**Лабораторная работа №1**

**Задание 1.** Для создания нового проекта выполните следующие действия (см. Приложение 1):

1. Создайте новый проект: File ► New Project, либо Сreate Project в окне Start Page

2. В окне New project в левой части выберите Visual C# Projects, в правой – пункт Console Application

3. В поле Name введите имя проекта, в поле Location – место его сохранения на диске

4. Ознакомьтесь с основными окнами среды.

5. Рассмотрите каждую строку заготовки программы.

6. Наберите приведенный пример программы (Листинг 1).Вставьте свои значения соответствующих типов в пропущенных местах операторов.

Листинг исходного кода:

using System;

namespace matemat1

{ class Program

{public static void Main(string[] args)

//после знака = для x,y,z вставьте Ваши данные

{int x= ;

double y= ;

Console.WriteLine("x: " + x);

double b = ;

Console.WriteLine ("y: " + y);}}}

7. Сохраните весь проект на диске: File ► Save All

8. Для выполнения программы: Debug ► Start Without Debugging

Листинг программы:

using System;

namespace matemat1

{

class Program

{

public static void Main(string[] args)

//после знака = для x,y,z вставьте Ваши данные

{int x = 2;

double y = 5;

Console.WriteLine("x: " + x);

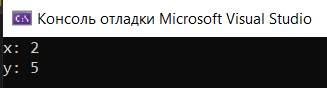
double b = 25;

Console.WriteLine("y: " + y);

}

}

}

Рисунок 1 – Полученный результат

**Задание 2.** Создайте новое консольное приложение для решения задачи.

Введите вещественные числа x, у, z из области допустимых значений исходных данных. Для преобразования к числовой форме используйте класс Convert и метод Parse. Вычислите а, b. Результаты выведите на экран с использованием формата и шаблонов.



Рисунок 2 – Решаемые примеры

Листинг программы:

using System;

namespace LR1T2

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите x: ");

double x = double.Parse(Console.ReadLine());

Console.Write("Введите y: ");

double y = double.Parse(Console.ReadLine());

Console.Write("Введите z: ");

double z = double.Parse(Console.ReadLine());

double a = (Math.Pow(Math.Abs(x), 0.5) \* (x - (Math.Pow(y, 3) / (z + Math.Pow(x, 2)))));

Console.WriteLine("a= {0}", a);

double b = (x - (Math.Pow(x, 2) / 2) + (Math.Pow(x, 5) / 5));

Console.WriteLine("b= {0}", b);

}

}

}

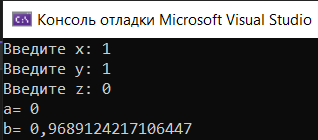


Рисунок 3 – Полученный результат

**Контрольные вопросы:**

1. Основные принципы технологии .NET.

− Платформенная независимость: Приложения .NET могут выполняться на различных операционных системах благодаря использованию среды исполнения CLR (Common Language Runtime).

− Безопасность типов: Все операции проверяются на этапе компиляции и выполнения, чтобы избежать ошибок, связанных с неправильной работой с памятью.

− Многоязыковая поддержка: Поддержка множества языков программирования через общий промежуточный язык CIL (Common Intermediate Language), который транслируется в машинный код на этапе выполнения.

− Унифицированная библиотека классов: .NET Framework включает обширную библиотеку классов, упрощающую разработку приложений.

2. Что представляет собой платформа Visual [Studio.NET](https://web.telegram.org/a/Studio.NET)?

Visual Studio — это интегрированная среда разработки (IDE), предназна-ченная для создания приложений на платформе .NET. Она поддерживает различ-ные языки программирования, такие как C#, [VB.NET](https://web.telegram.org/a/VB.NET), F#, C++ и другие. Встроен-ный редактор кода, инструменты отладки, система управления версиями и другие функции делают её мощным инструментом для разработчиков.

3. Как создать консольное приложение?  
Чтобы создать консольное приложение в Visual Studio:

1. Откройте Visual Studio.

2. Выберите «Файл» → «Создать» → «Проект».

3. Найдите шаблон «Консольное приложение (.NET)» и выберите его.

4. Задайте имя проекта и нажмите «ОК».

4. Принципы объектно-ориентированного программирования.

− Инкапсуляция: Скрытие реализации класса и предоставление интерфейсов для взаимодействия с ним.

− Наследование: Возможность создавать новые классы на основе существующих, расширяя их функциональность.

− Полиморфизм: Способность объектов разных классов реагировать на одинаковые вызовы методов по-разному.

5. Литералы. Как определяются типы литералов?  
Литералы — это фиксированные значения, которые записываются непосредственно в исходном коде. Типы литералов в C# определяются автоматически на основании их формы записи:

− Целочисленные литералы: “int”, “long” (например, “123L”).

− Дробные литералы: “float”, “double” (например, “3.14F”).

− Символьные литералы: “char” (например, “A“).

− Строковые литералы: “string” (например, “Hello`).

6. Какие типы относятся к встроенным?

Встроенными типами являются базовые типы данных, предоставляемые языком C#:

− “bool”: Логический тип (“true”/”false”).

− “byte”, “sbyte”: 8-битные целые числа.

− “short”, “ushort”: 16-битные целые числа.

− “int”, “uint”: 32-битные целые числа.

− “long”, “ulong”: 64-битные целые числа.

− “float”, “double”: Числовые типы с плавающей точкой.

− “decimal”: Высокоточный числовой тип.

− “char”: Символ Unicode.

− “object”: Базовый класс всех типов.

− “string”: Последовательность символов Unicode.

7. Чем отличаются типы-значения и ссылочные типы?

Типы значений хранятся непосредственно в стеке памяти и содержат сами данные. Примеры: int, struct.

Ссылочные типы хранят ссылку на область памяти, где находятся данные. Примеры: class, interface, delegate. Когда вы присваиваете один ссылочный тип другому, копируется ссылка, а не сам объект.

8. Какие типы числовых переменных имеются?

Числовые переменные делятся на два вида:

− Целочисленные: byte, sbyte, short, ushort, int, uint, long, ulong.

− Дробные: float, double, decimal.

9. Что такое объявление и инициализация?

Объявление – это создание переменной с указанием её типа.

Инициализация – присвоение начального значения переменной.

10.Для чего используется упаковка и распаковка?

− Упаковка (boxing) — это процесс преобразования типа-значения в тип-объект (например, упаковать int в объект типа object).

− Распаковка (unboxing) — это преобразование объекта обратно в тип-значение.

11.Как в С# выполняется преобразование типа?

− Преобразование может быть неявным (например, от меньшего к большему типу) или явным (например, (int)3.14). Также используется метод Convert.

12.Как осуществляется консольный ввод?

− Используется класс Console. Для чтения строки используется метод Console.ReadLine(), для чтения символа — метод Console.Read().

13.Чем отличаются методы Read и ReadLine?

Метод ReadLine считывает данные и очищает буфер. Метод Read считывает из буфера только один символ, и в отличие от ReadLine, не очищает буфер, по-этому следующий после него ввод будет выполняться с того места, на котором за-кончился предыдущий, т.е. будет читаться код клавиши . Поэтому необходимо прочитать остаток строки методом ReadLine().

14.Как обеспечить вывод данных на экран?

Для вывода данных на экран используется метод Console.WriteLine() или Console.Write().

15.Для чего предназначен и как используется форматный вывод дан-ных?

Форматный вывод позволяет контролировать представление данных (например, количество знаков после запятой). Используется с помощью метода Console.WriteLine() с форматированием (например, "Value: {0:F2}", value).

16.Каковы основные правила использования стандартных функций?

Следует соблюдать синтаксис вызова функции, передавать необходимые параметры, учитывать область видимости переменных и обрабатывать возможные исключения.

17.Основные приемы работы в среде разработки Visual [Studio.NET](https://web.telegram.org/a/Studio.NET):

− Как создать консольное приложение?

− Откройте Visual Studio.

− Выберите «Файл» → «Создать» → «Проект».

− Найдите шаблон «Консольное приложение (.NET)» и выберите его.

− Задайте имя проекта и нажмите «ОК».

− Как сохранить проект с заданным именем?

− Выберите "Файл" → "Сохранить как..." и укажите имя.

− Как загрузить проект?

− Выберите "Файл" → "Открыть" → "Проект/Решение" и выберите файл проекта.

− Как выполнить отладку программы?

− Нажмите F5 или выберите "Отладка" → "Запустить отладку"

− Как откомпилировать и выполнить программу?

− Нажмите Ctrl + F5 для компиляции и выполнения без отладки.

− Как просмотреть результаты выполнения программы?

− Результаты выводятся в окне консоли или в окне вывода Visual Studio.

**Лабораторная работа № 2**

**Объявление и определение методов**

**Задание 1.** Создайте проект для решения задачи: На экран выводить исходные данные и результаты. В работе использовать только стандартные типы: числовые, символьный и булевский.

9. По введенному символу и значению аргумента выведите значение одной из функций sin(x), cos(x), tg(x), ctg(x) и соответствующий комментарий.

Листинг программы:

using System;

class Program

{

static void Main()

{

Console.Write("Введите символ функции (sin, cos, tg, ctg): ");

string function = Console.ReadLine().Trim().ToLower();

Console.Write("Введите значение аргумента (в радианах): ");

if (!double.TryParse(Console.ReadLine(), out double x))

{

Console.WriteLine("Неверный ввод для значения аргумента.");

return;

}

double result = 0;

string comment = "";

switch (function)

{

case "sin":

result = Math.Sin(x);

comment = "Синус заданного угла.";

break;

case "cos":

result = Math.Cos(x);

comment = "Косинус заданного угла.";

break;

case "tg":

result = Math.Tan(x);

comment = "Тангенс заданного угла.";

break;

case "ctg":

if (Math.Tan(x) != 0)

{

result = 1.0 / Math.Tan(x);

comment = "Котангенс заданного угла.";

}

else

{

Console.WriteLine("Котангенс не определен для этого угла (разделение на ноль).");

return;

}

break;

default:

Console.WriteLine("Неверный символ функции.");

return;

}

Console.WriteLine($"Результат: {result}");

Console.WriteLine(comment);

}

}

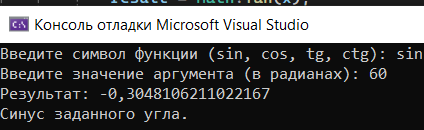


Рисунок 4 – Полученный результат

**Задание 2.** Напишите функции в виде методов. Напишите тестирующую программу с выдачей результатов на экран.

9. Напишите программу, которая вычисляет сумму цифр натурального числа, если число - четное, и остаток от деления на 10, если нечетное.

Листинг программы:

using System;

class Program

{

static void Main()

{

Console.Write("Введите натуральное число: ");

if (!int.TryParse(Console.ReadLine(), out int number) || number <= 0)

{

Console.WriteLine("Введите корректное натуральное число.");

return;

}

if (number % 2 == 0)

{

// Четное число: вычисляем сумму цифр

int sum = 0;

while (number > 0)

{

sum += number % 10; // Получаем последнюю цифру

number /= 10; // Убираем последнюю цифру

}

Console.WriteLine($"Сумма цифр: {sum}");

}

else

{

// Нечетное число: вычисляем остаток от деления на 10

int remainder = number % 10;

Console.WriteLine($"Остаток от деления на 10: {remainder}");

}

}

}

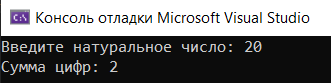


Рисунок 5 – Полученный результат

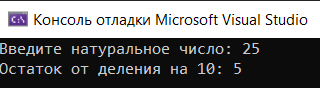


Рисунок 6 – Полученный результат

**Контрольные вопросы:**

1. Выражением какого типа является условие в операторе if? Какие значения оно может принимать?

Условие в операторе if в C# является булевым выражением (тип bool). Оно может принимать два значения: true (истина) или false (ложь).

2. Как работает оператор if, если отсутствует часть else <оператор2>?

Если часть else отсутствует, то оператор if выполняет блок кода, если условие истинно. Если условие ложно, выполнение продолжается с следующей строки после блока if, и никаких дополнительных действий не происходит.

3. В каких случаях используется оператор switch?

Оператор switch используется для выбора одного из нескольких блоков кода для выполнения на основе значения выражения. Он удобен, когда нужно проверить одно значение на соответствие множеству возможных вариантов, например, при работе с перечислениями или конкретными значениями.

4. Какого типа может быть <выражение> в операторе switch?

В C# выражение в операторе switch должно быть целочисленного типа (int, char, byte, short), строкового типа (string) или перечислением (enum).

5. В каком случае выполняется последовательность инструкций default-ветви?

Последовательность инструкций в default-ветви выполняется, если ни одно из значений в case не соответствует значению выражения в операторе switch.

6. В чем отличие операторов while и do ... while

Основное отличие заключается в том, что в цикле while условие проверяется перед выполнением тела цикла, а в цикле do ... while тело цикла выполняется хотя бы один раз, так как условие проверяется после выполнения.

7. Что представляет собой элемент <инициализация> в операторе for?

Элемент инициализации в операторе for используется для инициализации переменной цикла и выполняется один раз перед началом цикла. Например: for (int i = 0; i < 10; i++) { /\* тело цикла \*/ }

8. Какого типа может быть элемент <условие> в цикле for?.

Элемент условия в цикле for должен быть булевым выражением (bool), которое определяет, будет ли цикл продолжаться.

9.Назначение управляющей переменной цикла for?

Управляющая переменная цикла for используется для отслеживания текущего состояния итерации цикла и управления количеством повторений.

10.Назначение управляющих операторов goto, break, continue, return.

• goto: используется для безусловного перехода к другой метке в коде.

• break: завершает выполнение текущего цикла или оператора switch.

• continue: пропускает оставшуюся часть текущей итерации цикла и переходит к следующей итерации.

• return: завершает выполнение метода и возвращает значение (если метод не void).

11.Как программируются циклические алгоритмы с явно заданным числом повторений цикла?

Циклические алгоритмы с явно заданным числом повторений обычно программируются с помощью цикла for, где количество итераций задается в условии: for (int i = 0; i < 10; i++) { // действия }

12.Что представляют собой методы?

Методы представляют собой блоки кода, которые выполняют определенные действия и могут принимать параметры и возвращать значения. Они помогают организовать код и повторно использовать его.

13.Как объявляется метод?

Метод объявляется с указанием модификаторов доступа (например, public, private), возвращаемого типа, имени метода и списка параметров в круглых скобках. Например: int Sum(int a, int b) { return a + b;}

14.Какова область действия параметров метода?

Параметры метода имеют область действия внутри этого метода и недоступны за его пределами.

15.Как вызываются методы?

Методы вызываются по их имени, с указанием необходимых аргументов (если они есть). Например: int result = Sum(5, 10);

16.Общие (статические) методы класса.

Общие (статические) методы класса объявляются с использованием ключевого слова static. Они могут быть вызваны без создания экземпляра класса и могут обращаться только к статическим переменным и методам класса. Например: public static void DisplayMessage() {Console.WriteLine("Hello, world!");}

**Лабораторная работа №3**

**ПРОСТЕЙШИЕ КЛАССЫ.**

**ИНКАПСУЛЯЦИЯ И СВОЙСТВА.**

**Задание 1.** Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), методы и свойства. Методы и свойства должны обеспечивать непротиворечивый и удобный интерфейс класса.

В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта.

9. Описать класс, представляющий четырехугольник. Разработать методы для определения является ли четырехугольник параллелограммом.

using System;

class Quadrilateral

{

// Поля для хранения координат вершин четырехугольника

private double x1, y1;

private double x2, y2;

private double x3, y3;

private double x4, y4;

// Конструктор с параметрами для инициализации вершин

public Quadrilateral(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.x3 = x3;

this.y3 = y3;

this.x4 = x4;

this.y4 = y4;

}

// Метод для определения, является ли четырехугольник параллелограммом

public bool IsParallelogram()

{

// Проверяем, что противоположные стороны параллельны

double slope1 = (y2 - y1) / (x2 - x1);

double slope2 = (y4 - y3) / (x4 - x3);

return Math.Abs(slope1 - slope2) < 0.0001;

}

// Метод для ввода координат вершин

public static Quadrilateral InputQuadrilateral()

{

Console.WriteLine("Введите координаты вершин четырехугольника:");

Console.Write("x1: ");

double x1 = double.Parse(Console.ReadLine());

Console.Write("y1: ");

double y1 = double.Parse(Console.ReadLine());

Console.Write("x2: ");

double x2 = double.Parse(Console.ReadLine());

Console.Write("y2: ");

double y2 = double.Parse(Console.ReadLine());

Console.Write("x3: ");

double x3 = double.Parse(Console.ReadLine());

Console.Write("y3: ");

double y3 = double.Parse(Console.ReadLine());

Console.Write("x4: ");

double x4 = double.Parse(Console.ReadLine());

Console.Write("y4: ");

double y4 = double.Parse(Console.ReadLine());

return new Quadrilateral(x1, y1, x2, y2, x3, y3, x4, y4);

}

}

class Program

{

static void Main()

{

// Вводим координаты вершин четырехугольника

Quadrilateral quad = Quadrilateral.InputQuadrilateral();

// Проверяем, является ли он параллелограммом

if (quad.IsParallelogram())

{

Console.WriteLine("Четырехугольник является параллелограммом.");

}

else

{

Console.WriteLine("Четырехугольник не является параллелограммом.");

}

}

}

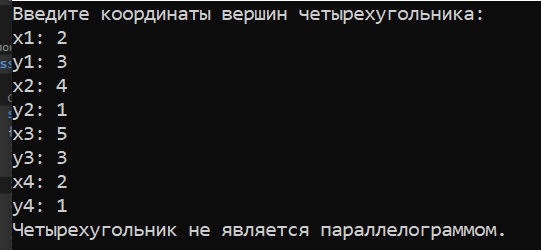


Рисунок 7 – Полученный результат

**Задание 2**. Включите в проект Задания 1обработку исключений.

using System;

class Quadrilateral

{

// Поля для хранения координат вершин четырехугольника

private double x1, y1;

private double x2, y2;

private double x3, y3;

private double x4, y4;

// Конструктор с параметрами для инициализации вершин

public Quadrilateral(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.x3 = x3;

this.y3 = y3;

this.x4 = x4;

this.y4 = y4;

}

// Метод для определения, является ли четырехугольник параллелограммом

public bool IsParallelogram()

{

// Проверяем, что противоположные стороны параллельны

double slope1 = (y2 - y1) / (x2 - x1);

double slope2 = (y4 - y3) / (x4 - x3);

return Math.Abs(slope1 - slope2) < 0.0001;

}

// Метод для ввода координат вершин

// Метод для ввода координат вершин

public static Quadrilateral InputQuadrilateral()

{

try

{

Console.WriteLine("Введите координаты вершин четырехугольника:");

Console.Write("x1: ");

double x1 = double.Parse(Console.ReadLine());

Console.Write("y1: ");

double y1 = double.Parse(Console.ReadLine());

Console.Write("x2: ");

double x2 = double.Parse(Console.ReadLine());

Console.Write("y2: ");

double y2 = double.Parse(Console.ReadLine());

Console.Write("x3: ");

double x3 = double.Parse(Console.ReadLine());

Console.Write("y3: ");

double y3 = double.Parse(Console.ReadLine());

Console.Write("x4: ");

double x4 = double.Parse(Console.ReadLine());

Console.Write("y4: ");

double y4 = double.Parse(Console.ReadLine());

return new Quadrilateral(x1, y1, x2, y2, x3, y3, x4, y4);

}

catch (FormatException ex)

{

Console.WriteLine("Ошибка формата ввода: " + ex.Message);

return null; // Возвращаем null в случае ошибки

}

}

}

class Program

{

static void Main()

{

// Вводим координаты вершин четырехугольника

Quadrilateral quad = Quadrilateral.InputQuadrilateral();

// Проверяем, является ли он параллелограммом

if (quad.IsParallelogram())

{

Console.WriteLine("Четырехугольник является параллелограммом.");

}

else

{

Console.WriteLine("Четырехугольник не является параллелограммом.");

}

}

}

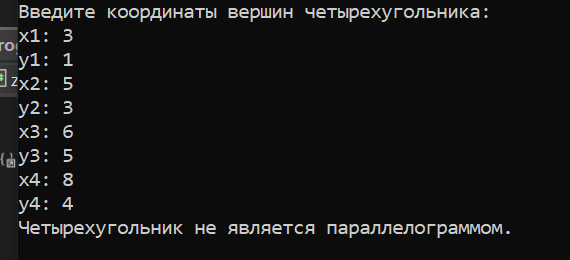


Рисунок 8 – Полученный результат

**Контрольные вопросы:**

1. Как описываются классы в С#?

Классы в C# описываются с использованием ключевого слова class, за которым следует имя класса и тело класса в фигурных скобках. Пример описания класса:

public class MyClass

{

// Поля, свойства, методы и другие члены класса

}

2. Что относится к членам класса?

Члены класса включают в себя поля (variables), свойства (properties), методы (methods), события (events), индексаторы (indexers) и конструкторы (constructors). Эти члены определяют состояние и поведение объектов, созданных на основе класса.

3. Что такое статические члены класса?

Статические члены класса относятся к классу в целом, а не к конкретным экземплярам класса. Они могут быть вызваны без создания объекта класса и обычно используются для хранения общей информации или выполнения операций, не зависящих от конкретного состояния объекта. Статические члены класса объявляются с использованием ключевого слова static.

4. Данные: поля и константы.

В классах C# данные могут быть представлены с помощью полей и констант. Поля (fields) представляют переменные, которые хранят состояние объекта класса, а константы (constants) представляют значения, которые остаются постоянными на протяжении выполнения программы.

5. Спецификаторы полей и констант класса.

Спецификаторы полей класса могут включать в себя:

* public: поле доступно из любого места в программе.
* private: поле доступно только внутри класса.
* protected: поле доступно внутри класса и его производных классов.
* internal: поле доступно внутри сборки.
* protected internal: поле доступно внутри сборки и его производных классов.

Спецификаторы констант класса:

* const: определяет константу времени компиляции.
* readonly: определяет константу времени выполнения, которая может быть установлена только в конструкторе класса или при объявлении

6. Как передаются параметры в методы?

Параметры передаются в методы путем указания их типов и имен в списке параметров метода в круглых скобках при объявлении метода. При вызове метода значения аргументов передаются в соответствии с порядком параметров.

7. Для чего предназначен параметр params?

Параметр params в C# позволяет методу принимать переменное количество аргументов одного типа. При использовании параметра params можно передавать любое количество аргументов указанного типа, и компилятор автоматически упаковывает их в массив.

8. Что представляет собой конструктор? Для чего он используется?

Конструктор в C# является специальным методом класса, который вызывается при создании нового объекта этого класса. Он используется для инициализации объекта, установки начальных значений полей, выполнения необходимых действий при создании объекта.

9. Какие бывают конструкторы?

В C# существуют следующие типы конструкторов:

* + Параметризованный конструктор: принимает параметры для инициализации объекта.
  + Конструктор по умолчанию: не принимает параметры и используется, если не определены другие конструкторы.
  + Конструктор копирования: создает новый объект на основе существующего объекта.

10. Может ли класс не иметь конструктора?

Да, класс может не иметь явно определенных конструкторов. В этом случае компилятор автоматически создает конструктор по умолчанию (без параметров), если в классе не определены другие конструкторы.

11. Для чего предназначена система сбора мусора?

Система сбора мусора в C# предназначена для автоматического управления памятью и освобождения ресурсов, занимаемых объектами, когда они больше не нужны. Сборщик мусора отслеживает объекты, на которые нет ссылок из программы, и освобождает память, занимаемую этими объектами, чтобы предотвратить утечку памяти и повысить производительность программы.

**Лабораторная работа №4**

**Задание 1.** Создайте проект, в котором опишите класс для решения задачи Вашего варианта. Каждый разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы с параметрами и без параметров, методы, свойства, индексаторы.

Класс должен реализовывать следующие операции над массивами:

* задание произвольной размерности массива при создании объекта;
* доступ к элементу по индексам с контролем выхода за пределы массива;
* вывод на экран элемента массива по заданному индексу и всего массива.

При возникновении ошибок должны выбрасываться исключения.

В программе должна выполняться проверка всех разработанных элементов класса.

9. Описать класс для работы с двумерным числовым массивом. Обеспечить возможность проверки, является ли элемент массива палиндромом. Палиндром принимает одно и то же значение при чтении его как справа налево, так и слева направо.

using System;

class TwoDimensionalArray

{

private int[,] array;

public TwoDimensionalArray(int[,] inputArray)

{

array = inputArray;

}

// Метод для проверки, является ли элемент массива палиндромом

public bool IsElementPalindrome(int row, int col)

{

int value = array[row, col];

string valueAsString = value.ToString();

// Проверяем, что значение одинаково при чтении справа налево и слева направо

for (int i = 0; i < valueAsString.Length / 2; i++)

{

if (valueAsString[i] != valueAsString[valueAsString.Length - 1 - i])

{

return false;

}

}

return true;

}

// Метод для вывода массива

public void PrintArray()

{

int rows = array.GetLength(0);

int cols = array.GetLength(1);

Console.WriteLine("Двумерный массив:");

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

Console.Write($"{array[i, j]} ");

}

Console.WriteLine();

}

}

}

class Program

{

static void Main()

{

// Пример создания двумерного массива

int[,] myArray = {

{ 121, 234, 345 },

{ 456, 787, 123 },

{ 321, 111, 222 }

};

TwoDimensionalArray arrayHandler = new TwoDimensionalArray(myArray);

// Выводим массив

arrayHandler.PrintArray();

// Запрашиваем строку и столбец для проверки

Console.Write("Введите номер строки: ");

int rowToCheck = int.Parse(Console.ReadLine());

Console.Write("Введите номер столбца: ");

int colToCheck = int.Parse(Console.ReadLine());

// Проверяем элемент на палиндром

bool isPalindrome = arrayHandler.IsElementPalindrome(rowToCheck, colToCheck);

if (isPalindrome)

{

Console.WriteLine($"Элемент в строке {rowToCheck}, столбце {colToCheck} является палиндромом.");

}

else

{

Console.WriteLine($"Элемент в строке {rowToCheck}, столбце {colToCheck} не является палиндромом.");

}

}

}

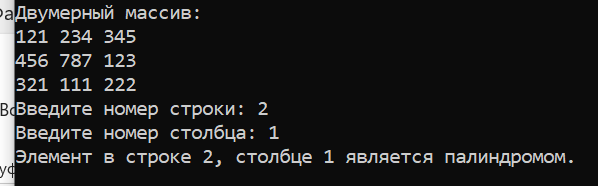


Рисунок 9 – Полученный результат

**Контрольные вопросы:**

1. Что понимается под массивом?

Под массивом понимается структура данных, которая представляет собой упорядоченный набор элементов одного типа, расположенных в памяти компьютера последовательно. Каждый элемент массива имеет свой индекс, который позволяет обращаться к нему.

2. Каковы возможные способы описания массивов (одномерных и многомерных)?

В C# возможны следующие способы описания массивов:

* + Одномерные массивы: объявляются с помощью квадратных скобок [], например int[] numbers = new int[5];.
  + Многомерные массивы: могут быть двумерными, трехмерными и т.д. Объявляются с помощью запятых между квадратными скобками, например int[,] matrix = new int[3, 3];.

3. В каких случаях целесообразно описывать двумерный массив с помощью одномерных?

Целесообразно описывать двумерный массив с помощью одномерных в случаях, когда удобнее работать с данными в одномерной структуре. Например, если требуется представить матрицу в виде одномерного массива для оптимизации работы с памятью или для удобства обработки данных. При этом индексация элементов в одномерном массиве может быть более гибкой и удобной для определенных операций.

4. Какие типы допустимы для описания индексов массивов?

Для описания индексов массивов в C# допустимы следующие типы:

* + - Целочисленные типы: int, uint, long, ulong, short, ushort, byte, sbyte, char.
    - Перечисления (enum), основанные на целочисленных типах.
    - System.Index и System.Range (начиная с C# 8.0).

5. Какие типы могут использоваться в качестве базовых для описания массивов?

В качестве базовых типов для описания массивов в C# могут использоваться любые типы данных, включая примитивные типы (например, int, double, char), пользовательские классы, структуры, перечисления и другие массивы.

6. Как осуществляется ввод и вывод массивов?

Ввод и вывод массивов в C# можно осуществить с помощью циклов, методов класса Console или других специализированных методов. Например, для ввода массива можно использовать цикл for или foreach, считывая значения с клавиатуры и сохраняя их в элементы массива. Для вывода массива можно также использовать циклы или методы вывода Console.WriteLine.

7. Для чего предназначен цикл foreach?

Цикл foreach в C# предназначен для перебора элементов коллекции или массива без явного использования индексов. Он удобен для случаев, когда не требуется знать текущий индекс элемента, а нужно просто последовательно обработать все элементы коллекции. foreach автоматически перебирает все элементы коллекции от начала до конца.

8. Можно ли использовать цикл foreach для ввода элементов массива?

Нет, цикл foreach не предназначен для ввода элементов массива. Цикл foreach используется для перебора элементов коллекции или массива, но не позволяет изменять значения элементов. Для ввода элементов массива обычно используются циклы for или while, где можно явно указать индексы элементов и их значения.

9. Как определяется базовый тип индексатора?

Базовый тип индексатора определяется возвращаемым типом значения элемента массива или коллекции, к которому происходит доступ с помощью индексатора.

10.Что записывается в качестве имени индексатора?

Имя индексатора записывается как this, за которым в квадратных скобках указываются параметры индексации. Например, public int this[int index].

11.Что содержит список параметров индексатора?

Список параметров индексатора содержит параметры, по которым происходит индексация элементов. В случае одномерного массива параметром может быть один целочисленный индекс, а для многомерного массива список параметров будет содержать несколько индексов для доступа к элементу по соответствующим координатам.

**Лабораторная работа №5**

**Задание 1.** Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Каждый разрабатываемый класс должен, содержать следующие элементы: скрытые и открытые поля, конструкторы (один из них должен передавать параметром массив), перегруженные операции.

В программе должна выполняться проверка всех разработанных элементов класса.

9. Описать класс для работы с одномерным массивом вещественных чисел. Обеспечить следующие возможности: нахождение суммы элементов массива (перегрузка операции +).

using System;

namespace ArrayOperations

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите количество элементов массива: ");

int size = int.Parse(Console.ReadLine());

// Создаем массив вещественных чисел

double[] myArray = new double[size];

// Вводим значения для массива

for (int i = 0; i < size; i++)

{

Console.Write($"Введите элемент {i + 1}: ");

myArray[i] = double.Parse(Console.ReadLine());

}

// Создаем объект класса для работы с массивом

ArrayCalculator calculator = new ArrayCalculator(myArray);

// Вычисляем сумму элементов массива

double sum = calculator.CalculateSum();

Console.WriteLine($"Сумма элементов массива: {sum}");

}

}

class ArrayCalculator

{

private double[] array;

// Конструктор, принимающий массив в качестве параметра

public ArrayCalculator(double[] inputArray)

{

array = inputArray;

}

// Перегрузка операции сложения для массивов

public static double operator +(ArrayCalculator calc, double value)

{

double sum = calc.CalculateSum();

return sum + value;

}

// Метод для вычисления суммы элементов массива

public double CalculateSum()

{

double sum = 0;

foreach (double element in array)

{

sum += element;

}

return sum;

}

}

}

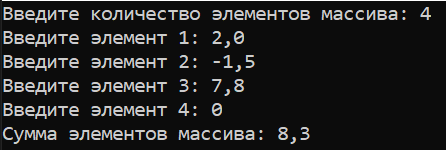


Рисунок 10 – Полученный результат

**Контрольные вопросы:**

1. Что представляет собой перегрузка методов?

Перегрузка методов представляет собой возможность создания нескольких методов с одним и тем же именем, но разной сигнатурой. Сигнатура метода включает в себя имя метода, количество параметров, типы параметров и их порядок. Например, можно иметь несколько версий метода Sum, каждая из которых принимает разное количество аргументов или аргументы разных типов.

2. Что представляет собой перегрузка операций

Перегрузка операций позволяет определить пользовательскую реализацию стандартных операторов (например, сложение, вычитание, равенство) для пользовательских типов данных. Это делается с помощью ключевого слова operator. Например, можно перегрузить операторы +, -, \*, / для пользовательской структуры, представляющей рациональное число.

3. Формат описания операции класса.

Формат описания операции класса включает в себя объявление класса с ключевым словом class, за которым следует уникальный идентификатор. Внутри класса можно определить поля, свойства, методы и события.

4. Какие операции нельзя перегружать?

Операции, которые нельзя перегружать, включают условные логические операторы (например, &&, ||) и операторы присваивания (например, +=, -=).

5. Что является результатом перегрузки унарных операций?

Результатом перегрузки унарных операций может быть изменение состояния объекта или возврат нового объекта с измененными значениями.

6. Какие параметры могут быть у бинарных операций класса?

Параметры бинарных операций класса могут быть любого типа, но хотя бы один из параметров должен иметь тип, который содержит объявление оператора.

7. Как выполняется перегрузка операций отношения?

Перегрузка операций отношения позволяет определить пользовательское поведение для операторов сравнения (например, ==, <, >).

8. Чем являются строки в С#?

Строки в С# являются ссылочными типами данных и представляют последовательности символов.

9. Какие операции определены для строк?

Для строк определены операции конкатенации (+), сравнения (==, !=), доступа к символам и другие.

10.Как создаются строки?

Строки создаются с помощью ключевого слова new и конструктора string.

11.Можноли изменять значение строки?

Значение строки нельзя изменять, так как строки в C# являются неизменяемыми (immutable).

**Лабораторная работа №6**

**Наследование**

**Задание 1.** Составить программу с одним родительским классом и потомком. Все поля должны быть закрытыми. Базовый класс должен содержать конструкторы с параметрами, методы доступа к закрытым полям, вывод полей и указанный в таблице метод. Производный класс содержит дополнения и изменения, организовать вывод новых полей потомка, при этом имена методов совпадают с именами методов базового класса. Составить тестирующую программу с выдачей результатов. Создать объекты базового и производного типов. В программе должна выполняться проверка всех разработанных элементов класса.

9. Базовый класс: Студент (поле: имя, средний балл s)

Метод: Стипендия 300000+10000([s]-5)

Потомок: Магистр (поле – специальность)

Изменения в потомках: Увеличить стипендию на m руб.

using System;

// Базовый класс (родительский класс)

class Student

{

// Закрытые поля

private string name;

private double averageGrade;

// Конструктор с параметрами

public Student(string name, double averageGrade)

{

this.name = name;

this.averageGrade = averageGrade;

}

// Метод доступа к закрытым полям

public string GetName()

{

return name;

}

public double GetAverageGrade()

{

return averageGrade;

}

// Метод для расчета стипендии

public double CalculateScholarship()

{

return 300000 + 10000 \* (averageGrade - 5);

}

}

// Производный класс (потомок)

class Master : Student

{

// Дополнительное поле для специальности магистра

private string specialty;

// Конструктор с параметрами

public Master(string name, double averageGrade, string specialty)

: base(name, averageGrade)

{

this.specialty = specialty;

}

// Метод для увеличения стипендии на m рублей

public double IncreaseScholarship(double m)

{

return CalculateScholarship() + m;

}

// Метод для вывода информации о магистре

public void DisplayMasterInfo()

{

Console.WriteLine($"Магистр {GetName()}, специальность: {specialty}");

Console.WriteLine($"Стипендия: {CalculateScholarship()} рублей");

}

}

class Program

{

static void Main()

{

// Создаем объект базового класса (Студент)

Student student = new Student("Иван", 4.8);

// Создаем объект производного класса (Магистр)

Master master = new Master("Анна", 5.2, "Информационные технологии");

// Выводим информацию о студенте и магистре

Console.WriteLine("Информация о студенте:");

Console.WriteLine($"Имя: {student.GetName()}, Средний балл: {student.GetAverageGrade()}");

Console.WriteLine($"Стипендия: {student.CalculateScholarship()} рублей");

Console.WriteLine("\nИнформация о магистре:");

master.DisplayMasterInfo();

// Увеличиваем стипендию магистра на 5000 рублей

double increasedScholarship = master.IncreaseScholarship(5000);

Console.WriteLine($"\nУвеличенная стипендия магистра: {increasedScholarship} рублей");

}

}

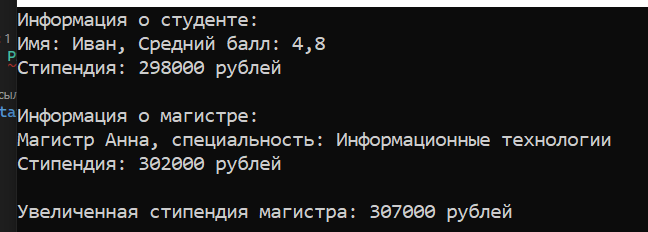


Рисунок 11 – Полученный результат

**Контрольные вопросы:**

1. В чем состоит принцип наследования?

Принцип наследования заключается в том, что класс может наследовать свойства и методы от другого класса, который называется базовым классом или родительским классом. Это позволяет создавать иерархию классов, где дочерние классы наследуют общие характеристики от родительских.

2. Какие члены класса наследуются?

Члены класса, которые наследуются, включают в себя:

* Поля (переменные): Дочерний класс может наследовать поля из родительского класса.
* Методы: Дочерний класс может наследовать методы, определенные в родительском классе.
* Свойства: Свойства (getters и setters) также наследуются.

3. Что представляет собой защищенный доступ?

Защищенный доступ (protected access) представляет собой уровень доступа, при котором члены класса видны только внутри класса и его дочерних классов. Он обозначается символом \_ перед именем члена (например, \_protectedField).

4. Как происходит вызов конструкторов базового класса?

Вызов конструкторов базового класса происходит с помощью ключевого слова super. В дочернем классе мы можем вызвать конструктор родительского класса, чтобы выполнить инициализацию общих полей.

5. Что такое сокрытие имен при наследовании?

Сокрытие имен при наследовании (name hiding) возникает, когда дочерний класс определяет член с тем же именем, что и в родительском классе. В этом случае член родительского класса становится недоступным в дочернем классе.

6. Как получить доступ к сокрытому члену базового класса?

Для доступа к сокрытым членам базового класса можно использовать ключевое слово base. Например, base.MethodName() вызовет метод из родительского класса.

**Лабораторная работа №7**

**Полиморфизм. Виртуальные методы**

**Задание 1.**

Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с указанием названий и полей и их значений соответствующего объекта. Составить тестирующую программу с выдачей протокола на экран монитора.

При этом создать объекты базового и производных типов, используя полиморфный контейнер - массив ссылок базового класса на объекты базового и производных классов (количество объектов >=5).

|  |  |  |
| --- | --- | --- |
| Родительский класс | Потомки | Полиморфные методы |
| Вектор (поле-название) | Двумерный вектор (поля: компоненты вектора)  Трёхмерный вектор(поле размерность) | Длина вектора |

using System;

// Базовый класс (родительский класс)

class Vector

{

private string name; // Название вектора

// Конструктор с параметрами

public Vector(string name)

{

this.name = name;

}

// Виртуальный метод для расчета длины вектора

public virtual double CalculateLength()

{

return 0; // Базовый вектор имеет нулевую длину

}

// Метод для вывода информации о векторе

public void DisplayInfo()

{

Console.WriteLine($"Вектор \"{name}\":");

Console.WriteLine($"Длина вектора: {CalculateLength()}");

}

}

// Производный класс (потомок) для двумерных векторов

class TwoDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

public TwoDimensionalVector(string name, double x, double y)

: base(name)

{

this.x = x;

this.y = y;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y);

}

}

// Производный класс (потомок) для трехмерных векторов

class ThreeDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

private double z; // Компонента z

public ThreeDimensionalVector(string name, double x, double y, double z)

: base(name)

{

this.x = x;

this.y = y;

this.z = z;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y + z \* z);

}

}

class Program

{

static void Main()

{

// Создаем объекты базового и производных классов

Vector baseVector = new Vector("Базовый вектор");

TwoDimensionalVector twoDVector = new TwoDimensionalVector("2D вектор", 3, 4);

ThreeDimensionalVector threeDVector = new ThreeDimensionalVector("3D вектор", 1, 2, 3);

// Создаем массив ссылок на базовый класс

Vector[] vectors = { baseVector, twoDVector, threeDVector };

// Выводим информацию о векторах

foreach (var vector in vectors)

{

vector.DisplayInfo();

}

}

}

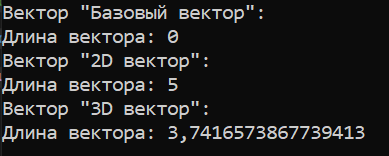


Рисунок 12 – Полученный результат

**Задание 2.** Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 1. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов >=5).

using System;

// Базовый класс (родительский класс)

class Vector

{

protected string name; // Название вектора

// Конструктор с параметрами

public Vector(string name)

{

this.name = name;

}

// Виртуальный метод для расчета длины вектора

public virtual double CalculateLength()

{

return 0; // Базовый вектор имеет нулевую длину

}

// Метод для вывода информации о векторе

public void DisplayInfo()

{

Console.WriteLine($"Вектор \"{name}\":");

Console.WriteLine($"Длина вектора: {CalculateLength()}");

}

}

// Производный класс (потомок) для двумерных векторов

class TwoDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

public TwoDimensionalVector(string name, double x, double y)

: base(name)

{

this.x = x;

this.y = y;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y);

}

}

// Производный класс (потомок) для трехмерных векторов

class ThreeDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

private double z; // Компонента z

public ThreeDimensionalVector(string name, double x, double y, double z)

: base(name)

{

this.x = x;

this.y = y;

this.z = z;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y + z \* z);

}

}

class Program

{

static void Main()

{

Vector[] vectors = new Vector[5]; // Создаем массив объектов базового класса

vectors[0] = new TwoDimensionalVector("2D вектор 1", 3, 4);

vectors[1] = new TwoDimensionalVector("2D вектор 2", 1, 7);

vectors[2] = new ThreeDimensionalVector("3D вектор 1", 1, 2, 3);

vectors[3] = new ThreeDimensionalVector("3D вектор 2", 4, 5, 6);

vectors[4] = new TwoDimensionalVector("2D вектор 3", 8, 2);

// Выводим информацию о векторах

foreach (var vector in vectors)

{

vector.DisplayInfo();

}

// Вычисляем суммарную длину векторов

double totalLength = 0;

foreach (var vector in vectors)

{

totalLength += vector.CalculateLength();

}

Console.WriteLine($"Суммарная длина всех векторов: {totalLength}");

}

}

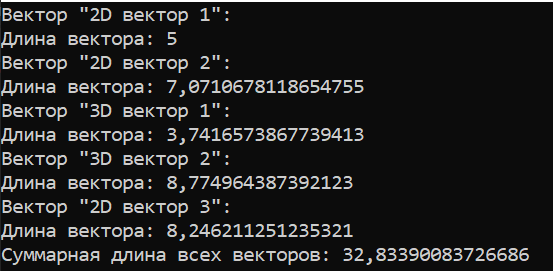


Рисунок 13 – Полученный результат

**Контрольные вопросы:**

1. Что означает принцип полиморфизма?

Принцип полиморфизма в объектно-ориентированном программировании позволяет разным сущностям выполнять одни и те же действия. Независимо от того, как эти сущности устроены внутри и чем они различаются, программист может использовать общий код для работы с ними. Примером полиморфизма может быть метод “Отправить”, который может применяться как к картинкам, так и к видео1.

2. Для чего используется позднее связывание?

Позднее связывание — это механизм, при котором вызов метода происходит на основе типа объекта во время выполнения программы. Если в иерархии классов есть цепочка виртуальных методов (с помощью ключевых слов virtual и override), то компилятор использует позднее связывание. Это позволяет реализовать полиморфизм2.

3. В каких случаях используются виртуальные методы?

Виртуальные методы — это методы класса, которые могут быть переопределены в классах-наследниках. Они позволяют обеспечить разную реализацию метода для разных типов объектов, не зависимо от точного типа ссылки на объект3.

4. Какие условия необходимо соблюдать при переопределении виртуального метода?

При переопределении виртуального метода необходимо соблюдать следующие условия:

* Метод в базовом классе должен быть объявлен как виртуальный с помощью ключевого слова virtual.
* Метод в производном классе должен быть объявлен с помощью ключевого слова override.
* Сигнатура (параметры и возвращаемый тип) переопределенного метода должна совпадать с сигнатурой базового метода.

5. Что представляют собой абстрактные классы? Для чего они предназначены?

Абстрактные классы — это базовые классы, которые не предполагают создания экземпляров. Они могут содержать абстрактные методы, которые должны быть реализованы в классах-наследниках. Абстрактные классы предоставляют общий функционал для классов-потомков4.

6. Могут ли в абстрактном классе быть неабстрактные методы?

В абстрактном классе могут быть как абстрактные, так и неабстрактные методы. Неабстрактные методы могут содержать реализацию, а абстрактные — только декларацию без реализации. Абстрактные классы предоставляют базовый функционал, который может быть переопределен в классах-наследниках

**Лабораторная работа № 8**

**Интерфейсы.**

**Задание 1.** Интерфейсы Ix, Iy, Iz, содержат объявления методов с

одной и той же сигнатурой следующим образом

interface Ix

{

void IxF0(параметр);

void IxF1();

}

interface Iy

{

void F0(параметр);

void F1();

}

interface Iz

{

void F0(параметр);

void F1();

}

Эти интерфейсы наследуются в классе TestClass, содержащий

член w типа параметр и реализуются так, как задано в варианте. В каж-

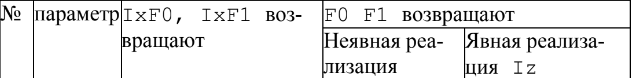
дом методе задать вывод результата.

Рассмотреть случай

* неявной реализации интерфейсов
* явной реализации интерфейса Iz

В программе должна выполняться:

* неявная неоднозначная реализация методов интерфейсов Iy и Iz,
* вызов функций с явным приведением к типу интерфейса,
* вызов метода для объекта посредством интерфейсной ссылки.





using System;

using System.Collections.Generic;

interface Ix

{

void IxF0(double param);

void IxF1();

}

interface Iy

{

void F0(double param);

void F1();

}

interface Iz

{

void F0(double param);

void F1();

}

class TestClass : Ix, Iy, Iz

{

private double w;

// Неявная реализация методов интерфейсов Ix, Iy и Iz

public void IxF0(double param)

{

Console.WriteLine($"Log(w) = {Math.Log(w)}");

}

public void IxF1()

{

Console.WriteLine($"2/w = {2 / w}");

}

void Iy.F0(double param)

{

Console.WriteLine($"2/w = {2 / w} (неоднозначная реализация)");

}

void Iy.F1()

{

Console.WriteLine("Вызван метод F1() интерфейса Iy");

}

// Явная реализация интерфейса Iz

void Iz.F0(double param)

{

Console.WriteLine($"w^3 = {Math.Pow(w, 3)}");

}

void Iz.F1()

{

Console.WriteLine("Вызван метод F1() интерфейса Iz");

}

public TestClass(double w)

{

this.w = w;

}

}

class Program

{

static void Main(string[] args)

{

double w = 5.0; // Произвольное значение параметра w

TestClass testObject = new TestClass(w);

// Вызов методов через интерфейсные ссылки

Ix ix = testObject;

Iy iy = testObject;

Iz iz = testObject;

ix.IxF0(w);

ix.IxF1();

iy.F0(w);

iy.F1();

iz.F0(w);

iz.F1();

}

}

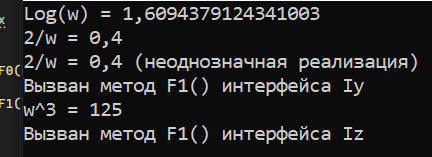


Рисунок 14 – Полученный результат

**Задание 2.** Выполнить задания, используя для хранения экземпляров разработанных классов стандартные параметризованные коллекции. Во всех классах реализовать интерфейсы IComparable и IComparer перегрузить операции отношения для реализации сравнения объектов по указанному полю. Результат вывести на экран.

9. Составить список больных отделения, включив следующие данные: ФИО, болезнь, дата поступления, рабочий стаж. Вывести в новый список информацию о больных, находящихся на лечении больше недели, отсортировав их по ФИО.

using System;

using System.Collections.Generic;

// Класс, представляющий информацию о больном

class Patient : IComparable<Patient>

{

public string FullName { get; set; }

public string Disease { get; set; }

public DateTime AdmissionDate { get; set; }

public int WorkExperience { get; set; }

// Реализация интерфейса IComparable для сравнения по ФИО

public int CompareTo(Patient other)

{

return string.Compare(FullName, other.FullName, StringComparison.OrdinalIgnoreCase);

}

}

class Program

{

static void Main()

{

// Создаем список больных

var patients = new List<Patient>

{

new Patient { FullName = "Иванов Иван Иванович", Disease = "Грипп", AdmissionDate = DateTime.Parse("2024-02-10"), WorkExperience = 5 },

new Patient { FullName = "Петрова Ольга Сергеевна", Disease = "Ангина", AdmissionDate = DateTime.Parse("2024-02-15"), WorkExperience = 8 },

new Patient { FullName = "Сидоров Павел Николаевич", Disease = "Простуда", AdmissionDate = DateTime.Parse("2024-02-12"), WorkExperience = 3 },

// Добавьте еще больных по аналогии

};

// Фильтруем больных, находящихся на лечении больше недели

var hospitalizedMoreThanWeek = patients.FindAll(p => (DateTime.Now - p.AdmissionDate).TotalDays > 7);

// Сортируем по ФИО

hospitalizedMoreThanWeek.Sort();

// Выводим информацию о больных

Console.WriteLine("Список больных, находящихся на лечении больше недели (по ФИО):");

foreach (var patient in hospitalizedMoreThanWeek)

{

Console.WriteLine($"{patient.FullName}, болезнь: {patient.Disease}, дата поступления: {patient.AdmissionDate.ToShortDateString()}, стаж: {patient.WorkExperience} лет");

}

}

}

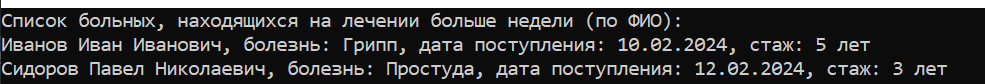


Рисунок 15 – Полученный результат

**Контрольные вопросы:**

1. Как описывается интерфейс? Его назначение.

Интерфейс в C# содержит определения для группы связанных функциональных возможностей, которые должны реализовать неабстрактный класс или структура. Он позволяет определить контракт, который классы и структуры обязаны соблюдать.

2. Какие члены может содержать интерфейс?

Назначение интерфейса: Интерфейс позволяет включить в класс поведение из нескольких источников. Это важно, так как C# не поддерживает множественное наследование классов. Также интерфейсы используются для имитации наследования для структур, которые не могут фактически наследовать от другой структуры или класса.

3. Какие спецификаторы допустимы у методов, реализующих интерфейс?

Интерфейсы могут содержать методы экземпляра, свойства, события, индексаторы, а также любое сочетание этих четырех типов членов. Они также могут содержать статические конструкторы, поля, константы или операторы1.

4. В каких случаях используется явная реализация интерфейса?

Методы, реализующие интерфейс, могут иметь любые допустимые спецификаторы доступа, такие как public, protected, internal, private, protected internal или private protected1.

5. Как осуществляется наследование интерфейсов?

Явная реализация интерфейса используется, когда необходимо разрешить конфликт имен методов, которые реализуют один и тот же интерфейс1.

6. Можно ли явно реализованные методы объявлять виртуальными?

Класс или структура может реализовывать несколько интерфейсов, но наследовать только от одного класса.

7. Можно ли повторно реализовать интерфейс, указав его имя в списке предков класса наряду с классом-предком?

Нет, явно реализованные методы нельзя объявлять виртуальными.

8. Какие стандартные интерфейсы используются для работы с коллек-циями?

Да, можно.

9. Чем отличаются интерфейсы IComparable и IComparer?

Для работы с коллекциями используются интерфейсы IEnumerable, ICollection, IList, IDictionary и другие.

10. Чем отличаются интерфейсы IComparable и IComparer?

IComparable используется для сравнения объектов одного типа, а IComparer позволяет определить пользовательское сравнение для разных типов1.