**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.

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

8. Для выполнения программы: Debug ► Start Without Debugging

using System;

namespace matemat1

{

class Program

{

public static void Main(string[] args)

{

// Объявление переменной x и присвоение ей значения 12

int x = 12;

// Объявление переменной y и присвоение ей значения 3

double y = 3;

// Вывод значения переменной x на консоль

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

// Объявление переменной b и присвоение ей значения 9

double b = 9;

// Вывод значения переменной y на консоль

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

}

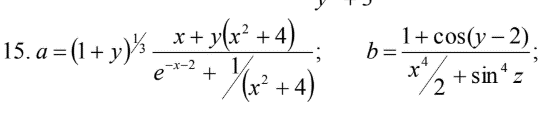
}

}



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.



using System;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите вещественное число x:");

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

Console.WriteLine("Введите вещественное число y:");

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

Console.WriteLine("Введите вещественное число z:");

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

double a = Math.Pow(1 + y, 1.0 / 3) \* (x + y \* (x \* x + 4)) /

(Math.Exp(-x \* x) + 1.0 / (x \* x + 4));

double b = (1 + Math.Cos(y - 2)) / (Math.Pow(x, 4) / 2 + Math.Pow(Math.Sin(z), 4));

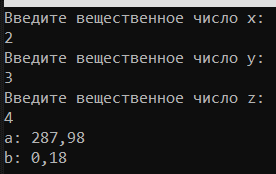
Console.WriteLine("a: {0:F2}", a);

Console.WriteLine("b: {0:F2}", b);

}

}

}



Контрольные вопросы:

1. Основные принципы технологии .NET.

• Многоязычность: .NET поддерживает несколько языков программирования, таких как C#, VB.NET, F#, и другие, что позволяет разработчикам выбирать наиболее подходящий язык для конкретной задачи.

• Управляемый код: .NET использует Common Language Runtime (CLR) для управления выполнением программ, обеспечивая безопасность, управление памятью и другие важные функции.

• Библиотека классов: .NET Framework предоставляет обширную библиотеку классов, которая содержит множество готовых компонентов для разработки приложений.

• Переносимость: Приложения, разработанные на платформе .NET, могут быть запущены на различных устройствах и операционных системах.

2. Что представляет собой платформа Visual Studio.NET?

Платформа Visual Studio .NET представляет собой интегрированную среду разработки (IDE) для создания приложений под управлением технологии .NET. Visual Studio предоставляет разработчикам широкий набор инструментов для написания, отладки, тестирования и развертывания приложений, а также интегрированные средства для работы с различными языками программирования, включая C#, VB.NET, F# и другие.

3. Как создать консольное приложение?

Для создания консольного приложения в Visual Studio .NET можно выполнить следующие шаги:

\* Открыть Visual Studio и создать новый проект.

\* Выбрать тип проекта "Console Application" (Консольное приложение) из доступных шаблонов.

\* Задать имя и местоположение проекта, а затем нажать "Create" (Создать).

\* После этого будет создано консольное приложение, где можно писать код для выполнения различных операций в командной строке.

4. Принципы объектно-ориентированного программирования.

• Инкапсуляция: скрытие внутренних данных объекта от внешнего доступа и предоставление доступа к ним только через методы объекта.

• Наследование: возможность создания новых классов на основе существующих, что позволяет повторно использовать код и создавать иерархии классов.

• Полиморфизм: способность объектов разных классов отвечать на одинаковые запросы с различными действиями, что обеспечивает гибкость и расширяемость программ.

5. Литералы. Как определяются типы литералов?

Литералы - это постоянные значения, встроенные непосредственно в текст программы. Типы литералов определяются на основе их формата и значения. Например, целочисленные литералы могут быть определены как целые числа, литералы с плавающей запятой - как числа с плавающей запятой, логические литералы - как значения true или false, и т. д.

6. Какие типы относятся к встроенным?

К встроенным типам относятся:

\* Целочисленные типы (int, long, short, byte и другие).

\* Типы с плавающей запятой (float, double).

\* Символьный тип (char).

\* Логический тип (bool).

\* Строковый тип (string).

7. Чем отличаются типы-значения и ссылочные типы?

Типы-значения (value types) хранят свои данные в стеке и содержат само значение, в то время как ссылочные типы (reference types) хранят ссылку на данные в куче. При работе с типами-значения происходит копирование значений, в то время как при работе с ссылочными типами копируется только ссылка на данные. Типы-значения в C# включают целочисленные типы, типы с плавающей запятой, логический тип, структуры и перечисления, в то время как классы, интерфейсы, делегаты и массивы относятся к ссылочным типам.

8. Какие типы числовых переменных имеются?

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

• Типы с плавающей запятой: float, double, decimal.

• Логический тип: bool.

• Символьный тип: char.

9. Что такое объявление и инициализация?

Объявление переменной - это процесс создания переменной с указанием её типа и имени. Инициализация переменной - это присвоение начального значения переменной в момент её создания или позже в программе.

10.Для чего используется упаковка и распаковка?

Упаковка (boxing) в C# используется для преобразования значимых типов (типов-значений) в ссылочные типы (object или другие классы). Распаковка (unboxing) - обратная операция, при которой значение упакованного объекта извлекается и преобразуется обратно в значимый тип.

11.Как в С# выполняется преобразование типа?

Преобразование типа в C# может быть явным (explicit) или неявным (implicit). Неявное преобразование выполняется автоматически компилятором, если нет потери данных, например, преобразование от int к long. Явное преобразование требует явного указания типа, например, при преобразовании от double к int.

12.Как осуществляется консольный ввод?

Для консольного ввода в C# можно использовать методы класса Console:

• Для чтения строки: Console.ReadLine().

• Для чтения символа: Console.ReadKey().

• Для чтения числа: считать строку с помощью Console.ReadLine() и затем преобразовать в нужный тип данных, например, с помощью метода int.Parse() для целых чисел.

13.Чем отличаются методы Read и ReadLine?

Методы Read() и ReadLine() в C# отличаются следующим образом:

• Read() считывает следующий символ из стандартного ввода (обычно консоли) и возвращает его в виде целочисленного значения (код символа).

• ReadLine() считывает следующую строку текста из стандартного ввода и возвращает её в виде строки.

14.Как обеспечить вывод данных на экран?

Для вывода данных на экран в C# можно использовать методы класса Console, например:

• Console.Write() для вывода текста без перехода на новую строку.

• Console.WriteLine() для вывода текста с переходом на новую строку.

15.Для чего предназначен и как используется форматный вывод данных?

Форматный вывод данных в C# предназначен для управления способом отображения данных при выводе. Для этого используется метод Console.WriteLine() с использованием специальных форматирующих строк, например:

• {0} - для подстановки первого аргумента.

• {1} - для подстановки второго аргумента.

• {0:N2} - для вывода числа с двумя знаками после запятой и т.д.

16.Каковы основные правила использования стандартных функций?

Основные правила использования стандартных функций включают:

• Понимание назначения и синтаксиса функции.

• Правильное указание аргументов функции в соответствии с её требованиями.

• Обработка возвращаемых значений функции, если они есть.

• Учет возможных исключений или ошибок, которые может генерировать функция.

17.Основные приемы работы в среде разработки Visual Studio.NET:

- Как создать консольное приложение?

- Как сохранить проект с заданным именем?

- Как загрузить проект?

- Как выполнить отладку программы?

- Как откомпилировать и выполнить программу?

- Как просмотреть результаты выполнения программы?

• Для создания консольного приложения:

1. Откройте Visual Studio.

2. Выберите "File" -> "New" -> "Project".

3. В окне "New Project" выберите "Console App (.NET Core)".

4. Укажите имя проекта и нажмите "Create".

• Для сохранения проекта с заданным именем:

1. Выберите "File" -> "Save All".

2. Укажите путь и имя файла проекта.

• Для загрузки проекта:

1. Выберите "File" -> "Open" -> "Project/Solution".

2. Укажите путь к файлу проекта и нажмите "Open".

• Для выполнения отладки программы:

1. Установите точки останова (breakpoints) в нужных местах кода.

2. Нажмите "Start Debugging" (F5) или выберите "Debug" -> "Start Debugging".

• Для компиляции и выполнения программы:

1. Нажмите "Start" (зеленый треугольник) или выберите "Debug" -> "Start Without Debugging" (Ctrl + F5).

• Для просмотра результатов выполнения программы:

1. Результаты вывода на консоль можно увидеть в окне Output.

2. Если программа записывает результаты в файл, вы можете открыть этот файл для просмотра.

**Лабораторная работа №2**

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

3. Напишите программу для определения, какому алфавиту (латинскому или русскому) принадлежит введенный с клавиатуры символ. На экран вывести символ и комментарий. Предусмотреть возможность ввода неверного символа.

using System;

class Program

{

static void Main(string[] args)

{

Console.Write("Введите символ: ");

char inputChar = Console.ReadKey().KeyChar; // Считываем символ с клавиатуры

Console.WriteLine(); // Переход на новую строку

// Проверяем, к какому алфавиту принадлежит символ

if ((inputChar >= 'A' && inputChar <= 'Z') || (inputChar >= 'a' && inputChar <= 'z'))

{

Console.WriteLine($"Символ '{inputChar}' принадлежит латинскому алфавиту.");

}

else if ((inputChar >= 'А' && inputChar <= 'Я') || (inputChar >= 'а' && inputChar <= 'я'))

{

Console.WriteLine($"Символ '{inputChar}' принадлежит русскому алфавиту.");

}

else

{

Console.WriteLine($"Символ '{inputChar}' не принадлежит ни латинскому, ни русскому алфавиту.");

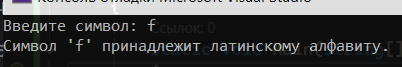
}

}

}







Задание 2. Напишите функции в виде методов. Напишите тестирующую

программу с выдачей результатов на экран.

3. В последовательности чисел вводимых с клавиатуры исключить все

цифры 1 и 3, оставив прежним порядок оставшихся цифр.

using System;

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите последовательность чисел без пробелов: ");

string input = Console.ReadLine(); // Считываем строку с числовой последовательностью

string result = ExcludeDigits(input); // Вызываем метод для исключения цифр

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

}

// Метод для исключения цифр 1 и 3 из строки

static string ExcludeDigits(string input)

{

string output = ""; // Переменная для хранения результата

foreach (char digit in input) // Проходим по каждому символу в строке

{

if (digit != '1' && digit != '3') // Проверяем, что символ не равен '1' и '3'

{

output += digit; // Добавляем символ к результату

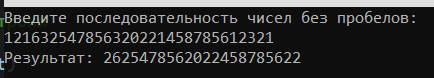
}

}

return output; // Возвращаем результат

}

}



Контрольные вопросы:

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

Условие в операторе if является логическим выражением. Оно может принимать значения true или false.

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

Если отсутствует часть else в операторе if, то в случае, если условие не истинно (false), выполнение программы просто продолжится с следующей инструкции после блока if.

3. В каких случаях используется оператор switch?

Оператор switch используется для выбора одного из множества вариантов на основе значения переменной или выражения.

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

В качестве выражения в операторе switch может быть целое число, символ, строка или перечисление.

5. В каком случае выполняется последовательность инструкций default ветви?

Последовательность инструкций default-ветви выполняется, если ни одно из условий case не совпало с значением выражения.

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

Основное отличие между операторами while и do ... while в том, что в while условие проверяется перед выполнением тела цикла, а в do ... while - после его выполнения.

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

Элемент инициализации в операторе for представляет собой код, который выполняется перед тем, как начнется выполнение цикла. Обычно он используется для инициализации счетчиков.

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

Элемент условия в цикле for может быть логическим выражением, которое будет оцениваться перед каждой итерацией цикла.

9.Назначение управляющей переменной цикла for?

Управляющая переменная цикла for используется для отслеживания текущего состояния выполнения цикла, чаще всего для подсчета итераций.

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

- goto - перенаправляет выполнение к указанной метке.

- break - завершает выполнение текущего цикла или переключателя.

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

- return - завершает выполнение метода и может вернуть значение, если это предусмотрено.

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

Циклические алгоритмы с явно заданным числом повторений реализуются с помощью цикла for, где количество итераций задается в условии.

12.Что представляют собой методы?

Методы представляют собой группы связанных инструкций, которые выполняют определенную задачу.

13.Как объявляется метод?

Метод объявляется с указанием типа возвращаемого значения, имени метода и при необходимости параметров в круглых скобках, например: public int MyMethod(int parameter).

14.Какова область действия параметров метода?

Область действия параметров метода ограничена телом метода; они существуют только в рамках этого метода.

15.Как вызываются методы?

Методы вызываются по их имени, с указанием необходимых аргументов в круглых скобках, например: MyMethod(5).

16.Общие (статические) методы класса.

Общие (статические) методы класса могут быть вызваны без создания экземпляра класса. Они объявляются с использованием ключевого слова static, например: public static void MyStaticMethod().

**Лабораторная работа №3**

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.

3. Составить описание класса для вектора, заданного его координатами в

трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора, вычисления скалярного произведения двух векторов, длины вектора.

Задание 2. Включите в проект Задания 1 обработку исключений.

using System;

public class Vector

{

// Свойство для вычисления длины вектора

public double Length { get => Math.Sqrt(x \* x + y \* y + z \* z); }

// поля для координат вектора

private double x, y, z;

public Vector() { }

// Конструктор с параметрами для установки значений координат

public Vector(double x, double y, double z)

{

this.x = x;

this.y = y;

this.z = z;

}

// Оператор сложения векторов

public static Vector operator +(Vector a, Vector b) => new Vector(a.x + b.x, a.y + b.y, a.z + b.z);

// Оператор вычитания векторов

public static Vector operator -(Vector a, Vector b) => new Vector(a.x - b.x, a.y - b.y, a.z - b.z);

// Оператор скалярного произведения векторов

public static double operator \*(Vector a, Vector b) => a.x \* b.x + a.y \* b.y + a.z \* b.z;

// Переопределение метода для отображения вектора в виде строки

public override string ToString() => $"({x}, {y}, {z})";

}

class Program

{

static void Main()

{

// Чтение координат для первого вектора

Vector vector1 = new Vector(ReadDouble(), ReadDouble(), ReadDouble());

// Чтение координат для второго вектора

Vector vector2 = new Vector(ReadDouble(), ReadDouble(), ReadDouble());

// Вычисление суммы векторов

Vector sum = vector1 + vector2;

// Вычисление разности векторов

Vector difference = vector1 - vector2;

// Вычисление скалярного произведения векторов

double dotProduct = vector1 \* vector2;

// Вывод векторов и их атрибутов на экран

Console.WriteLine($"Vector 1: {vector1}");

Console.WriteLine($"Vector 2: {vector2}");

Console.WriteLine($"Length of Vector 1: {vector1.Length}");

Console.WriteLine($"Length of Vector 2: {vector2.Length}");

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

Console.WriteLine($"Difference: {difference}");

Console.WriteLine($"Dot Product: {dotProduct}");

// Создание нулевого вектора и вывод его на экран

Vector vector = new Vector();

Console.WriteLine($"zero vector: {vector}");

// Создание двух пустых векторов

Vector v1 = new Vector();

Vector v2 = new Vector();

// Сложение двух пустых векторов

Vector v3 = v1 + v2;

// Ожидание ввода пользователя перед завершением программы

Console.ReadLine();

}

// Метод для чтения вещественных чисел из консоли с обработкой ошибок

private static double ReadDouble()

{

while (true)

{

try

{

// Чтение и парсинг ввода пользователя

return double.Parse(Console.ReadLine());

}

catch (Exception)

{

// Сообщение об ошибке, если ввод некорректен

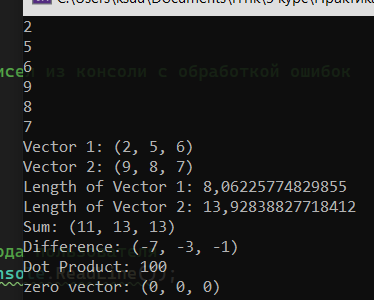
Console.WriteLine("Введены некорректные данные, введите вещественное число");

}

}

}

}



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.

3. Описать класс для работы с одномерным массивом целых чисел. Обеспечить возможность нахождения суммы элементов, стоящих после введенного с клавиатуры значения и вывода элементов, стоящих после введенного с клавиатуры значения.

using System;

using System.Collections.Generic;

using System.Linq;

internal class Program

{

static void Main(string[] args)

{

// Создаем экземпляр класса ArrayAdder

ArrayAdder a = new ArrayAdder();

// Выводим текущее состояние массива

a.ShowArrayState();

// Запрашиваем элемент для вывода и суммирования элементов после него

Console.WriteLine("Введите элемент для вывода и суммирования элементов после него");

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

// Показываем элементы после введенного элемента

a.ShowElementsAfter(el);

// Выводим сумму элементов после введенного элемента

Console.WriteLine($"Сумма последовательности элементов: {a.SumElementsAfter(el)}");

Console.ReadLine();

}

}

class ArrayAdder

{

private int[] arr; // Массив для хранения уникальных целых чисел

// Индексатор для доступа к элементам массива

public int this[int i]

{

get

{

// Проверяем, не выходит ли индекс за границы массива

if (arr.Length <= i || i < 0)

throw new Exception("Данное число выходит за границы индексов массива");

return arr[i];

}

set

{

// Проверяем, если значение уже существует в массиве

if (arr.Contains(value))

throw new Exception("Данное значение уже имеется");

arr[i] = value; // Устанавливаем значение по индексу

}

}

// Конструктор без параметров, инициализирует массив случайными уникальными числами от 1 до 10

public ArrayAdder()

{

Random rnd = new Random();

arr = new int[10]; // Инициализация массива размером 10

HashSet<int> set = new HashSet<int>();

// Генерируем уникальные случайные числа

while (set.Count < 10)

set.Add(rnd.Next(1, 11));

arr = set.ToArray(); // Преобразуем HashSet в массив

}

// Конструктор с параметром для создания массива заданного размера

public ArrayAdder(int n)

{

arr = new int[n];

}

// Метод для вывода элемента по индексу

public void ShowElement(int i) => Console.WriteLine(this[i]);

// Метод для отображения текущего состояния массива

public void ShowArrayState() => Console.WriteLine(string.Join(" ", arr));

// Метод для вывода элементов массива, находящихся после указанного элемента

public void ShowElementsAfter(int el)

{

string res = "";

int i = GetElementIndex(el) + 1; // Получаем индекс элемента и переходим к следующему

for (int j = i; j < arr.Length; j++)

res += $"{arr[j]} "; // Собираем элементы в строку

Console.WriteLine(res); // Выводим результат

}

// Метод для суммирования элементов массива, находящихся после указанного элемента

public int SumElementsAfter(int el)

{

int i = GetElementIndex(el) + 1; // Получаем индекс элемента и переходим к следующему

int sum = 0;

for (int j = i; j < arr.Length; j++)

sum += arr[j]; // Суммируем элементы

return sum; // Возвращаем сумму

}

// Метод для получения индекса указанного элемента в массиве

private int GetElementIndex(int el)

{

for (int i = 0; i < arr.Length; i++)

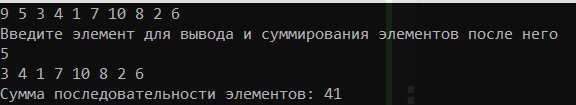
if (arr[i] == el)

return i; // Возвращаем индекс, если элемент найден

throw new Exception("Массив не имеет данного элемента"); // Исключение, если элемент не найден

}

}



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.

3. Описать класс для работы с одномерным массивом строк фиксированной длины. Обеспечить следующие возможности: сравнения массивов

на равенство ( перегрузку операции == для поэлементного сравнения строк).

using System;

internal class Program

{

static void Main(string[] args)

{

// Создаем две последовательности строк

StringSequence firstSeq = new StringSequence(new string[] { "hey", "bye" });

StringSequence secondSeq = new StringSequence(new string[] { "hello", "bye" });

// Сравниваем две последовательности и выводим результат

// Сравнение выполняется с использованием перегруженных операторов == и !=

Console.WriteLine($"{firstSeq == secondSeq} {firstSeq != secondSeq}");

// Присваиваем вторую последовательность первой

firstSeq.Seq = secondSeq.Seq;

// Сравниваем последовательности снова после изменения

Console.WriteLine($"{firstSeq == secondSeq} {firstSeq != secondSeq}");

// Ожидаем нажатия клавиши перед выходом

Console.ReadLine();

}

}

class StringSequence

{

private string[] arr; // Массив для хранения последовательности строк

// Свойство для доступа к массиву строк

public string[] Seq

{

get => arr; // Возвращаем массив строк

set => arr = value; // Устанавливаем новый массив строк

}

// Конструктор, принимающий массив строк в качестве параметра

public StringSequence(string[] arr)

{

this.arr = arr; // Инициализируем массив строк

}

// Пустой конструктор

public StringSequence() { }

// Перегрузка оператора == для сравнения двух объектов StringSequence

public static bool operator ==(StringSequence left, StringSequence right)

{

// Если один из объектов равен null, проверяем равен ли другой объект null

if (ReferenceEquals(left, right)) return true;

if (ReferenceEquals(left, null) || ReferenceEquals(right, null)) return false;

// Если длины массивов различаются, возвращаем false

if (left.arr.Length != right.arr.Length)

return false;

// Сравниваем элементы массивов поэлементно

for (int i = 0; i < left.arr.Length; i++)

if (left.arr[i] != right.arr[i])

return false; // Если хоть один элемент не равен, возвращаем false

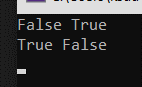
return true; // Все элементы равны, возвращаем true

}

// Перегрузка оператора != для сравнения двух объектов StringSequence

public static bool operator !=(StringSequence left, StringSequence right) => !(left == right);

}



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.

1. 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.

1. 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.

1. Какие операции нельзя перегружать?

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

1. Что является результатом перегрузки унарных операций?

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

1. 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.

1. Как выполняется перегрузка операций отношения?

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

1. Чем являются строки в С#?

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

1. Какие операции определены для строк?

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

1. Как создаются строки?

Строки создаются с помощью ключевого слова new и конструктора string.

1. Можно ли 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.

3. Базовый класс: Тетрадь (поле: название, количество листов –k)

Метод: Стоимость: 15 k

Потомок: Общая (поле – материал обложки)

Изменения в потомках: Изменить стоимость с учетом надбавки за обложку на 50р.

using System;

internal class Program

{

static void Main(string[] args)

{

// Создание экземпляров тетрадей

Notebook notebook1 = new Notebook("В клетку", 12); // Обычная тетрадь

CoveredNotebook notebook2 = new CoveredNotebook("В линейку", 24, "Картон"); // Тетрадь с обложкой

// Вывод информации о тетрадях

Console.WriteLine(notebook1);

Console.WriteLine(notebook2);

// Ожидание ввода пользователя перед завершением программы

Console.ReadLine();

}

}

class Notebook

{

private string name; // Название тетради

public string Name => name; // Свойство для получения названия тетради

private int k; // Количество листов

public int Sheets => k; // Свойство для получения количества листов

public virtual int Price => k \* 15; // Свойство для расчета цены тетради

// Конструктор для инициализации тетради с именем и количеством листов

public Notebook(string name, int k)

{

this.name = name;

this.k = k;

}

// Переопределение метода ToString для представления информации о тетради

public override string ToString() => $"Информация о тетради\n\tНазвание: {name}\n\tКол-во листов: {k}\n\tЦена: {Price}";

}

class CoveredNotebook : Notebook

{

// Переопределение цены для тетради с обложкой, добавляя 50 единиц к базовой цене

public override int Price => base.Price + 50;

private string coverMaterial; // Материал обложки

// Конструктор для инициализации тетради с обложкой

public CoveredNotebook(string name, int k, string material) : base(name, k)

{

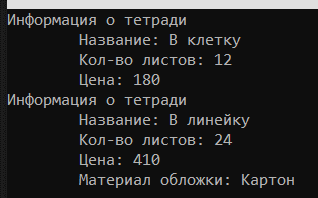
coverMaterial = material; // Присвоение материала обложки

}

// Переопределение метода ToString для добавления информации о материале обложки

public override string ToString() => base.ToString() + $"\n\tМатериал обложки: {coverMaterial}";

}



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.

1. 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.

1. 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).

1. 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.

1. 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.

1. 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).

Таблица 1 – Варианты заданий

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Родительский класс | Потомки | Полиморфные методы |
| 3 | Населенный  пункт  (поле название) | Село (поле количество домов-h, число жителей в доме, площадь села)  Город (поле количество  жителей- h, площадь города) | Плотность населения |

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).

3. Найти значение наименьшей плотности населения

using System;

using System.Linq;

internal class Program

{

static void Main(string[] args)

{

// Создаем массив объектов LivingPlace, включая города и деревни

LivingPlace[] livingPlaces = new LivingPlace[]

{

new City(2\_000\_000, 25000), // Город с 2 миллионами населения и площадью 25000

new Village(2, 1000, 5), // Деревня с 2 домами, площадью 1000 и средним числом людей в доме 5

new City(1\_000\_000, 10000), // Город с 1 миллион населения и площадью 10000

new Village(20, 10000, 7), // Деревня с 20 домами, площадью 10000 и 7 людей в каждом доме

new Village(5, 10, 100) // Деревня с 5 домами, площадью 10 и 100 людей в каждом доме

};

// Вывод плотности населения для каждого объекта LivingPlace

foreach (var lp in livingPlaces)

Console.WriteLine($"Плотность населения: {lp.GetPopulationDensity()}");

// Находим и выводим максимальную плотность населения среди всех объектов LivingPlace

Console.WriteLine(livingPlaces.Select(x => x.GetPopulationDensity()).Max());

// Ожидание ввода пользователя перед завершением программы

Console.ReadLine();

}

}

abstract class LivingPlace

{

protected int h; // Хранит количество домов или населения

protected double square; // Хранит площадь

// Конструктор для инициализации базовых данных

protected LivingPlace(int h, double square)

{

this.h = h;

this.square = square;

}

// Абстрактный метод для вычисления плотности населения, должен быть реализован в дочерних классах

public virtual double GetPopulationDensity() { throw new NotImplementedException(); }

}

class Village : LivingPlace

{

public int Homes => h; // Свойство для получения количества домов

public readonly double AveragePeopleInHouse; // Среднее число людей в доме

// Конструктор для инициализации деревни

public Village(int homesCount, double square, double averagePeopleInHouse) : base(homesCount, square)

{

AveragePeopleInHouse = averagePeopleInHouse;

}

// Переопределение метода для расчета плотности населения в деревне

public override double GetPopulationDensity() => (Homes \* AveragePeopleInHouse) / square;

}

class City : LivingPlace

{

public int Population => h; // Свойство для получения населения в городе

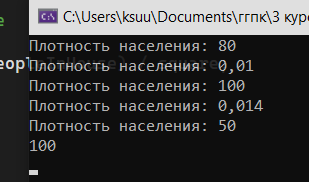
// Конструктор для инициализации города

public City(int population, double square) : base(population, square) { }

// Переопределение метода для расчета плотности населения в городе

public override double GetPopulationDensity() => Population / square;

}



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.

1. 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.

1. 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.

1. 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.

1. 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 каждом 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.

Таблица 2 – Варианты заданий

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Параметр | IxF0, IxF1 воз-  вращают | F0 F1 возвращают | |
| Неявная реализация | Явная реализация Iz |
| 3 | int | w+5 | w3 | 7w-2 |

using System;

internal class Program

{

static void Main(string[] args)

{

// Создаем экземпляр TestClass, который реализует интерфейсы Iy, Ix и Iz

Iy iy = new TestClass();

// Вызываем метод F0 из интерфейса Iy с параметром 2

iy.F0(2);

// Приводим iy к интерфейсу Iz и вызываем метод F1

((Iz)iy).F1();

// Ожидание ввода пользователя перед завершением программы

Console.ReadLine();

}

}

// Определение интерфейса Ix с двумя методами

interface Ix

{

void IxF0(int n); // Метод IxF0 принимает целочисленный параметр

void IxF1(); // Метод IxF1 без параметров

}

// Определение интерфейса Iy с двумя методами

interface Iy

{

void F0(int n); // Метод F0 принимает целочисленный параметр

void F1(); // Метод F1 без параметров

}

// Определение интерфейса Iz с двумя методами

interface Iz

{

void F0(int n); // Метод F0 принимает целочисленный параметр

void F1(); // Метод F1 без параметров

}

// Класс TestClass, реализующий три интерфейса: Ix, Iy и Iz

class TestClass : Ix, Iy, Iz

{

// Реализация метода F0 интерфейса Iz, выводит 7 \* w - 2

void Iz.F0(int w) => Console.WriteLine(7 \* w - 2);

// Реализация метода F1 интерфейса Iz, выводит -2

void Iz.F1() => Console.WriteLine(-2);

// Реализация метода F0 интерфейса Iy, выводит w в кубе

public void F0(int w) => Console.WriteLine(Math.Pow(w, 3));

// Реализация метода F1 интерфейса Iy, выводит 0

public void F1() => Console.WriteLine(0);

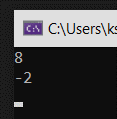
// Реализация метода IxF0, выводит w + 5

public void IxF0(int w) => Console.WriteLine(w + 5);

// Реализация метода IxF1, выводит 5

public void IxF1() => Console.WriteLine(5);

}



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.

3. Составить автомобильную ведомость, включив следующие данные: марка автомобиля, фамилия его владельца, год приобретения, пробег. Вывести в новый список информацию об автомобилях, выпущенных ранее определенного года, отсортировав их по пробегу.

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

// Класс Car реализует интерфейсы IComparable и IComparer для сравнения объектов автомобиля

public class Car : IComparable<Car>, IComparer<Car>

{

// Свойства автомобиля

public string Brand { get; set; } // Марка автомобиля

public string Owner { get; set; } // Владелец автомобиля

public int YearOfPurchase { get; set; } // Год приобретения

public int Mileage { get; set; } // Пробег автомобиля

// Конструктор для инициализации свойств автомобиля

public Car(string brand, string owner, int yearOfPurchase, int mileage)

{

Brand = brand;

Owner = owner;

YearOfPurchase = yearOfPurchase;

Mileage = mileage;

}

// Реализация метода CompareTo для сравнения по пробегу

public int CompareTo(Car other) => Mileage.CompareTo(other.Mileage);

// Реализация метода Compare из интерфейса IComparer

int IComparer<Car>.Compare(Car x, Car y)

{

// Сравниваем автомобили по пробегу с использованием обычной логики

if (x.Mileage < y.Mileage) return -1; // x меньше y

if (x.Mileage > y.Mileage) return 1; // x больше y

return 0; // x равен y

}

}

static void Main(string[] args)

{

// Создаем список автомобилей

List<Car> cars = new List<Car>

{

new Car("Toyota", "Иванов", 2010, 150000),

new Car("Ford", "Петров", 2005, 200000),

new Car("BMW", "Сидоров", 2015, 50000),

new Car("Audi", "Кузнецов", 2000, 300000),

new Car("Nissan", "Смирнов", 2008, 120000)

};

// Выводим информацию о всех машинах

Console.WriteLine("Машины");

foreach (var car in cars)

{

Console.WriteLine($"Марка: {car.Brand}, Владелец: {car.Owner}, Год приобретения: {car.YearOfPurchase}, Пробег: {car.Mileage}");

}

// Считываем год для фильтрации автомобилей

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

// Фильтруем автомобили по году приобретения

var filteredCars = cars

.Where(car => car.YearOfPurchase < filterYear) // Сравниваем с введенным годом

.ToList();

// Сортируем отфильтрованные автомобили по пробегу

filteredCars.Sort();

// Выводим список отфильтрованных и отсортированных автомобилей

Console.WriteLine("Автомобили, выпущенные ранее " + filterYear + " года, отсортированные по пробегу:");

foreach (var car in filteredCars)

{

Console.WriteLine($"Марка: {car.Brand}, Владелец: {car.Owner}, Год приобретения: {car.YearOfPurchase}, Пробег: {car.Mileage}");

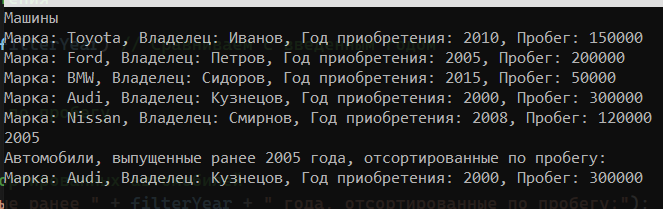
}

// Ожидание ввода пользователя перед завершением программы

Console.ReadLine();

}

}



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.

1. 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.

1. Какие члены может содержать 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.

1. Какие спецификаторы допустимы у методов, реализующих интерфейс?

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

1. В каких случаях используется явная реализация интерфейса?

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

1. Как осуществляется наследование интерфейсов?

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

1. Можно ли явно реализованные методы объявлять виртуальными?

Нет, явно реализованные методы нельзя объявлять виртуальными.

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

Да, можно.

1. Какие стандартные интерфейсы используются для работы с коллекциями?

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

1. Чем отличаются интерфейсы IComparable и IComparer?

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