументы)
{
//тело конструктора
}
```

где `список_аргументов` обозначает любые аргументы, необходимые конструктору в базовом классе.

Виртуальный метод отличается от обычного тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода. Кроме того, виртуальные методы интересны тем, что именно происходит при их вызове по ссылке на базовый класс. В этом случае средствами языка C# определяется именно тот вариант виртуального метода, который следует вызывать, исходя из типа объекта, к которому происходит обращение по ссылке, причем это делается во время выполнения программы, а не во время компиляции. Поэтому при ссылке на разные типы объектов выполняются разные варианты виртуального метода.

Метод объявляется как виртуальный в базовом классе с помощью ключевого слова virtual, указываемого перед его именем. Когда же виртуальный метод переопределяется в производном классе, то для этого используется модификатор override. А сам процесс повторного определения виртуального метода в производном классе называется переопределением метода.

Иногда требуется создать базовый класс, в котором определяется лишь самая общая форма для всех его производных классов, а наполнение ее деталями предоставляется каждому из этих классов. В таком классе определяется лишь характер мето-

дов, которые должны быть конкретно реализованы в производных классах, а не в самом базовом классе. Подобная ситуация возникает, например, в связи с невозможностью получить содержательную реализацию метода в базовом классе.

Абстрактный метод создается с помощью указываемого модификатора типа abstract. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования. Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора virtual.

Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как абстрактный, и для этого перед его объявлением class указывается модификатор abstract. А поскольку реализация абстрактного класса не определяется полностью, то у него не может быть объектов. Следовательно, попытка создать объект абстрактного класса с помощью оператора new приведет к ошибке во время компиляции.

Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как abstract. Таким образом, атрибут abstract наследуется до тех пор, пока не будет достигнута полная реализация класса.

Задание

Вариант 1

Разработать класс Docent, являющийся наследником класса Worker, добавив следующие поля:

godZ (год присвоения звания),
uchStep (ученая степень).

При создании объекта класса Docent в соответствующее поле базового класса должна автоматически заноситься должность «доцент», которую невозможно изменить в дальнейшем.

Вариант 2

Разработать класс Voditel, являющийся наследником класса Worker (сотрудник), добавив следующие поля:

prava (номер водительских прав),
category (категория).

При создании объекта класса Voditel в соответствующее поле базового класса должна автоматически заноситься должность «водитель», которую невозможно изменить в дальнейшем.

Вариант 3

Разработать класс HighJobless, являющийся наследником класса Jobless, добавив следующие поля:

institut (оконченный институт),
date (дата постановки на учёт),
godEnd (год окончания).

Добавить в класс метод EverWork() для определения, работал ли человек вообще — если дата постановки на учёт совпадает с датой окончания института, можно считать, что он не работал никогда.

Вариант 4

Разработать класс LongJobless, являющийся наследником класса Jobless (безработный), добавив следующие поля:

date (дата постановки на учёт),
reason (причина отсутствия работы),
lastPlace (последнее место работы).

Добавить в класс методы:

GetDuration(), определяющий как долго человек является безработным.

GetJoblessPercent(), определяющий какую часть жизни человек является безработным.

Вариант 5

Разработать класс ZarganPassport, являющийся наследником класса Passport, добавив следующие поля:

country (страна, выдавшая паспорт),
lastBorderCrossDate (дата последнего пересечения границы),
lastBorderCrossState (государство, чья граница была пересечена).

Добавить в класс метод Cross(string state), в котором реализовать автоматическую установку текущей даты в поле lastBorderCrossDate и запись в поле lastBorderCrossState параметра state.

Вариант 6

Разработать класс OldPassport, являющийся наследником класса Passport, добавив следующие поля:

dateChange (дата обмена),
reason (причина обмена).

Причиной обмена могут быть только истечение срока действия, утеря и изменение личных данных.

Вариант 7

Разработать класс ZagranRoute, являющийся наследником класса Route, добавив следующие поля:

country (страна),

needVisa (необходимость получения визы).

Необходимость получения визы зависит от страны — для поездки в некоторые страны получать визу не обязательно. Это поле должно заполняться автоматически только в результате указания страны, и ни в каком другом случае.

Вариант 8

Разработать класс HardRoute, являющийся наследником класса Route, добавив следующие поля:

skills (необходимые навыки),

place (местность, по которой проходит маршрут).

Необходимые навыки зависят от местности, по которой проходит маршрут и должны устанавливаться автоматически только при указании местности.

Вариант 9

Разработать класс PhisCust, являющийся наследником класса Cust, добавив следующие поля:

passport (номер паспорта),

snils (страховой номер индивидуального лицевого счёта).

Для физического лица поля название и фамилия руководителя должны всегда совпадать.

Вариант 10

Разработать класс UrCust, являющийся наследником класса Cust, добавив следующие поля:

OGRN (основной государственный регистрационный номер),

type (тип юридического лица),

fullAddress (полный адрес) — должен включать в себя адрес арендуемого помещения и номер кабинета и формироваться автоматически.

Вариант 11

Разработать класс TelephonWitnInternet, являющийся наследником класса TelephonNumber, добавив следующие поля:

tarifName (название тарифа),

tarifCost (стоимость тарифа),

speed (ограничение скорости интернета).

Метод GetCost() вызванный для объекта класса Telephon-

WitnInternet должен при расчёте платы за телефон учитывать также плату за интернет.

Вариант 12

Разработать класс MobileNumber, являющийся наследником класса TelephonNumber, добавив следующие поля:

operator (оператор связи),
region (домашний регион),
balans (текущий баланс счёта).

Добавить в класс метод Call(int minutes) отвечающий за снятие со счёта денег за звонок, длительность которого задаётся параметром minutes. Снятие должно осуществляться только в случае, если в поле базового класса tariffType указано «поминутный».

Вариант 13

Разработать класс PayedCount, являющийся наследником класса Count, добавив следующие поля:

datePayed (дата оплаты),
who (кто оплатил).

Добавить в базовый класс Count метод Pay(string who) реализующий оплату счёта и возвращающий в качестве результата объект класса PayedCount. Дата оплаты должны устанавливаться автоматически на основе текущей даты.

Вариант 14

Разработать класс UnpayedCount, являющийся наследником класса Count, добавив следующие поля:

srok (срок оплаты),
days (количество просроченных дней).

При получении значения поля сумма для неоплаченного счёта оно должно быть увеличено на некоторую сумму, составляющую пеню за просрочку и зависящую от количества дней просрочки.

Вариант 15

Разработать класс CardBuy, являющийся наследником класса Buy (покупка), добавив следующие поля:

creditNumber (номер кредитной карты),
creditCode (код кредитной карты).

Скидки не должны распространяться на покупки, совершенные по кредитной карте, соответственно метод базового класса GetSumSoSkidkoi, если он вызван из класса CardBuy, должен

выдавать соответствующее сообщение и возвращать исходное значение суммы, независимо от того, какая величина скидки была указана в качестве параметра.

Вариант 16

Разработать класс InternetBuy, являющийся наследником класса Buy (покупка), добавив следующие поля:

paymentSystem (платёжная система),
commission (величина комиссии).

При получении поля sum (сумма) базового класса необходимо прибавлять к ней величину комиссии.

Вариант 17

Разработать класс BakalSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

srok (срок обучения),
magistr (возможность дальнейшего поступления в магистратуру).

Разработать класс MagSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

srok (срок обучения).

Вариант 18

Разработать класс SpecSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

qualification (получаемая квалификация),
yearClose (год окончания приёма на данную специальность).

Если год окончания приёма больше текущего года, то при попытке проверить достаточно ли баллов для поступления на данную специальность должно быть выдано сообщение о том, что на данную специальность больше нельзя поступить.

Вариант 19

Разработать класс LocalBilet, являющийся наследником класса Bilet (билет), добавив следующие поля:

town1 (пункт отправления),
town2 (пункт назначения).

Добавить в класс метод ReturnTicket(int nomer) — обратный билет, для которого необходимо указать только номер места и поменять местами пункт отправления и пункт назначения.

Вариант 20

Разработать класс ForeignBilet, являющийся наследником класса Bilet (билет), добавив следующие поля:

country (страна назначения),
airport (название аэропорта).

Контрольные вопросы

1. Что такое наследование?
2. Как на языке C# при описание класса указать, что он является наследником от другого класса?
3. Как работает модификатор доступа protected?
4. Что такое виртуальный метод?
5. Что такое абстрактный класс?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задание,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 8

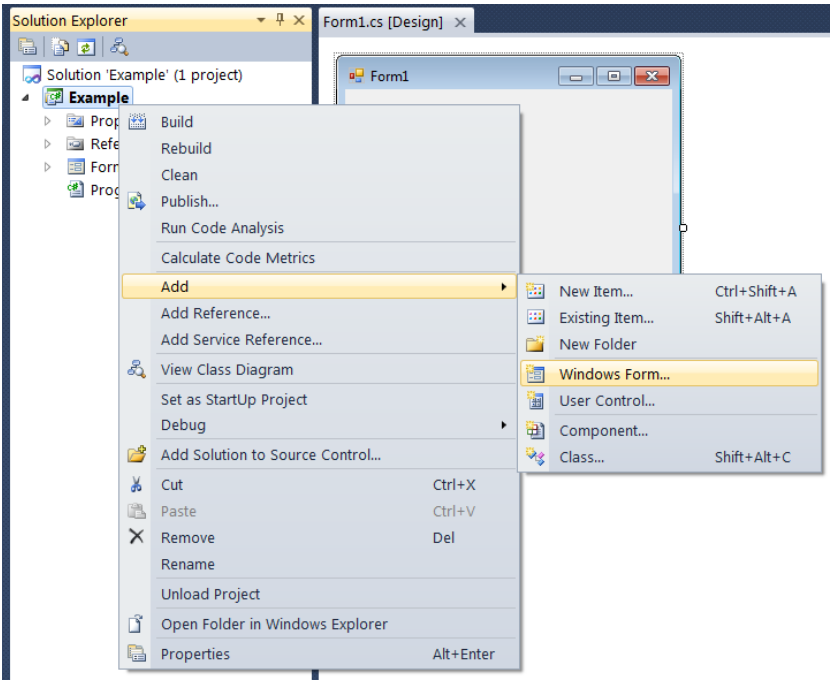
Тема: Библиотека Windows Forms.

Цель: Изучить основные классы библиотеки Windows Forms, их свойства, методы и события; изучить способы передачи данных между формами.

Теоретические сведения

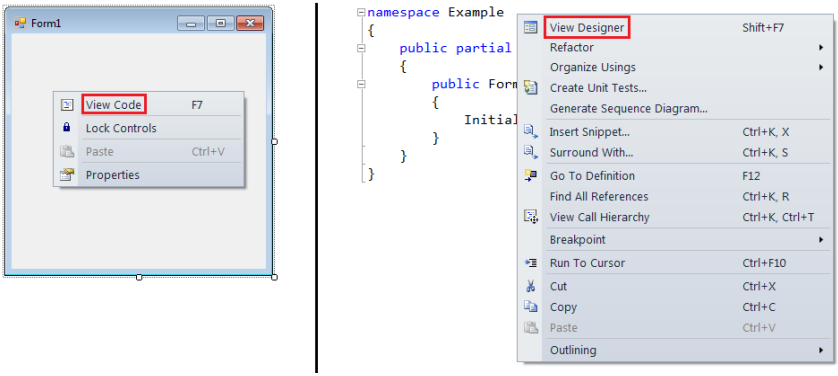
Windows Forms — название интерфейса программирования приложений (API), отвечающего за графический интерфейс пользователя и являющегося частью Microsoft .NET Framework.

Важнейшим элементом WinForms является форма, которая представляет собой стандартное окно, используемое большинством программ. Для добавления формы в проект нужно в окне обозревателя решения (Solution Explorer) нажать правой кнопкой мыши на названии проекта, в контекстном меню выбрать пункт «Add» и в появившемся меню пункт «Windows Forms...»



Форма имеет графическое представление, а также содержит в себе программный код. Для переключения между двумя

данными составляющими форму можно использовать пункты контекстного меню «View Code»/«View Designer», представленные на рисунке, или кнопку F7.

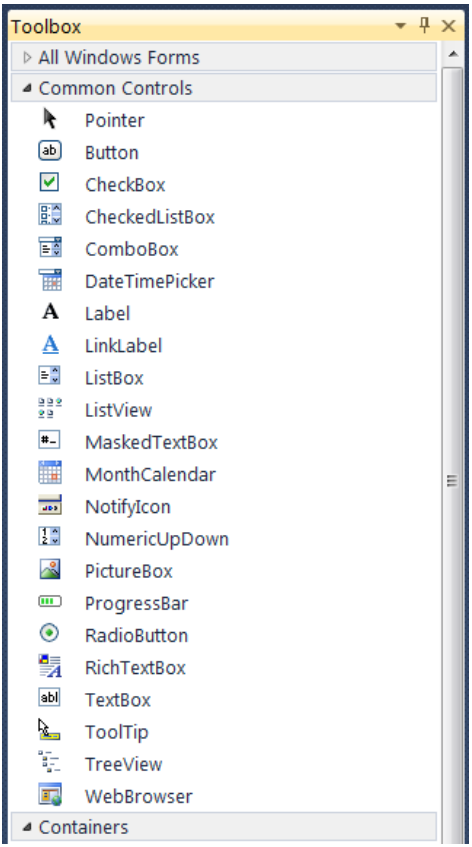


Как и всё в C# форма является классом. При её создании автоматически генерируется описание класса формы, например:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
}
```

Наличие в приведённом выше фрагменте кода выражения Form1 : Form говорит о том, что созданный класс Form1 является наследником от встроенного класса Form, т.е. новая форма будет иметь все поля и свойства, характерные для формы вообще, при этом будет иметься возможность добавления к ней новых полей, свойств и методов.

Библиотека Windows Form включает в себя большое число стандартных элементов графического интерфейса, таким как поля для ввода, выпадающие списки, надписи, кнопки и ползунки, меню, и другие. Их добавление на форму осуществляется с помощью окна инструментов (Toolbox).



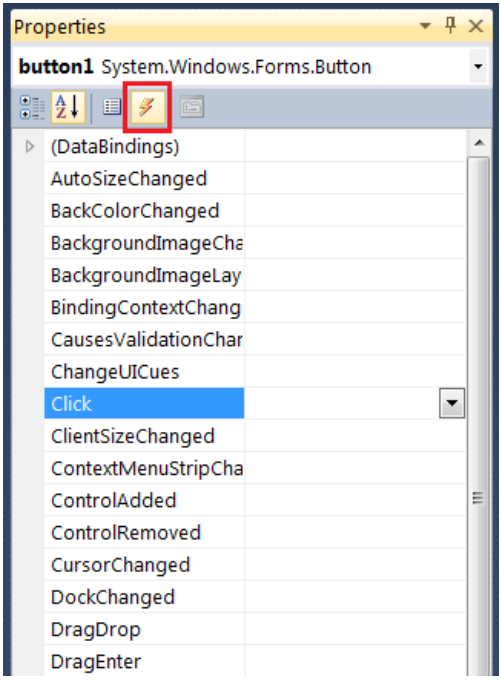
Если данное окно отсутствует, его можно вызвать из меню Вид (View) или с помощью клавиш Ctrl+W+X.

После добавления элементов на форму их можно использовать для выполнения необходимых действий, обращаясь к соответствующему полю, свойству или методу объекта, представляющего элемент управления, например для получения содержимого текстового поля используется его свойство Text:

```
string s = textBox1.Text;
```

Взаимодействие программы с пользователем или внешними системами осуществляется посредством событий. Событие представляет собой автоматическое уведомление о том, что произошло некоторое действие. Событие связано с объектом, над которым это действие было произведено. Например, клик мышкой на кнопке, нажатие клавиши в окне ввода, перемещение курсора по

форме и т.д. Если в программе необходимо произвести какие-либо действия при наступлении определённого события, то необходимо создать обработчик данного события. Для этого в окне свойств объекта необходимо переключиться на вкладку События (Events) и дважды щёлкнуть на событие, обработчик которого нужно создать.



Ниже приведён пример обработчика события нажатия на кнопку, который выполняет сложение двух чисел:

```
private void button1_Click(object sender, EventArgs e)
{
    double a = double.Parse(textBox1.Text);
    double b = double.Parse(textBox2.Text);
    double c = a + b;
    textBox3.Text = c.ToString();
}
```

Как правило, в одной программе имеется необходимость использовать несколько форм, выполняющих разные функции и обрабатывающих разные данные. При этом необходимо организовать обмен данными между этими формами. Для этого существу-

ют несколько способов.

1. Передача данных в конструктор формы Form2.

Для этого необходимо добавить в Form2 конструктор, принимающий необходимые параметры:

```
public partial class Form2 : Form
{
    double a;
    double b;

    public Form2()
    {
        InitializeComponent();
    }

    // добавленный конструктор //
    public Form2(double a, double b)
    {
        InitializeComponent();

        this.a = a;
        this.b = b;
    }
}
```

Тогда при вызове этой формы ей можно будет передавать значения:

```
private void button1_Click(object sender, EventArgs e)
{
    double a = double.Parse(textBox1.Text);
    double b = double.Parse(textBox2.Text);

    Form2 f2 = new Form2(a, b);
    f2.ShowDialog();
}
```

Данный метод является наиболее простым, однако позволяет передавать данные только в одном направлении — от первой формы ко второй.

2. Использование свойства «Владелец» (Owner).

При вызове второй формы необходимо указать её владельца:

```
f2.ShowDialog(this);
```

Ключевое слово this (это) означает, что владельцем создаваемой формы будет та форма, на которой её создают.

После этого из второй формы можно будет обращаться к любым полям первой формы, объявленным как public:

```
private void button1_Click(object sender, EventArgs e)
{
    ((Form1)Owner).textBox3.Text = "Привет";
    this.Close();
}
```

Задание

Вычислить и вывести на экран в виде таблицы значения функции из задания к лабораторной работе № 2 «Разветвляющиеся алгоритмы. Циклы» в интервале от $x_{нач}$ до $x_{кон}$ с шагом dx.

Все исходные данные для расчёта значений функции необходимо задавать на начальной форме, а результаты вычислений выводить на другой. После вычисления функции на исходной форме необходимо вывести количество посчитанных значений. Внешний вид форм должен соответствовать представленным на рисунках 1 и 2 соответственно.

Если введённые значения $x_{нач}$, $x_{кон}$ и dx не позволяют организовать цикл (например $x_{кон} < x_{нач}$ при положительном значении dx, или dx=0 и т.д.), то необходимо вывести сообщение об ошибке и не проводить расчётов. Вид сообщения приведён на рисунке 3.

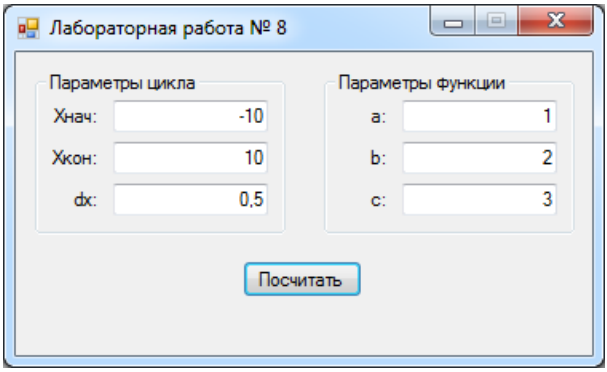
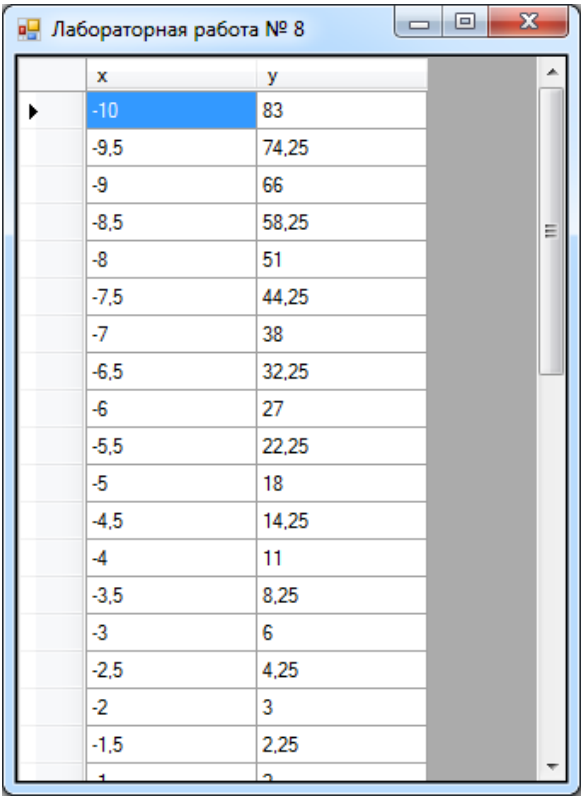


Рисунок 1 — Форма для ввода исходных данных.



x	y
-10	83
-9,5	74,25
-9	66
-8,5	58,25
-8	51
-7,5	44,25
-7	38
-6,5	32,25
-6	27
-5,5	22,25
-5	18
-4,5	14,25
-4	11
-3,5	8,25
-3	6
-2,5	4,25
-2	3
-1,5	2,25

Рисунок 2 — Форма для вывода значений функции.

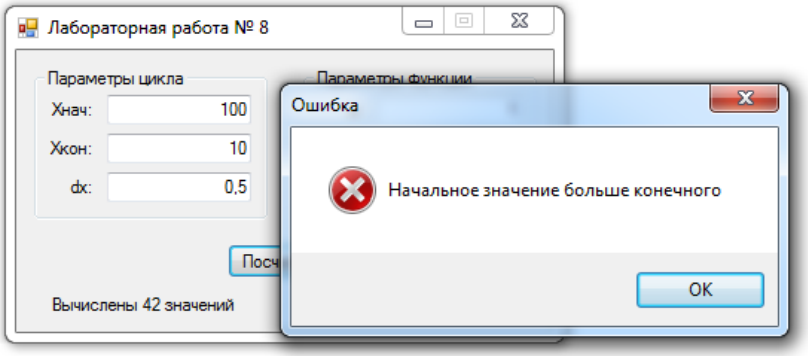


Рисунок 3 — Сообщение об ошибке.

Контрольные вопросы

1. Что такое форма?
2. Как создать новую форму?
3. Что такое событие?
4. Что такое обработчик события?
5. Как создать обработчик события?
6. Как передавать данные между формами?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:
— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задания,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 9

Тема: Работа с файлами. Сериализация.

Цель: Изучить способы сохранения данных в файл с использованием классов потокового ввода-вывода, изучить применение механизма сериализации для записи и чтения объектов из файла.

Теоретические сведения

Потоки

Ввод-вывод в программах на C# осуществляется посредством потоков. Поток — абстрактное представление устройства, облегчающее последовательное хранение данных и доступ к ним. В качестве такого устройства могут выступать расположенный на диске файл, принтер, область памяти, любой другой объект, допускающий последовательное считывание и запись информации. Все потоки действуют одинаково — даже если они связаны с разными физическими устройствами. Поэтому классы и методы ввода-вывода могут применяться к самым разным типам устройств. Например, методами вывода на консоль (Write(), WriteLine(), ReadLine()) можно пользоваться и для вывода в файл на диске.

Большинство устройств, предназначенных для выполнения операций ввода-вывода, являются байт-ориентированными. Этим и объясняется тот факт, что на самом низком уровне все операции ввода-вывода манипулируют с байтами в рамках байтовых потоков.

С другой стороны, значительный объём работ, для которых, собственно и используется вычислительная техника, предполагает работу с символами, а не с байтами (заполнение экранной формы, вывод информации в наглядном и легко читаемом виде, текстовые редакторы).

Символьно-ориентированные потоки, предназначенные для манипулирования с символами, а не с байтами, являются потоками ввода-вывода более высокого уровня. В рамках Framework.NET

определены соответствующие классы, которые при реализации операций ввода-вывода обеспечивают автоматическое преобразование данных типа byte в данные типа char и обратно.

В дополнение к байтовым и символьным потокам в C# определены два класса, реализующих механизмы считывания и записи информации непосредственно в двоичном представлении.

Классы потоков, определены в пространстве имен System.IO. Поэтому в самом начале любой использующей их программы обычно вводится следующая строка кода:

```
using System.IO;
```

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

```
FileStream(string filename, FileMode mode)
```

где:

1. параметр filename определяет имя файла, с которым будет связан поток ввода-вывода данных; при этом filename определяет либо полный путь к файлу, либо имя файла, который находится в папке bin/debug вашего проекта.
2. параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением FileMode:

— `FileMode.Append` — предназначен для добавления данных в конец файла;

— `FileMode.Create` — предназначен для создания нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;

— `FileMode.CreateNew` — предназначен для создания нового файла, при этом файл с таким же именем не должен существовать;

— `FileMode.Open` — предназначен для открытия суще-

ствующего файла;

— `FileMode.OpenOrCreate` — если файл существует, то открывает его, в противном случае создает новый;

— `FileMode.Truncate` — открывает существующий файл, но усекает его длину до нуля.

Для организации выходного символьного потока предназначен класс StreamWriter. В нем определено несколько конструкторов. Один из них позволяет открыть поток сразу через обращения к файлу и записывается следующим образом:

```
StreamWriter(string путь)
```

где параметр «путь» определяет имя определяет имя открываемого файла.

Другой вариант конструктора записывается следующим образом:

```
StreamWriter(string путь, bool appendFlag);
```

где параметр appendFlag может принимать значение true — если нужно добавлять данные в конец файла, или false — если файл необходимо перезаписать.

Класс StreamReader предназначен для организации входного символьного потока. Он имеет конструктор, аналогичный конструктору класса StreamWriter:

```
StreamReader (string путь)
```

где параметр путь определяет имя открываемого файла.

Сериализация

Сериализация представляет собой процесс преобразования объекта в поток байтов с целью сохранения его в памяти, в базе данных или в файле. Ее основное назначение — сохранить состояние объекта для того, чтобы иметь возможность воссоздать его при необходимости.

Для того, чтобы объект класса можно было сериализовать,

класс должен быть помечен как сериализуемый следующим образом:

```
[Serializable]
class MyClass
{
 …
}
```

Существуют 2 вида сериализации: двоичная и XML-сериализация. Двоичная работает быстрее и полученный файл имеет меньший размер, XML позволяет получить файл пригодный для чтения и редактирования.

Независимо от вида сериализации эта процедура включает в себя следующие этапы:

1. Создание файлового потока для вывода.
2. Создание объекта, выполняющего сериализацию.
3. Преобразование объекта в поток байт и запись в файл.

Для бинарной сериализации эти этапы будут реализованы в программе следующим образом:

```
FileStream f = new FileStream("file.bin", FileMode.Create);
Sys-
tem.Runtime.Serialization.Formatters.Binary.BinaryForm
atter bf =
    new Sys-
tem.Runtime.Serialization.Formatters.Binary.BinaryForm
atter();
bf.Serialize(f, obj);
f.Close();
```

где obj — имя сериализуемого объекта.

Для xml-сериализации:

```
FileStream fs = new System.IO.FileStream("file.xml", File-
Mode.Create);
```

```
System.Xml.Serialization.XmlSerializer sr =
    new Sys-
tem.Xml.Serialization.XmlSerializer(obj.GetType());
sr.Serialize(fs, obj);
fs.Close();
```

Обратный процесс называется десериализацией. Он также состоит из трёх этапов:

1. Создание файлового потока для ввода.

2. Создание объекта, выполняющего десериализацию (аналогичного тому, с помощью которого выполнялась сериализация).

3. Чтение из файла и преобразование полученного объекта к нужному типу.

Бинарная десериализация:

```
FileStream f = new FileStream("file.bin", File-
Mode.Open);
Sys-
tem.Runtime.Serialization.Formatters.Binary.BinaryForma
tter bf =
    new Sys-
tem.Runtime.Serialization.Formatters.Binary.BinaryForma
tter();
t = (MyClass)bf.Deserialize(f);
```

Xml-десереализация:

```
FileStream fs = new System.IO.FileStream("file.xml",
FileMode.Open);
System.Xml.Serialization.XmlSerializer sr =
    new Sys-
tem.Xml.Serialization.XmlSerializer(obj.GetType());
t = (MyClass)sr.Deserialize(fs);
fs.Close();
```

Задание

Реализовать отображение информации об объектах класса, разработанного в лабораторной работе № 7, в DataGridView.

Создать меню содержащее пункты «Сохранить…» и «Загрузить…».

Реализовать сохранение в файл, выбранный пользователем, и загрузку из файла массива объектов класса, разработанного в лабораторной работе № 7. Для сохранения использовать как потоковые классы, так и сериализацию. Для выбора файла использовать классы OpenFileDialog и SaveFileDialog.

Контрольные вопросы

1. Что такое поток?
2. Какие виды потоков существуют в C#?
3. В чём разница между различными типами потоков?
4. Для чего предназначены классы StreamWriter и StreamReader?
5. Что такое сериализация?
6. В чём разница между различными видами сериализации?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:

— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задание в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.

ЛАБОРАТОРНАЯ РАБОТА № 10

Тема: Коллекции.

Цель: Изучение классов коллекций в языке C# и способов их применения при решении практических задач.

Теоретические сведения

Коллекция в C# представляет собой совокупность объектов. Коллекции упрощают решение многих задач программирования благодаря тому, что предлагают готовые решения для создания целого ряда типичных, но порой трудоемких для разработки структур данных. Например, в среду .NET Framework встроены коллекции, предназначенные для поддержки динамических массивов, связных списков, стеков, очередей и хеш-таблиц. Главное преимущество коллекций заключается в том, что они стандартизируют обработку групп объектов в программе.

Часто используемым примером коллекций является список (List). В классе List<T> реализуется обобщенный динамический массив. Списки применяются там, где количество элементов в коллекции наперёд не известно. Они выгодно отличаются от массивов тем, что по ходу выполнения программы их размер можно изменять в любую сторону.

В программе можно создавать списки объектов любого типа, например:

```
List<double> l = new List<double>();
```

создаст пустой список объектов типа double (вещественных чисел), а:

```
List<MyClass> l2 = new List<MyClass>();
```

создаст список объектов типа MyClass.

Для добавления элементов в список используется метод Add(), например:

```
l.Add(5);
```

```
l.Add(10);
```

добавит в список с именем l два элемента — сначала число 5, затем число 10. При этом размерность списка автоматически увеличится на 2.

Доступ к элементам списка осуществляется аналогично доступу к элементам массива — с помощью оператора [].

Помимо добавления, список имеет следующие несколько методов, реализующих стандартные и часто необходимые действия над элементами списка.

— Clear — удаляет все элементы из списка;

— Contains — определяет, входит ли элемент в состав списка;

— IndexOf — возвращает отсчитываемый от нуля индекс первого вхождения значения в список;

— Insert — добавляет элемент в список в позиции с указанным индексом;

— LastIndexOf — возвращает отсчитываемый от нуля индекс последнего вхождения значения в список;

— Remove — удаляет первое вхождение указанного объекта из списка

— Sort — сортирует элементы в списке;

— ToArray — копирует элементы списка в обычный массив

— Max — возвращает значение максимального элемента списка;

— Min — возвращает значение минимального элемента списка;

— FindAll — извлекает все элементы, удовлетворяющие условиям указанного предиката;

— FindIndex — выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого найденного вхождения в пределах всего списка;

— RemoveAll — удаляет все элементы, удовлетворяющие

условиям указанного предиката.

Последние несколько методов требуют указания в качестве параметра предиката — логической функции, задающей определённое условие, которому могут соответствовать или не соответствовать элементы списка. Например, для удаления из списка всех элементом равных 0 необходимо создать следующий предикат и затем использовать его в методе RemoveAll:

```
Predicate<double> p = FindZero;// сам объект класса Predicate

public static bool FindZero(double val)// логическая функция
{
    return val == 0;
}

public void UdalitVseNuli()
{
    l.RemoveAll(p);
}
```

Задание

1. Выполнить задание из лабораторной работы № 3 (Одномерные массивы) используя вместо массивов списки. При возможности использовать имеющиеся в классе List методы для сортировки, поиска минимальных/максимальных значений, удаления элементов и т.д.

2. При выполнении задания использовать возможности объектно-ориентированного программирования:

создать класс Lab3, содержащий поле типа List<double> для хранения массива и методы, выполняющие первую и вторую части задания.

3. Программа должна иметь графический интерфейс, в ко-

тором для ввода элементов массива используется компонент DataGridView.

Контрольные вопросы

1. Что такое коллекции?
2. Что такое список?
3. Как создать аналог двумерного массива, используя списки?
4. Перечислите основные методы класса список.
5. Что такое предикат и для чего он применяется?

Требования к отчёту

Отчёт по лабораторной работе должен содержать:
— титульный лист,
— цель работы,
— краткие теоретические сведения,
— задание в соответствии с вариантом,
— листинг программы,
— пример работы программы в виде скриншотов,
— вывод.