Билет 1

1. Опишите, как выполняется программа на платформе .NET

Выполнение программы на платформе .NET — это многоэтапный процесс, который обеспечивает независимость от конкретного процессора и операционной системы. Вот ключевые шаги:

* Написание кода и компиляция в CIL (MSIL) Разработчик пишет исходный код на одном из .NET-совместимых языков, таких как C#, F# или Visual Basic .NET. Компилятор соответствующего языка (например, csc.exe для C#) не компилирует код напрямую в машинный код, понятный процессору. Вместо этого он компилирует его в промежуточный язык — CIL (Common Intermediate Language), ранее известный как MSIL (Microsoft Intermediate Language). Этот CIL-код является универсальным и не зависит от архитектуры процессора (x86, x64, ARM).
* Создание сборок (Assemblies) Скомпилированный CIL-код вместе с метаданными упаковывается в файл, который называется сборкой (assembly). Сборка может быть либо исполняемым файлом (.exe), либо библиотекой (.dll). Метаданные — это важнейшая часть сборки. Они содержат полную информацию о типах данных, методах, их параметрах и зависимостях, что позволяет среде выполнения эффективно управлять кодом.
* Запуск программы и роль CLR. Когда пользователь запускает .exe файл, операционная система передает управление общеязыковой среде выполнения — CLR (Common Language Runtime). CLR — это "сердце" платформы .NET. Она предоставляет ключевые сервисы для выполнения кода.
* JIT-компиляция (Just-In-Time Compilation) Процессор не может напрямую выполнять CIL-код. Поэтому CLR использует JIT-компилятор (компилятор "на лету"). Когда метод вызывается в первый раз, JIT-компилятор переводит CIL-код этого метода в нативный машинный код, оптимизированный для конкретного процессора и операционной системы, на которой запущена программа. Этот скомпилированный машинный код кэшируется в памяти. При последующих вызовах этого же метода будет использоваться уже готовый нативный код, что обеспечивает высокую производительность.
* Выполнение машинного кода После JIT-компиляции центральный процессор (CPU) выполняет полученный нативный код.
* Дополнительные службы CLR Во время выполнения программы CLR предоставляет множество важных сервисов: Управление памятью и сборка мусора (Garbage Collection, GC): Автоматически освобождает память от объектов, которые больше не используются. Безопасность (Security): Проверяет права доступа кода. Обработка исключений (Exception Handling): Предоставляет унифицированный механизм для работы с ошибками. Взаимодействие с другими компонентами (Interoperability): Позволяет .NET-коду вызывать неуправляемый код (например, библиотеки Win32 API или COM-объекты) и наоборот.

1. В одномерном массиве, состоящем из N целочисленных элементов, вычислить номер минимального

Эта задача решается путем одного прохода по массиву. Мы предполагаем, что первый элемент является минимальным, а затем сравниваем его со всеми остальными, обновляя наш минимум по мере необходимости.

using System;

public class ArrayMinFinder

{

public static void Main(string[] args)

{

// Пример массива

int[] numbers = { 15, 4, 88, -5, 12, -5, 42, 10 };

// Проверка на случай пустого массива

if (numbers == null || numbers.Length == 0)

{

Console.WriteLine("Массив пуст или не существует.");

return;

}

// 1. Инициализация

int minIndex = 0;

int minValue = numbers[0];

// 2. Цикл для поиска

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

{

// 3. Сравнение

if (numbers[i] < minValue)

{

// 4. Обновление

minValue = numbers[i];

minIndex = i;

}

}

// Вывод результата

Console.WriteLine("Массив: [" + string.Join(", ", numbers) + "]");

Console.WriteLine($"Минимальный элемент: {minValue}");

Console.WriteLine($"Номер (индекс) минимального элемента: {minIndex}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: МЕБЕЛЬ (производитель – string, вид – string, стоимость – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: вид. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderBy

// 1. Определение пользовательского класса МЕБЕЛЬ (Mebel)

public class Mebel

{

// Поля данных в виде авто-свойств (современный подход в C#)

// производитель – string

public string Manufacturer { get; set; }

// вид – string

public string Type { get; set; }

// стоимость – double

public double Price { get; set; }

// Конструктор для удобства создания объекта

public Mebel(string manufacturer, string type, double price)

{

Manufacturer = manufacturer;

Type = type;

Price = price;

}

// Пустой конструктор, чтобы можно было создать объект и заполнить поля позже

public Mebel() { }

// Переопределяем метод ToString() для красивого вывода объекта на экран

public override string ToString()

{

return $"Вид: {Type}, Производитель: {Manufacturer}, Стоимость: {Price:C}"; // :C форматирует как валюту

}

}

public class FurnitureProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8; // Для корректного вывода кириллицы в консоли

// Запрос количества элементов

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

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

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

}

// 2. Создание списка для хранения объектов

var furnitureList = new List<Mebel>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для мебели №{i + 1} ---");

var newItem = new Mebel();

Console.Write("Производитель: ");

newItem.Manufacturer = Console.ReadLine();

Console.Write("Вид (например, стол, стул, шкаф): ");

newItem.Type = Console.ReadLine();

Console.Write("Стоимость: ");

double price;

while (!double.TryParse(Console.ReadLine(), out price) || price < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Стоимость: ");

}

newItem.Price = price;

furnitureList.Add(newItem);

}

// 4. Сортировка списка по возрастанию по полю "вид" (Type)

// Используем LINQ метод OrderBy для простой и читаемой сортировки

var sortedList = furnitureList.OrderBy(item => item.Type).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список мебели (по виду) ---");

foreach (var item in sortedList)

{

Console.WriteLine(item);

}

}

}

Билет 2

1. Опишите типы данных: классификация типов данных, перечислите встроенные типы, назовите занимаемый размер, диапазон. Опишите преобразование типов. Приведите примеры.

В платформе .NET все типы данных делятся на две большие категории:

Типы-значения

* Где хранятся: Переменные этих типов хранят свое значение непосредственно в области памяти, где они объявлены (чаще всего в стеке).
* Что хранят: Само значение.
* Поведение при присваивании: При присваивании одной переменной типа-значения другой, происходит копирование значения. Изменение одной копии не влияет на другую.
* Примеры: Все числовые типы (int, double), структуры (struct), перечисления (enum), bool, char.

Ссылочные типы (Reference Types)

* Где хранятся: Сами объекты хранятся в специальной области памяти, называемой управляемой кучей (managed heap). Переменная же хранит не сам объект, а ссылку (адрес) на местоположение этого объекта в куче.
* Что хранят: Ссылку на объект.
* Поведение при присваивании: При присваивании происходит копирование ссылки, а не самого объекта. Обе переменные в итоге будут указывать на один и тот же объект в памяти. Изменение объекта через одну переменную будет видно через другую.
* Примеры: Классы (class), строки (string), массивы, делегаты, интерфейсы.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип C# | Тип .NET | Размер (байт) | Диапазон значений |
| Целочисленные со знаком | | | |
| sbyte | System.SByte | 1 | от -128 до 127 |
| short | System.Int16 | 2 | от -32 768 до 32 767 |
| int | System.Int32 | 4 | от -2 147 483 648 до 2 147 483 647 |
| long | System.Int64 | 8 | от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 |
| Целочисленные без знака | | | |
| byte | System.Byte | 1 | от 0 до 255 |
| ushort | System.UInt16 | 2 | от 0 до 65 535 |
| uint | System.UInt32 | 4 | от 0 до 4 294 967 295 |
| ulong | System.UInt64 | 8 | от 0 до 18 446 744 073 709 551 615 |
| С плавающей точкой | | | |
| float | System.Single | 4 | ±1.5 x 10⁻⁴⁵ до ±3.4 x 10³⁸ (точность ~6-9 цифр) |
| double | System.Double | 8 | ±5.0 × 10⁻³²⁴ до ±1.7 × 10³⁰⁸ (точность ~15-17 цифр) |
| decimal | System.Decimal | 16 | ±1.0 x 10⁻²⁸ до ±7.9228 x 10²⁸ (точность 28-29 цифр) |
| Другие типы | | | |
| char | System.Char | 2 | Символы Unicode (U+0000 до U+FFFF) |
| bool | System.Boolean | 1 | true или false |

Неявное (Implicit Conversion). Выполняется компилятором автоматически, когда нет риска потери данных. Например, при преобразовании из меньшего целочисленного типа в больший.

Явное (Explicit Conversion, или приведение типов). Требуется, когда есть риск потери данных (например, потеря точности или выход за пределы диапазона). Программист должен явно указать свое намерение, используя оператор приведения (тип).

1. В одномерном массиве, состоящем из N вещественных элементов, вычислить номер максимального по модулю элемента массива.

Для решения этой задачи нужно пройти по всему массиву, для каждого элемента вычислять его модуль (Math.Abs()) и сравнивать с текущим максимальным модулем.

using System;

class ArrayTask

{

static void Main(string[] args)

{

// N - количество элементов. Для примера зададим массив вручную.

// Вещественные элементы - используем тип double.

double[] array = { 1.5, -2.8, 0.7, 10.1, -15.5, 3.0, -15.4 };

// Проверка, что массив не пустой

if (array.Length == 0)

{

Console.WriteLine("Массив пуст.");

return;

}

// 1. Инициализируем переменные для хранения индекса и значения

// максимального по модулю элемента.

// Начнем с первого элемента массива.

int indexOfMaxAbs = 0;

double maxAbsValue = Math.Abs(array[0]);

// 2. Проходим по массиву, начиная со второго элемента (индекс 1)

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

{

// Вычисляем модуль текущего элемента

double currentAbsValue = Math.Abs(array[i]);

// 3. Сравниваем с максимальным найденным модулем

if (currentAbsValue > maxAbsValue)

{

// Если текущий модуль больше, обновляем наши переменные

maxAbsValue = currentAbsValue;

indexOfMaxAbs = i;

}

}

// 4. Выводим результат

Console.WriteLine("Исходный массив:");

Console.WriteLine(string.Join(", ", array)); // Удобный способ вывести массив

Console.WriteLine($"\nМаксимальный по модулю элемент: {array[indexOfMaxAbs]}");

Console.WriteLine($"Его значение по модулю: {maxAbsValue}");

Console.WriteLine($"Номер (индекс) этого элемента: {indexOfMaxAbs}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: СТУДЕНТ (имя – string, курс – int, пол – char). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: курс. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Необходимо для использования OrderBy

// Пользовательский класс СТУДЕНТ (определен выше)

public class Student

{

public string Name { get; set; }

public int Course { get; set; }

public char Gender { get; set; }

public Student(string name, int course, char gender)

{

Name = name;

Course = course;

Gender = gender;

}

public override string ToString()

{

return $"Имя: {Name, -15} | Курс: {Course} | Пол: {Gender}";

}

}

class StudentsProgram

{

static void Main(string[] args)

{

// 1. Создание списка для хранения объектов класса Student

List<Student> students = new List<Student>();

// 2. Запрос количества студентов

Console.Write("Введите количество студентов (N): ");

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

// 3. Заполнение списка данными с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для студента #{i + 1} ---");

Console.Write("Имя: ");

string name = Console.ReadLine();

Console.Write("Курс: ");

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

Console.Write("Пол (М/Ж): ");

char gender = char.Parse(Console.ReadLine().ToUpper());

// Создаем новый объект и добавляем его в список

Student newStudent = new Student(name, course, gender);

students.Add(newStudent);

}

Console.WriteLine("\n--- Список студентов до сортировки ---");

foreach (var student in students)

{

Console.WriteLine(student);

}

// 4. Сортировка списка по полю "курс" по возрастанию

// Используем LINQ метод OrderBy. Он не изменяет исходный список,

// а возвращает новую отсортированную последовательность.

List<Student> sortedStudents = students.OrderBy(s => s.Course).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Список студентов после сортировки по курсу ---");

foreach (var student in sortedStudents)

{

Console.WriteLine(student);

}

}

}

Билет 3

1. Опишите синтаксис и назначение оператора if-else. Приведите примеры.

Оператор if-else (условный оператор) — это одна из фундаментальных конструкций в программировании, которая позволяет управлять потоком выполнения программы. Его основное назначение — выполнять определенный блок кода только в том случае, если заданное логическое условие истинно (true), и (опционально) выполнять другой блок кода, если это условие ложно (false). Проще говоря, if-else позволяет программе принимать решения.

Существует три основные формы этой конструкции:

* Простой if. Используется, когда нужно выполнить действие только при истинности условия. Если условие ложно, блок кода просто пропускается.
* Конструкция if-else. Используется, когда нужно выбрать один из двух путей выполнения: один для истинного условия, другой — для ложного.
* Конструкция if-else if-else. Используется для проверки нескольких последовательных условий. Как только одно из условий оказывается истинным, выполняется его блок кода, а все остальные проверки (else if и else) пропускаются.

1. В одномерном массиве, состоящем из N целочисленных элементов, найти сумму элементов массива, расположенных до последнего положительного элемента.

Алгоритм: Создать целочисленный массив. Найти индекс последнего положительного элемента. Для этого: Создать переменную lastPositiveIndex и инициализировать её значением, которое не может быть индексом (например, -1). Запустить цикл с конца массива к началу (for (int i = array.Length - 1; i >= 0; i--)). Как только будет найден первый элемент > 0, сохранить его индекс в lastPositiveIndex и прервать цикл (break), так как мы нашли то, что искали. Проверить, был ли найден положительный элемент (lastPositiveIndex != -1). Если не был найден, значит, в массиве нет положительных элементов, и сумма равна 0. Если был найден, то вычислить сумму. Вычислить сумму элементов, расположенных до этого индекса. Инициализировать переменную sum = 0.

Запустить цикл от начала массива до lastPositiveIndex (не включая его, т.е. for (int i = 0; i < lastPositiveIndex; i++)). В цикле прибавлять каждый элемент к sum. Вывести результат.

using System;

class ArraySumTask

{

static void Main(string[] args)

{

// Целочисленный массив

int[] array = { 4, -2, 5, 8, -10, 3, -1, -6 }; // Последний положительный элемент '3' с индексом 5

// Сумма элементов до него: 4 + (-2) + 5 + 8 + (-10) = 5

Console.WriteLine("Исходный массив:");

Console.WriteLine(string.Join(", ", array));

// 1. Находим индекс последнего положительного элемента

int lastPositiveIndex = -1;

for (int i = array.Length - 1; i >= 0; i--)

{

if (array[i] > 0)

{

lastPositiveIndex = i;

break; // Нашли, выходим из цикла

}

}

// 2. Вычисляем сумму

long sum = 0; // Используем long для суммы на случай больших чисел

if (lastPositiveIndex == -1)

{

Console.WriteLine("\nВ массиве нет положительных элементов. Сумма равна 0.");

}

else

{

Console.WriteLine($"\nПоследний положительный элемент: {array[lastPositiveIndex]} (индекс: {lastPositiveIndex})");

// Суммируем все элементы до этого индекса

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

{

sum += array[i];

}

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

}

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: АВТОМОБИЛЬ (модель – string, мощность – double, стоимость – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: мощность. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Threading;

// Класс Automobile (определен выше)

public class Automobile

{

public string Model { get; set; }

public double Power { get; set; }

public double Cost { get; set; }

public Automobile(string model, double power, double cost)

{

Model = model;

Power = power;

Cost = cost;

}

public override string ToString()

{

return $"Модель: {Model,-15} | Мощность: {Power,7:F1} л.с. | Стоимость: {Cost:C}";

}

}

class AutoProgram

{

static void Main(string[] args)

{

// Устанавливаем культуру, чтобы разделителем для double была точка, а валюта - рубли

Thread.CurrentThread.CurrentCulture = new CultureInfo("ru-RU");

// 1. Создание списка

List<Automobile> cars = new List<Automobile>();

// 2. Запрос количества автомобилей

Console.Write("Введите количество автомобилей (N): ");

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

// 3. Заполнение списка

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

{

Console.WriteLine($"\n--- Ввод данных для автомобиля #{i + 1} ---");

Console.Write("Модель: ");

string model = Console.ReadLine();

Console.Write("Мощность (л.с.): ");

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

Console.Write("Стоимость: ");

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

cars.Add(new Automobile(model, power, cost));

}

Console.WriteLine("\n--- Список автомобилей до сортировки ---");

foreach (var car in cars)

{

Console.WriteLine(car);

}

// 4. Сортировка по возрастанию мощности

List<Automobile> sortedCars = cars.OrderBy(car => car.Power).ToList();

// 5. Вывод отсортированного списка

Console.WriteLine("\n--- Список автомобилей после сортировки по мощности ---");

foreach (var car in sortedCars)

{

Console.WriteLine(car);

}

}

}

Билет 4

1. Опишите синтаксис и назначение оператора switch-case. Приведите примеры.

Оператор switch-case (оператор выбора) — это управляющая конструкция, которая позволяет выбрать для выполнения один из множества блоков кода в зависимости от значения переменной или выражения. Он является более читаемой и структурированной альтернативой длинным цепочкам if-else if-else, когда все сравнения идут с одной и той же переменной. Основное назначение — сравнить значение выражения с набором константных значений (case) и выполнить соответствующий этому значению код.

switch (выражение): В скобках указывается переменная или выражение, результат которого будет сравниваться. Тип этого выражения может быть целочисленным (int, char, byte...), enum, string или bool.

case константа:: Метка, которая определяет одно из возможных значений. константа должна быть литералом или константным выражением.

break;: Оператор, который немедленно завершает выполнение switch. В C# он обязателен в конце каждого непустого блока case. Это предотвращает случайное "проваливание" (fall-through) к следующему case, что является частым источником ошибок в других языках (например, C++).

default:: Необязательная метка. Блок кода под ней выполняется, если значение выражения не совпало ни с одной из меток case. Рекомендуется всегда использовать default для обработки непредвиденных значений.

1. В одномерном массиве, состоящем из N целочисленных элементов, вычислить количество элементов массива кратных 3, расположенных на нечётных местах.

Алгоритм: Создать целочисленный массив. Инициализировать счетчик count = 0. Пройти по массиву в цикле for от i = 0 до конца массива. Внутри цикла для каждого элемента проверять два условия одновременно: Условие 1 (место): Индекс элемента чётный (i % 2 == 0). Условие 2 (значение): Значение элемента кратно 3 (array[i] % 3 == 0). Если оба условия истинны, увеличить счетчик count на 1. После завершения цикла вывести значение count.

using System;

class ArrayCountTask

{

static void Main(string[] args)

{

int[] array = { 9, 4, 12, 5, 6, 15, 7, 21 };

// ^ ^ ^ ^

// Индексы: 0 1 2 3 4 5 6 7

// Места: 1 2 3 4 5 6 7 8

//

// Проверяем элементы на нечетных местах (индексы 0, 2, 4, 6):

// array[0] = 9. Место 1-е (нечетное), 9 % 3 == 0 (да) -> +1

// array[2] = 12. Место 3-е (нечетное), 12 % 3 == 0 (да) -> +1

// array[4] = 6. Место 5-е (нечетное), 6 % 3 == 0 (да) -> +1

// array[6] = 7. Место 7-е (нечетное), 7 % 3 != 0 (нет)

// Итого: 3

int count = 0;

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

{

// Проверяем, что место нечетное (т.е. индекс четный)

bool isOddPlace = (i % 2 == 0);

if (isOddPlace)

{

// Если место нечетное, проверяем, кратно ли значение трём

if (array[i] % 3 == 0)

{

count++; // Увеличиваем счетчик

}

}

}

Console.WriteLine("Исходный массив:");

Console.WriteLine(string.Join(", ", array));

Console.WriteLine($"Количество элементов, кратных 3, на нечётных местах: {count}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: РЕКА (название – string, площадь бассейна – double, протяженность – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: площадь бассейна. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Threading;

// Класс River (определен выше)

public class River

{

public string Name { get; set; }

public double BasinArea { get; set; }

public double Length { get; set; }

public River(string name, double basinArea, double length)

{

Name = name;

BasinArea = basinArea;

Length = length;

}

public override string ToString()

{

return $"Название: {Name,-15} | Площадь бассейна: {BasinArea,10:N0} тыс. км² | Протяженность: {Length:N0} км";

}

}

class RiversProgram

{

static void Main(string[] args)

{

// Устанавливаем культуру для корректного ввода чисел с плавающей точкой

Thread.CurrentThread.CurrentCulture = new CultureInfo("ru-RU");

List<River> rivers = new List<River>();

Console.Write("Введите количество рек (N): ");

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

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

{

Console.WriteLine($"\n--- Ввод данных для реки #{i + 1} ---");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Площадь бассейна (тыс. км²): ");

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

Console.Write("Протяженность (км): ");

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

rivers.Add(new River(name, basinArea, length));

}

Console.WriteLine("\n--- Список рек до сортировки ---");

foreach (var river in rivers)

{

Console.WriteLine(river);

}

// Сортировка по возрастанию площади бассейна с помощью LINQ

List<River> sortedRivers = rivers.OrderBy(r => r.BasinArea).ToList();

Console.WriteLine("\n--- Список рек после сортировки по площади бассейна ---");

foreach (var river in sortedRivers)

{

Console.WriteLine(river);

}

}

}

Билет 5

1. Опишите синтаксис и назначение оператора for. Приведите примеры.

Оператор for — это один из самых распространенных циклов в программировании. Его основное назначение — выполнение блока кода заранее известное (или вычисляемое) количество раз. Он идеально подходит для ситуаций, когда нужно перебрать элементы массива, выполнить действие N раз или пройти по последовательности с определенным шагом. Цикл for объединяет в одной строке три ключевых действия для управления циклом: инициализацию счетчика, проверку условия продолжения и изменение счетчика после каждой итерации.

Инициализация: Этот блок выполняется один раз перед началом первой итерации цикла. Обычно здесь объявляется и инициализируется переменная-счетчик (например, int i = 0). Условие: Это логическое выражение, которое проверяется перед каждой итерацией (включая первую). Если условие истинно (true), тело цикла выполняется. Как только условие становится ложным (false), цикл завершается. Итератор (модификатор): Этот блок выполняется после каждой итерации тела цикла. Обычно здесь изменяется счетчик (например, i++, i--, i += 2).

1. Найти количество положительных и отрицательных элементов массива.

Алгоритм: Создать массив целых чисел, включающий положительные, отрицательные и нулевые значения. Инициализировать два счетчика: positiveCount = 0 и negativeCount = 0. Пройти по каждому элементу массива. Для этого удобно использовать цикл foreach, так как нам не нужен индекс элемента. Внутри цикла проверять каждый элемент: Если элемент > 0, увеличить positiveCount. Если элемент < 0, увеличить negativeCount. Нулевые элементы игнорируются, так как они не являются ни положительными, ни отрицательными. После завершения цикла вывести значения обоих счетчиков.

using System;

class ArrayCountPosNegTask

{

static void Main(string[] args)

{

int[] array = { 5, -3, 0, 12, -8, -1, 4, 0, 100 };

int positiveCount = 0;

int negativeCount = 0;

// Используем цикл foreach для удобного перебора всех элементов

foreach (int number in array)

{

if (number > 0)

{

positiveCount++; // Увеличиваем счетчик положительных

}

else if (number < 0)

{

negativeCount++; // Увеличиваем счетчик отрицательных

}

}

Console.WriteLine("Исходный массив:");

Console.WriteLine(string.Join(", ", array));

Console.WriteLine($"\nКоличество положительных элементов: {positiveCount}");

Console.WriteLine($"Количество отрицательных элементов: {negativeCount}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ЖИВОТНОЕ (наименование – string, вид – string, средний вес – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: вид. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Threading;

// Класс Animal (определен выше)

public class Animal

{

public string Name { get; set; }

public string Species { get; set; }

public double AverageWeight { get; set; }

public Animal(string name, string species, double averageWeight)

{

Name = name;

Species = species;

AverageWeight = averageWeight;

}

public override string ToString()

{

return $"Вид: {Species,-15} | Наименование: {Name,-15} | Средний вес: {AverageWeight} кг";

}

}

class AnimalsProgram

{

static void Main(string[] args)

{

// Устанавливаем культуру для корректного ввода чисел с запятой

Thread.CurrentThread.CurrentCulture = new CultureInfo("ru-RU");

List<Animal> animals = new List<Animal>();

Console.Write("Введите количество животных (N): ");

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

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

{

Console.WriteLine($"\n--- Ввод данных для животного #{i + 1} ---");

Console.Write("Наименование (например, Амурский тигр): ");

string name = Console.ReadLine();

Console.Write("Вид (например, Хищник, Травоядное): ");

string species = Console.ReadLine();

Console.Write("Средний вес (кг): ");

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

animals.Add(new Animal(name, species, weight));

}

Console.WriteLine("\n--- Список животных до сортировки ---");

foreach (var animal in animals)

{

Console.WriteLine(animal);

}

// Сортировка списка по полю "вид" (Species) по возрастанию (алфавитному порядку)

List<Animal> sortedAnimals = animals.OrderBy(animal => animal.Species).ToList();

Console.WriteLine("\n--- Список животных после сортировки по виду ---");

foreach (var animal in sortedAnimals)

{

Console.WriteLine(animal);

}

}

}

Билет 6

1. Опишите синтаксис и назначение оператора while. Приведите примеры.

Оператор while (цикл с предусловием) предназначен для многократного выполнения блока кода до тех пор, пока указанное логическое условие остается истинным (true).

Ключевое отличие от цикла for заключается в том, что while идеально подходит для ситуаций, когда количество итераций заранее неизвестно. Цикл продолжается не определенное число раз, а до наступления некоторого события (например, пока пользователь не введет корректные данные, пока не будет достигнут конец файла, пока у игрового персонажа не закончится здоровье и т.д.).

Проверка условия происходит перед каждой итерацией. Если условие с самого начала ложно, тело цикла не выполнится ни разу.

1. В одномерном массиве, состоящем из N целочисленных элементов, вычислить сумму элементов массива с нечётными индексами.

Алгоритм: Создать целочисленный массив. Инициализировать переменную для хранения суммы, например, sum = 0. Использовать цикл for, так как нам важен именно индекс элемента. В цикле проверять, является ли текущий индекс i нечетным. Условие для этого: i % 2 != 0. Если индекс нечетный, прибавить значение элемента array[i] к переменной sum. После окончания цикла вывести результат.

using System;

class ArraySumOddIndexTask

{

static void Main(string[] args)

{

int[] array = { 10, 5, 20, 15, 30, 25 };

// Индексы: 0, 1, 2, 3, 4, 5

// Элементы с нечетными индексами: 5 (индекс 1), 15 (индекс 3), 25 (индекс 5)

// Сумма = 5 + 15 + 25 = 45

long sum = 0; // Используем long, чтобы избежать переполнения, если массив большой

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

{

// Проверяем, является ли индекс нечетным

if (i % 2 != 0)

{

sum += array[i]; // Добавляем элемент к сумме

}

}

Console.WriteLine("Исходный массив:");

Console.WriteLine(string.Join(", ", array));

Console.WriteLine($"\nСумма элементов с нечётными индексами: {sum}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: СЧЕТ (наименования – string, номер счета – int, сумма – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: номер счета. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Threading;

// Класс Account (определен выше)

public class Account

{

public string Name { get; set; }

public int AccountNumber { get; set; }

public double Balance { get; set; }

public Account(string name, int accountNumber, double balance)

{

Name = name;

AccountNumber = accountNumber;

Balance = balance;

}

public override string ToString()

{

return $"Номер счета: {AccountNumber,-8} | Наименование: {Name,-15} | Сумма: {Balance:C}";

}

}

class BankProgram

{

static void Main(string[] args)

{

// Устанавливаем культуру для корректного отображения валюты и ввода чисел

Thread.CurrentThread.CurrentCulture = new CultureInfo("ru-RU");

List<Account> accounts = new List<Account>();

Console.Write("Введите количество счетов (N): ");

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

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

{

Console.WriteLine($"\n--- Ввод данных для счета #{i + 1} ---");

Console.Write("Наименование счета (например, Текущий): ");

string name = Console.ReadLine();

Console.Write("Номер счета (целое число): ");

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

Console.Write("Сумма на счете: ");

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

accounts.Add(new Account(name, number, balance));

}

Console.WriteLine("\n--- Список счетов до сортировки ---");

foreach (var account in accounts)

{

Console.WriteLine(account);

}

// Сортировка списка по номеру счета (AccountNumber) по возрастанию

List<Account> sortedAccounts = accounts.OrderBy(acc => acc.AccountNumber).ToList();

Console.WriteLine("\n--- Список счетов после сортировки по номеру счета ---");

foreach (var account in sortedAccounts)

{

Console.WriteLine(account);

}

}

}

Билет 7

1. Опишите класс Array: назначение, свойства, методы. Приведите примеры.

System.Array — это базовый абстрактный класс для всех массивов в среде CLR (Common Language Runtime). Он предоставляет методы для создания, обработки, поиска и сортировки массивов и служит основой для всех типов массивов в C#. Прямо создать экземпляр класса Array нельзя, но его статические методы и свойства экземпляра используются для работы с любым массивом (например, int[], string[] и т.д.).

Назначение:

Основное назначение класса — предоставить общую функциональность для всех массивов, обеспечивая унифицированный способ работы с коллекциями фиксированного размера.

Ключевые свойства (экземпляра):

* Length: Возвращает общее количество элементов во всех измерениях массива. Это наиболее часто используемое свойство.
* Rank: Возвращает количество измерений (ранг) массива. Для одномерного массива Rank равен 1, для двумерного — 2 и т.д.
* IsFixedSize: Всегда возвращает true, так как размер массивов в .NET не может быть изменен после их создания.
* IsReadOnly: Возвращает false, указывая, что элементы массива можно изменять.

Ключевые методы (статические):

* Статические методы вызываются через сам класс Array (например, Array.Sort(...)).
* Sort(Array array): Сортирует элементы в одномерном массиве по возрастанию.
* Reverse(Array array): Изменяет порядок элементов в одномерном массиве на обратный.
* IndexOf(Array array, object value): Ищет указанное значение и возвращает индекс его первого вхождения в одномерном массиве. Если элемент не найден, возвращает -1.
* Clear(Array array, int index, int length): Устанавливает для диапазона элементов в массиве значение по умолчанию для их типа (0 для чисел, null для ссылочных типов).
* Copy(Array sourceArray, Array destinationArray, int length): Копирует указанное количество элементов из одного массива в другой.
* Find<T>(T[] array, Predicate<T> match): Ищет элемент, удовлетворяющий условиям указанного предиката (делегата), и возвращает первое найденное вхождение.

Пример использования:

using System;

public class ArrayExample

{

public static void Main(string[] args)

{

// Создание обычного массива

int[] numbers = { 5, 1, 9, 3, 7 };

Console.WriteLine("Исходный массив: " + string.Join(", ", numbers));

// Использование свойства Length

Console.WriteLine($"Длина массива: {numbers.Length}");

// Использование статического метода Sort

Array.Sort(numbers);

Console.WriteLine("Массив после сортировки: " + string.Join(", ", numbers));

// Использование статического метода Reverse

Array.Reverse(numbers);

Console.WriteLine("Массив после реверса: " + string.Join(", ", numbers));

// Использование статического метода IndexOf

int index = Array.IndexOf(numbers, 7);

Console.WriteLine($"Индекс элемента '7': {index}");

// Использование статического метода Clear

// Очистим 2 элемента, начиная с индекса 1 (элементы 7 и 5)

Array.Clear(numbers, 1, 2);

Console.WriteLine("Массив после очистки части элементов: " + string.Join(", ", numbers));

}

}

1. Данные о температуре воздуха за декабрь хранятся в вещественном массиве. Определить, сколько раз температура была выше средней за этот месяц.

using System;

using System.Linq; // Подключаем для удобного использования методов Sum() и Average()

public class TemperatureAnalysis

{

public static void Main(string[] args)

{

// Пример массива температур за декабрь (31 день)

double[] temperatures = {

1.5, 0.2, -3.0, -5.1, -4.0, -2.5, 0.0, 1.1, 2.3, -1.0, -6.2, -7.0,

-5.5, -3.1, -1.8, 0.5, 2.0, 1.2, -0.9, -3.4, -4.8, -6.1, -5.0,

-2.2, -0.1, 1.9, 3.0, 2.1, -1.4, -3.7, -4.2

};

if (temperatures == null || temperatures.Length == 0)

{

Console.WriteLine("Массив данных о температуре пуст.");

return;

}

// 1. Вычисление средней температуры

// Можно сделать циклом: double sum = 0; foreach(var t in temp) sum+=t; double avg = sum/temp.Length;

// Но с LINQ это делается в одну строку и более читаемо.

double averageTemp = temperatures.Average();

// 2. Подсчет количества дней с температурой выше средней

int daysAboveAverage = 0;

foreach (double temp in temperatures)

{

if (temp > averageTemp)

{

daysAboveAverage++;

}

}

// Альтернативный, более короткий способ с LINQ

// int daysAboveAverageLinq = temperatures.Count(temp => temp > averageTemp);

// Вывод результата

Console.WriteLine("Данные о температуре за месяц: [" + string.Join("; ", temperatures) + "]");

Console.WriteLine($"Средняя температура за месяц: {averageTemp:F2}°C"); // F2 - форматирование до 2 знаков после запятой

Console.WriteLine($"Количество дней с температурой выше средней: {daysAboveAverage}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ТОВАР (наименование – string, количество – int, стоимость – float). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: наименование. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderByDescending

// 1. Определение пользовательского класса ТОВАР (Tovar)

public class Tovar

{

// Поля данных в виде авто-свойств, которые реализуют методы для просмотра и установки

// наименование – string

public string Name { get; set; }

// количество – int

public int Quantity { get; set; }

// стоимость – float

public float Price { get; set; }

// Конструктор для удобства создания объекта

public Tovar(string name, int quantity, float price)

{

Name = name;

Quantity = quantity;

Price = price;

}

// Пустой конструктор для создания объекта перед заполнением с клавиатуры

public Tovar() { }

// Переопределяем метод ToString() для красивого вывода объекта на экран

public override string ToString()

{

return $"Наименование: {Name}, Количество: {Quantity} шт., Стоимость: {Price:C}";

}

}

public class ProductProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8; // Для корректного вывода кириллицы

Console.Write("Введите количество товаров (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество товаров (N): ");

}

// 2. Создание списка для хранения объектов

var productList = new List<Tovar>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для товара №{i + 1} ---");

var newProduct = new Tovar();

Console.Write("Наименование: ");

newProduct.Name = Console.ReadLine();

Console.Write("Количество: ");

int quantity;

while (!int.TryParse(Console.ReadLine(), out quantity) || quantity < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое неотрицательное число.");

Console.Write("Количество: ");

}

newProduct.Quantity = quantity;

Console.Write("Стоимость: ");

float price;

while (!float.TryParse(Console.ReadLine(), out price) || price < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Стоимость: ");

}

newProduct.Price = price;

productList.Add(newProduct);

}

// 4. Сортировка списка по убыванию по полю "наименование" (Name)

// Используем LINQ метод OrderByDescending

var sortedList = productList.OrderByDescending(product => product.Name).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список товаров (по наименованию, по убыванию) ---");

foreach (var product in sortedList)

{

Console.WriteLine(product);

}

}

}

Билет 8

1. Раскройте понятие «рекурсия». Приведите примеры реализации рекурсии.

Рекурсия — это метод решения задачи, при котором функция вызывает саму себя для решения подзадачи меньшего размера. Этот процесс продолжается до тех пор, пока не будет достигнуто простое, базовое условие, которое можно решить без дальнейших вызовов.

Можно представить рекурсию на примере матрёшки: чтобы открыть самую маленькую куклу, нужно сначала открыть большую, затем ту, что внутри неё, и так далее, пока не дойдёшь до последней. Каждое "открытие" — это рекурсивный шаг.

Два ключевых компонента рекурсии:

1. Базовый случай (условие выхода): Это условие, при котором функция перестает вызывать себя и возвращает конкретное значение. Без базового случая рекурсия будет бесконечной и приведет к ошибке переполнения стека (StackOverflowException).
2. Рекурсивный шаг (шаг рекурсии): Это часть функции, где она вызывает саму себя, но с измененными параметрами, которые приближают ее к базовому случаю.

Преимущества рекурсии:

* Элегантность и читаемость: Для некоторых задач (например, обход деревьев, вычисление факториала) рекурсивный код выглядит проще и понятнее, чем итеративный (с использованием циклов).
* Соответствие задаче: Некоторые математические и структурные проблемы по своей природе рекурсивны.

Недостатки рекурсии:

* Производительность: Каждый вызов функции требует выделения памяти в стеке вызовов. Глубокая рекурсия может потреблять много памяти и быть медленнее циклов.
* Риск переполнения стека: Если рекурсия слишком глубока (например, при обработке очень больших наборов данных), она может исчерпать всю доступную память в стеке.

Пример: Вычисление факториала

Факториал числа N (обозначается как N!) — это произведение всех натуральных чисел от 1 до N.

* 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120
* Математическое определение: N! = N \* (N-1)!
* Базовый случай: 0! = 1

using System;

public class RecursionExample

{

// Рекурсивный метод для вычисления факториала

public static long Factorial(int n)

{

// 1. Базовый случай (условие выхода)

if (n == 0)

{

return 1;

}

// 2. Рекурсивный шаг

else

{

// Функция вызывает саму себя с аргументом, который на 1 меньше (n - 1),

// приближаясь к базовому случаю (n == 0).

return n \* Factorial(n - 1);

}

}

public static void Main(string[] args)

{

int number = 5;

long result = Factorial(number);

Console.WriteLine($"Факториал числа {number} равен {result}"); // Вывод: Факториал числа 5 равен 120

number = 7;

result = Factorial(number);

Console.WriteLine($"Факториал числа {number} равен {result}"); // Вывод: Факториал числа 7 равен 5040

}

}

1. Найти сумму элементов массива, расположенных после первого отрицательного элемента.

using System;

public class ArraySumAfterNegative

{

public static void Main(string[] args)

{

// Пример массива

int[] numbers = { 10, 20, 5, -8, 15, -2, 30 };

// 1. Находим индекс первого отрицательного элемента

int firstNegativeIndex = -1; // -1 означает, что элемент не найден

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

{

if (numbers[i] < 0)

{

firstNegativeIndex = i;

break; // Прерываем цикл, так как нам нужен только первый

}

}

long sum = 0; // Используем long на случай большой суммы

// 2. Если отрицательный элемент был найден

if (firstNegativeIndex != -1)

{

Console.WriteLine($"Первый отрицательный элемент: {numbers[firstNegativeIndex]} (индекс {firstNegativeIndex})");

// Проверяем, не является ли он последним в массиве

if (firstNegativeIndex < numbers.Length - 1)

{

// 3. Суммируем элементы после него

for (int i = firstNegativeIndex + 1; i < numbers.Length; i++)

{

sum += numbers[i];

}

}

else

{

Console.WriteLine("Отрицательный элемент является последним в массиве, сумма равна 0.");

}

}

else

{

Console.WriteLine("В массиве нет отрицательных элементов.");

}

// Вывод результата

Console.WriteLine("Массив: [" + string.Join(", ", numbers) + "]");

Console.WriteLine($"Сумма элементов после первого отрицательного: {sum}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОБУВЬ (производитель – string, размер – double, цена – double). Условие: отсортировать список по возрастанию. Поле для выполнения сравнения объектов списка: размер. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderBy

// 1. Определение пользовательского класса ОБУВЬ (Shoe)

public class Shoe

{

// Авто-свойства реализуют методы для просмотра и установки полей

// производитель – string

public string Manufacturer { get; set; }

// размер – double

public double Size { get; set; }

// цена – double

public double Price { get; set; }

// Конструктор для удобства создания объекта

public Shoe(string manufacturer, double size, double price)

{

Manufacturer = manufacturer;

Size = size;

Price = price;

}

// Пустой конструктор для создания объекта перед заполнением

public Shoe() { }

// Переопределяем метод ToString() для красивого вывода

public override string ToString()

{

return $"Размер: {Size}, Производитель: {Manufacturer}, Цена: {Price:C}";

}

}

public class ShoeProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8; // Для корректного вывода кириллицы

Console.Write("Введите количество пар обуви (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество пар обуви (N): ");

}

// 2. Создание списка для хранения объектов

var shoeList = new List<Shoe>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для пары обуви №{i + 1} ---");

var newShoe = new Shoe();

Console.Write("Производитель: ");

newShoe.Manufacturer = Console.ReadLine();

Console.Write("Размер: ");

double size;

while (!double.TryParse(Console.ReadLine(), out size) || size <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Размер: ");

}

newShoe.Size = size;

Console.Write("Цена: ");

double price;

while (!double.TryParse(Console.ReadLine(), out price) || price < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Цена: ");

}

newShoe.Price = price;

shoeList.Add(newShoe);

}

// 4. Сортировка списка по возрастанию по полю "размер" (Size)

// Используем LINQ метод OrderBy

var sortedList = shoeList.OrderBy(shoe => shoe.Size).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список обуви (по размеру) ---");

foreach (var shoe in sortedList)

{

Console.WriteLine(shoe);

}

}

}

Билет 9

1. Опишите синтаксис класса, способы создания объекта класса. Приведите примеры.

Класс — это основной строительный блок объектно-ориентированного программирования (ООП) в C#. Он представляет собой шаблон или чертеж для создания объектов, объединяя в себе данные (поля) и поведение (методы).

Синтаксис класса:

Объявление класса имеет следующую базовую структуру:

[модификатор\_доступа] class ИмяКласса

{

// Тело класса:

// 1. Поля (Fields) - переменные для хранения состояния объекта.

// Обычно их делают приватными (private).

private string \_someField;

// 2. Свойства (Properties) - предоставляют управляемый доступ к полям.

// Это предпочтительный способ для работы с данными класса извне.

public string SomeProperty { get; set; }

// 3. Конструкторы (Constructors) - специальные методы для инициализации объекта при его создании.

// Имя конструктора совпадает с именем класса.

public ИмяКласса(string initialValue)

{

\_someField = initialValue;

}

// 4. Методы (Methods) - определяют поведение объекта, то есть действия, которые он может выполнять.

public void DoSomething()

{

// Логика метода

}

}

* модификатор\_доступа (например, public, internal) определяет, где этот класс будет виден. public означает, что класс доступен из любого места проекта.
* class — ключевое слово, обозначающее объявление класса.
* ИмяКласса — уникальное имя класса, по соглашению пишется с большой буквы (PascalCase).

Способы создания объекта класса (инстанцирование):

Создание объекта — это процесс выделения памяти под конкретный экземпляр класса. Основным инструментом для этого является оператор new.

1. Вызов конструктора по умолчанию (без параметров)

Если в классе не определено ни одного конструктора, компилятор создает его автоматически.

ИмяКласса myObject = new ИмяКласса();

1. Вызов параметризованного конструктора

Этот способ позволяет сразу задать начальные значения полей при создании объекта.

// Предполагается, что в классе есть конструктор public ИмяКласса(string value)

ИмяКласса myObject = new ИмяКласса("начальное значение");

1. Использование инициализатора объектов

Это удобный синтаксис, позволяющий задать значения публичных свойств сразу после создания объекта, без необходимости вызывать конструктор с параметрами.

ИмяКласса myObject = new ИмяКласса()

{

SomeProperty = "какое-то значение"

};

// Это можно совмещать с вызовом конструктора

ИмяКласса myObject2 = new ИмяКласса("значение для конструктора")

{

SomeProperty = "другое значение"

};

Пример:

using System;

// Определение класса Car

public class Car

{

// Свойства

public string Model { get; set; }

public int Year { get; set; }

private double \_currentSpeed; // Приватное поле

// Конструктор

public Car(string model, int year)

{

Model = model;

Year = year;

\_currentSpeed = 0;

}

// Метод

public void Accelerate(double speedIncrease)

{

\_currentSpeed += speedIncrease;

Console.WriteLine($"Машина {Model} ускорилась. Текущая скорость: {\_currentSpeed} км/ч");

}

}

public class Program

{

public static void Main()

{

// Способ 1: Создание объекта с помощью параметризованного конструктора

Car myCar1 = new Car("Tesla Model 3", 2021);

Console.WriteLine($"Создана машина: {myCar1.Model}, Год: {myCar1.Year}");

myCar1.Accelerate(50);

// Способ 2: Создание с помощью конструктора и инициализатора

// (в данном случае избыточно, но демонстрирует синтаксис)

Car myCar2 = new Car("Ford Mustang", 2020)

{

// Здесь можно было бы переопределить публичные свойства, если нужно

};

Console.WriteLine($"\nСоздана машина: {myCar2.Model}, Год: {myCar2.Year}");

myCar2.Accelerate(70);

}

}

1. В одномерном массиве, состоящем из N вещественных элементов, вычислить номер минимального элемента массива.

using System;

public class ArrayMinIndexFinder

{

public static void Main(string[] args)

{

// Пример массива вещественных чисел

double[] numbers = { 25.5, 10.1, 120.7, 4.25, -9.5, 18.0, 4.25 };

// Проверка на случай пустого массива

if (numbers == null || numbers.Length == 0)

{

Console.WriteLine("Массив пуст или не существует.");

return;

}

// 1. Инициализация

int minIndex = 0;

double minValue = numbers[0];

// 2. Цикл для поиска

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

{

// 3. Сравнение

if (numbers[i] < minValue)

{

// 4. Обновление

minValue = numbers[i];

minIndex = i;

}

}

// Вывод результата

Console.WriteLine("Массив: [" + string.Join(", ", numbers) + "]");

Console.WriteLine($"Минимальный элемент: {minValue}");

Console.WriteLine($"Номер (индекс) минимального элемента: {minIndex}");

}

}

Примечание: Если в массиве несколько одинаковых минимальных элементов, этот алгоритм найдет индекс первого из них.

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: КОРАБЛЬ (имя – string, водоизмещение – double, тип – string). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: тип. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderByDescending

// 1. Определение пользовательского класса КОРАБЛЬ (Ship)

public class Ship

{

// Свойства для просмотра и установки полей данных

// имя – string

public string Name { get; set; }

// водоизмещение – double

public double Displacement { get; set; }

// тип – string

public string Type { get; set; }

// Конструктор

public Ship(string name, double displacement, string type)

{

Name = name;

Displacement = displacement;

Type = type;

}

// Пустой конструктор

public Ship() { }

// Переопределяем метод ToString() для красивого вывода

public override string ToString()

{

return $"Тип: {Type}, Имя: {Name}, Водоизмещение: {Displacement} т.";

}

}

public class ShipProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.Write("Введите количество кораблей (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество кораблей (N): ");

}

// 2. Создание списка для хранения объектов

var shipList = new List<Ship>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для корабля №{i + 1} ---");

var newShip = new Ship();

Console.Write("Имя: ");

newShip.Name = Console.ReadLine();

Console.Write("Тип (например, крейсер, эсминец, авианосец): ");

newShip.Type = Console.ReadLine();

Console.Write("Водоизмещение (в тоннах): ");

double displacement;

while (!double.TryParse(Console.ReadLine(), out displacement) || displacement < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Водоизмещение (в тоннах): ");

}

newShip.Displacement = displacement;

shipList.Add(newShip);

}

// 4. Сортировка списка по убыванию по полю "тип" (Type)

// Используем LINQ метод OrderByDescending

var sortedList = shipList.OrderByDescending(ship => ship.Type).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список кораблей (по типу, по убыванию) ---");

foreach (var ship in sortedList)

{

Console.WriteLine(ship);

}

}

}

Билет 10

1. Опишите передачу параметров методу по значению, передачу необязательных параметров методов.

Передача параметров по значению (Pass by Value)

Это стандартный способ передачи параметров в методы в C#. Суть его заключается в том, что метод получает копию значения переменной, а не саму переменную. Поведение зависит от типа передаваемой переменной:

Для типов значений (Value Types): int, double, float, bool, char, struct.

В метод передается полная копия значения переменной. Любые изменения параметра внутри метода не влияют на исходную переменную, которая была передана в качестве аргумента.

Пример:

void ModifyValue(int number)

{

number = 100; // Изменение копии

Console.WriteLine($"Внутри метода: {number}"); // Выведет 100

}

int myValue = 5;

Console.WriteLine($"До вызова метода: {myValue}"); // Выведет 5

ModifyValue(myValue);

Console.WriteLine($"После вызова метода: {myValue}"); // Выведет 5 (исходное значение не изменилось)

Для ссылочных типов (Reference Types): string, class, array, List.

В метод передается копия ссылки (адреса в памяти), которая указывает на объект. Обе переменные (исходная и параметр метода) указывают на один и тот же объект в памяти.

* Если внутри метода изменить состояние объекта (например, поменять значение его свойства), эти изменения будут видны и снаружи, так как объект один и тот же.
* Однако если внутри метода попытаться присвоить параметру новую ссылку (создать новый объект через new), это повлияет только на локальную копию ссылки. Исходная переменная продолжит указывать на старый объект.

Пример:

class MyData { public string Name { get; set; } }

void ModifyReference(MyData data)

{

// 1. Изменение свойства объекта - повлияет на оригинал

data.Name = "Modified";

// 2. Присваивание новой ссылки - НЕ повлияет на оригинал

data = new MyData { Name = "New Object" };

}

MyData myObject = new MyData { Name = "Original" };

Console.WriteLine($"До вызова: {myObject.Name}"); // Выведет "Original"

ModifyReference(myObject);

Console.WriteLine($"После вызова: {myObject.Name}"); // Выведет "Modified"

Необязательные (опциональные) параметры

Необязательные параметры позволяют вызывать метод, опуская некоторые из его аргументов. Для этого в объявлении метода параметру присваивается значение по умолчанию с помощью оператора =.

Правила:

* Все необязательные параметры должны быть объявлены после всех обязательных параметров.
* Значение по умолчанию должно быть константой времени компиляции.

Это помогает избежать создания множества перегрузок одного и того же метода с разным количеством параметров.

Пример:

// `message` - обязательный параметр

// `isWarning` - необязательный, по умолчанию false

void Log(string message, bool isWarning = false)

{

string prefix = isWarning ? "Warning: " : "Info: ";

Console.WriteLine(prefix + message);

}

// Вызов с одним обязательным аргументом. isWarning будет false.

Log("File saved successfully."); // Вывод: Info: File saved successfully.

// Вызов с двумя аргументами. isWarning будет true.

Log("Disk space is low.", true); // Вывод: Warning: Disk space is low.

1. В одномерном массиве, состоящем из N вещественных элементов, заменить все отрицательные элементы массива числом 0.

using System;

public class ReplaceNegatives

{

public static void Main(string[] args)

{

// Пример массива вещественных чисел

double[] numbers = { 10.5, -3.2, 0.0, 15.8, -99.1, -0.1, 42.0 };

Console.WriteLine("Исходный массив: [" + string.Join(", ", numbers) + "]");

// Цикл для прохода по всем элементам массива

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

{

// Проверка, является ли элемент отрицательным

if (numbers[i] < 0)

{

// Замена отрицательного элемента на 0

numbers[i] = 0;

}

}

// Вывод результата

Console.WriteLine("Массив после замены: [" + string.Join(", ", numbers) + "]");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: АВТОБУС (модель – string, мощность – float, количество мест – int). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: количество мест. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderByDescending

// 1. Определение пользовательского класса АВТОБУС (Bus)

public class Bus

{

// Авто-свойства для просмотра и установки полей

// модель – string

public string Model { get; set; }

// мощность – float

public float Power { get; set; }

// количество мест – int

public int Seats { get; set; }

// Пустой конструктор для создания объекта перед заполнением

public Bus() { }

// Переопределяем метод ToString() для красивого вывода

public override string ToString()

{

return $"Мест: {Seats}, Модель: {Model}, Мощность: {Power} л.с.";

}

}

public class BusProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.Write("Введите количество автобусов (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество автобусов (N): ");

}

// 2. Создание списка для хранения объектов

var busList = new List<Bus>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для автобуса №{i + 1} ---");

var newBus = new Bus();

Console.Write("Модель: ");

newBus.Model = Console.ReadLine();

Console.Write("Количество мест: ");

int seats;

while (!int.TryParse(Console.ReadLine(), out seats) || seats < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое неотрицательное число.");

Console.Write("Количество мест: ");

}

newBus.Seats = seats;

Console.Write("Мощность (л.с.): ");

float power;

while (!float.TryParse(Console.ReadLine(), out power) || power < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Мощность (л.с.): ");

}

newBus.Power = power;

busList.Add(newBus);

}

// 4. Сортировка списка по убыванию по полю "количество мест" (Seats)

// Используем LINQ метод OrderByDescending

var sortedList = busList.OrderByDescending(bus => bus.Seats).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список автобусов (по количеству мест, по убыванию) ---");

foreach (var bus in sortedList)

{

Console.WriteLine(bus);

}

}

}

Билет 11

1. Опишите, как выполняется наследование классов, доступ к компонентам базового класса.

Наследование — это один из ключевых принципов объектно-ориентированного программирования (ООП), который позволяет создавать новый класс (наследник или производный класс) на основе уже существующего (базового класса или родителя). Производный класс автоматически получает (наследует) все не-private поля, свойства и методы базового класса.

Как выполняется наследование:

Наследование в C# указывается с помощью двоеточия (:) после имени производного класса, за которым следует имя базового класса.

Синтаксис:

// Базовый класс

public class Animal

{

public string Name { get; set; }

public void Eat()

{

Console.WriteLine($"{Name} ест.");

}

}

// Производный класс, который наследует от Animal

public class Dog : Animal

{

// Dog теперь имеет свойство Name и метод Eat()

public string Breed { get; set; } // Дополнительное свойство

public void Bark() // Дополнительный метод

{

Console.WriteLine($"{Name} лает.");

}

}

В этом примере класс Dog наследует от Animal. Это означает, что любой объект типа Dog будет иметь свойства Name и Breed, а также методы Eat() и Bark().

Доступ к компонентам базового класса:

Для обращения к членам базового класса из производного класса используется ключевое слово base. Это особенно полезно в двух случаях:

1. Вызов конструктора базового класса: Если у базового класса есть конструктор с параметрами, производный класс обязан вызвать его в своем конструкторе. Это делается с помощью : base(параметры).
2. Вызов метода базового класса: Если в производном классе переопределен метод базового класса (с помощью override), можно вызвать исходную реализацию этого метода из базового класса через base.ИмяМетода().

Пример использования base:

// Базовый класс с конструктором

public class Vehicle

{

public int MaxSpeed { get; }

public Vehicle(int maxSpeed)

{

MaxSpeed = maxSpeed;

Console.WriteLine("Конструктор Vehicle вызван.");

}

public virtual void ShowInfo()

{

Console.WriteLine($"Максимальная скорость: {MaxSpeed} км/ч");

}

}

// Производный класс

public class Car : Vehicle

{

public int NumberOfDoors { get; set; }

// 1. Вызываем конструктор базового класса с помощью : base(maxSpeed)

public Car(int maxSpeed, int doors) : base(maxSpeed)

{

NumberOfDoors = doors;

Console.WriteLine("Конструктор Car вызван.");

}

// Переопределяем метод и используем base для вызова родительской реализации

public override void ShowInfo()

{

// 2. Сначала вызываем метод базового класса

base.ShowInfo();

// Затем добавляем свою информацию

Console.WriteLine($"Количество дверей: {NumberOfDoors}");

}

}

// Использование:

Car myCar = new Car(200, 4);

/\*

Вывод конструкторов:

Конструктор Vehicle вызван.

Конструктор Car вызван.

\*/

myCar.ShowInfo();

/\*

Вывод информации:

Максимальная скорость: 200 км/ч

Количество дверей: 4

\*/

1. В одномерном массиве, состоящем из N целочисленных элементов, вычислить произведение элементов массива с чётными индексами.

using System;

using System.Numerics; // Подключаем для BigInteger на случай очень больших чисел

public class ProductOfEvenIndices

{

public static void Main(string[] args)

{

// Пример массива

int[] numbers = { 1, 5, 2, 8, 3, 4, 6 }; // Индексы: 0,1,2,3,4,5,6

// Элементы с четными индексами: 1, 2, 3, 6

if (numbers == null || numbers.Length == 0)

{

Console.WriteLine("Массив пуст.");

return;

}

// Используем BigInteger, чтобы избежать переполнения при перемножении

BigInteger product = 1;

// Цикл для прохода по массиву

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

{

// Проверяем, является ли индекс чётным

if (i % 2 == 0)

{

product \*= numbers[i];

}

}

// Вывод результата

Console.WriteLine("Массив: [" + string.Join(", ", numbers) + "]");

Console.WriteLine($"Произведение элементов с чётными индексами: {product}"); // 1 \* 2 \* 3 \* 6 = 36

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: СОБАКА (кличка – string, порода – string, возраст – float). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: возраст. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderByDescending

// 1. Определение пользовательского класса СОБАКА (Dog)

public class Dog

{

// Авто-свойства для просмотра и установки полей

// кличка – string

public string Name { get; set; }

// порода – string

public string Breed { get; set; }

// возраст – float

public float Age { get; set; }

// Пустой конструктор

public Dog() { }

// Переопределяем метод ToString() для красивого вывода

public override string ToString()

{

return $"Возраст: {Age}, Порода: {Breed}, Кличка: {Name}";

}

}

public class DogProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.Write("Введите количество собак (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество собак (N): ");

}

// 2. Создание списка для хранения объектов

var dogList = new List<Dog>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для собаки №{i + 1} ---");

var newDog = new Dog();

Console.Write("Кличка: ");

newDog.Name = Console.ReadLine();

Console.Write("Порода: ");

newDog.Breed = Console.ReadLine();

Console.Write("Возраст: ");

float age;

while (!float.TryParse(Console.ReadLine(), out age) || age < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Возраст: ");

}

newDog.Age = age;

dogList.Add(newDog);

}

// 4. Сортировка списка по убыванию по полю "возраст" (Age)

// Используем LINQ метод OrderByDescending

var sortedList = dogList.OrderByDescending(dog => dog.Age).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список собак (по возрасту, по убыванию) ---");

foreach (var dog in sortedList)

{

Console.WriteLine(dog);

}

}

}

Билет 12

1. Опишите абстрактные классы, виртуальные методы, сокрытие имен. Приведите примеры.

Эти три концепции тесно связаны с наследованием и полиморфизмом в ООП.

Абстрактные классы (Abstract Classes)

* Что это: Абстрактный класс — это класс, который нельзя инстанцировать (нельзя создать его объект с помощью new). Он служит в качестве "шаблона" или базовой заготовки для других классов.
* Ключевое слово: abstract.
* Назначение: Определить общий интерфейс и, возможно, базовую реализацию для группы связанных классов. Он может содержать как обычные, так и абстрактные методы.
* Абстрактные методы: Это методы, объявленные с ключевым словом abstract в абстрактном классе. У них нет тела (реализации). Любой неабстрактный класс, наследующий от абстрактного, обязан реализовать (переопределить через override) все его абстрактные члены.

Виртуальные методы (Virtual Methods)

* Что это: Метод в базовом классе, помеченный ключевым словом virtual. Это означает, что метод имеет реализацию по умолчанию, но производные классы могут (но не обязаны) предоставить свою собственную реализацию, используя ключевое слово override.
* Назначение: Обеспечить полиморфизм. Когда виртуальный метод вызывается через ссылку на базовый класс, будет выполнена версия метода из того класса, объект которого был фактически создан.

Сокрытие имен (Method Hiding)

* Что это: Ситуация, когда производный класс объявляет метод с таким же именем и сигнатурой, как у метода в базовом классе, но помечает его ключевым словом new.
* Назначение: Явно указать, что вы создаете новый метод, который скрывает одноименный метод базового класса, а не переопределяете его.
* Ключевое отличие от override:
* При override (с virtual) выбор метода для вызова зависит от фактического типа объекта во время выполнения.
* При new (сокрытие) выбор метода зависит от типа переменной, через которую происходит вызов, во время компиляции.

Пример, демонстрирующий все три концепции:

using System;

// 1. Абстрактный класс

public abstract class Shape

{

// Обычное свойство, наследуется всеми

public string Name { get; set; }

// 2. Абстрактный метод: нет реализации, должен быть переопределен

public abstract double CalculateArea();

// 3. Виртуальный метод: есть реализация, может быть переопределен

public virtual void Display()

{

Console.WriteLine($"Это фигура '{Name}'.");

}

}

// Производный класс

public class Circle : Shape

{

public double Radius { get; set; }

public Circle(double radius)

{

Name = "Круг";

Radius = radius;

}

// Реализация обязательного абстрактного метода

public override double CalculateArea()

{

return Math.PI \* Radius \* Radius;

}

// Переопределение виртуального метода для добавления деталей

public override void Display()

{

base.Display(); // Вызываем реализацию из базового класса

Console.WriteLine($"Его площадь: {CalculateArea():F2}");

}

// 4. Сокрытие метода: в базовом классе нет такого метода, но если бы был,

// ключевое слово 'new' скрыло бы его.

public new void GetInfo()

{

Console.WriteLine("Это информация из класса Circle.");

}

}

public class Program

{

public static void Main()

{

// Shape myShape = new Shape(); // Ошибка! Нельзя создать экземпляр абстрактного класса.

Circle myCircle = new Circle(5);

myCircle.Display(); // Вызывается переопределенный метод из Circle

Console.WriteLine("--- Работа через ссылку на базовый класс ---");

// Полиморфизм в действии

Shape shapeRef = myCircle;

shapeRef.Display(); // Все равно вызывается переопределенный метод из Circle

}

}

1. Задать одномерный массив, состоящий из N вещественных элементов. Найти сумму элементов массива, расположенных между первым и вторым отрицательными элементами.

using System;

public class SumBetweenNegatives

{

public static void Main(string[] args)

{

// Пример массива

double[] numbers = { 10.1, 5.5, -2.0, 15.0, 8.2, 4.0, -9.5, 3.1, -1.0 };

int firstNegativeIndex = -1;

int secondNegativeIndex = -1;

// 1. Находим индексы первого и второго отрицательных элементов

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

{

if (numbers[i] < 0)

{

if (firstNegativeIndex == -1) // Если первый еще не найден

{

firstNegativeIndex = i;

}

else // Если первый уже найден, значит это второй

{

secondNegativeIndex = i;

break; // Дальше искать не нужно

}

}

}

Console.WriteLine("Массив: [" + string.Join(", ", numbers) + "]");

// 2. Проверяем, были ли найдены оба элемента

if (firstNegativeIndex != -1 && secondNegativeIndex != -1)

{

Console.WriteLine($"Первый отрицательный элемент: {numbers[firstNegativeIndex]} (индекс {firstNegativeIndex})");

Console.WriteLine($"Второй отрицательный элемент: {numbers[secondNegativeIndex]} (индекс {secondNegativeIndex})");

double sum = 0;

// 3. Суммируем элементы между ними

// Цикл от (индекс первого + 1) до (индекс второго)

for (int i = firstNegativeIndex + 1; i < secondNegativeIndex; i++)

{

sum += numbers[i];

}

Console.WriteLine($"Сумма элементов между ними: {sum}"); // 15.0 + 8.2 + 4.0 = 27.2

}

else

{

Console.WriteLine("В массиве не найдено двух отрицательных элементов.");

}

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ПРИНТЕР (модель – string, скорость печати – float, тип – string). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: модель. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq; // Подключаем для использования OrderByDescending

// 1. Определение пользовательского класса ПРИНТЕР (Printer)

public class Printer

{

// Авто-свойства для просмотра и установки полей

// модель – string

public string Model { get; set; }

// скорость печати – float

public float PrintSpeed { get; set; }

// тип – string

public string Type { get; set; }

// Пустой конструктор

public Printer() { }

// Переопределяем метод ToString() для красивого вывода

public override string ToString()

{

return $"Модель: {Model}, Тип: {Type}, Скорость печати: {PrintSpeed} стр/мин";

}

}

public class PrinterProgram

{

public static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.Write("Введите количество принтеров (N): ");

int n;

while (!int.TryParse(Console.ReadLine(), out n) || n <= 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите целое положительное число.");

Console.Write("Введите количество принтеров (N): ");

}

// 2. Создание списка для хранения объектов

var printerList = new List<Printer>();

// 3. Заполнение списка значениями с клавиатуры

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

{

Console.WriteLine($"\n--- Ввод данных для принтера №{i + 1} ---");

var newPrinter = new Printer();

Console.Write("Модель: ");

newPrinter.Model = Console.ReadLine();

Console.Write("Тип (например, лазерный, струйный, матричный): ");

newPrinter.Type = Console.ReadLine();

Console.Write("Скорость печати (стр/мин): ");

float speed;

while (!float.TryParse(Console.ReadLine(), out speed) || speed < 0)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите положительное число.");

Console.Write("Скорость печати (стр/мин): ");

}

newPrinter.PrintSpeed = speed;

printerList.Add(newPrinter);

}

// 4. Сортировка списка по убыванию по полю "модель" (Model)

// Используем LINQ метод OrderByDescending

var sortedList = printerList.OrderByDescending(p => p.Model).ToList();

// 5. Вывод отсортированного списка на экран

Console.WriteLine("\n--- Отсортированный список принтеров (по модели, по убыванию) ---");

foreach (var printer in sortedList)

{

Console.WriteLine(printer);

}

}

}

**Билет 13**

1. Опишите, как выполняется описание двумерного массива, обращение к элементам массива, его заполнение, вывод на экран. Приведите примеры.

Описание двумерного массива:

Двумерный массив в C# описывается следующим образом:

тип\_данных[,] имя\_массива = new тип\_данных[количество\_строк, количество\_столбцов];

Пример:

int[,] matrix = new int[3, 4]; // 3 строки, 4 столбца

Обращение к элементам массива:

Элементы массива индексируются с нуля. Обращение происходит по индексам [i, j].

int element = matrix[1, 2]; // элемент во 2-й строке и 3-м столбце

Заполнение массива:

Можно заполнить массив вручную или с помощью циклов.

Пример заполнения случайными числами:

Random rnd = new Random();

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

matrix[i, j] = rnd.Next(1, 10); // числа от 1 до 9

}

}

Вывод массива на экран:

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

Console.Write(matrix[i, j] + " ");

}

Console.WriteLine();

}

Полный пример:

int[,] matrix = new int[3, 4];

Random rnd = new Random();

// Заполнение

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 4; j++)

{

matrix[i, j] = rnd.Next(1, 10);

}

}

// Вывод

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 4; j++)

{

Console.Write(matrix[i, j] + " ");

}

Console.WriteLine();

}

1. В одномерном массиве, состоящем из N вещественных элементов, вычислить сумму положительных элементов массива.

Алгоритм решения:

* Создаем массив вещественных чисел.
* Проходим по каждому элементу массива.
* Проверяем, является ли элемент положительным (> 0).
* Если да — добавляем его к сумме.
* Выводим результат.

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

// Заполнение массива

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

{

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

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

}

// Вычисление суммы положительных элементов

double sum = 0;

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

{

if (array[i] > 0)

{

sum += array[i];

}

}

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

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ДОМ (адрес – string, количество комнат – int, площадь – double). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: площадь домов. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

// Класс "Дом" (House) с тремя полями: адрес, количество комнат и площадь

class House

{

public string Address { get; set; } // Адрес дома (строка)

public int Rooms { get; set; } // Количество комнат (целое число)

public double Area { get; set; } // Площадь дома (вещественное число)

// Метод для вывода информации о доме

public void DisplayInfo()

{

Console.WriteLine($"Адрес: {Address}, Комнат: {Rooms}, Площадь: {Area} м²");

}

}

class Program

{

static void Main()

{

// Запрашиваем у пользователя количество домов

Console.Write("Введите количество домов: ");

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

List<House> houses = new List<House>(); // Создаем список домов

// Вводим данные для каждого дома

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

{

House house = new House(); // Создаем новый объект "Дом"

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

house.Address = Console.ReadLine();

Console.Write($"Введите количество комнат: ");

house.Rooms = int.Parse(Console.ReadLine());

Console.Write($"Введите площадь дома (в м²): ");

house.Area = double.Parse(Console.ReadLine());

houses.Add(house); // Добавляем дом в список

}

// Сортируем дома по убыванию площади

houses.Sort((x, y) => y.Area.CompareTo(x.Area));

// Выводим отсортированный список

Console.WriteLine("\nСписок домов (отсортирован по убыванию площади):");

foreach (var house in houses)

{

house.DisplayInfo(); // Выводим информацию о каждом доме

}

}

}}

**Билет 14**

1. Опишите, как выполняется описание одномерного массива, обращение к элементам массиваЖ, его заполнение, вывод на экран. Приведите примеры

***Описание массива***:

В C# одномерный массив описывается следующим образом:

тип\_данных[] имя\_массива = new тип\_данных[размер];

или с инициализацией:

тип\_данных[] имя\_массива = { элемент1, элемент2, ..., элементN };

***Обращение к элементам***:

Доступ к элементам массива осуществляется по индексу (начиная с 0):

элемент = имя\_массива[индекс]; // чтение

имя\_массива[индекс] = значение; // запись

***Заполнение массива***:

Массив можно заполнить вручную, с клавиатуры или случайными числами.

Вывод массива:

Для вывода используется цикл (например, for или foreach).

Пример:

using System;

class Program

{

static void Main()

{

// Описание массива

int[] numbers = new int[5]; // массив из 5 целых чисел

// Заполнение массива

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

{

numbers[i] = i \* 2; // пример заполнения

}

// Вывод массива

Console.WriteLine("Элементы массива:");

foreach (int num in numbers)

{

Console.Write(num + " ");

}

}

}

1. В одномерном массиве, состоящем из N вещественных элементов, вычислить количество отрицательных элементов массива

using System;

class Program

{

static void Main()

{

// Ввод размера массива

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

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

// Создание массива

double[] array = new double[N];

// Заполнение массива

Console.WriteLine("Введите элементы массива:");

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

{

Console.Write($"Элемент {i + 1}: ");

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

}

// Подсчёт отрицательных элементов

int negativeCount = 0;

foreach (double num in array)

{

if (num < 0)

{

negativeCount++;

}

}

// Вывод результата

Console.WriteLine($"Количество отрицательных элементов: {negativeCount}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОЗЕРО (название – string, площадь – double, местонахождение – string). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: название. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

using System.Linq;

class Lake

{

public string Name { get; set; }

public double Area { get; set; }

public string Location { get; set; }

public Lake(string name, double area, string location)

{

Name = name;

Area = area;

Location = location;

}

public override string ToString()

{

return $"Название: {Name}, Площадь: {Area}, Местонахождение: {Location}";

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество озер N: ");

int N = Convert.ToInt32(Console.ReadLine());

List<Lake> lakes = new List<Lake>();

// Заполнение списка

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

{

Console.WriteLine($"\nВвод данных для озера {i + 1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Площадь: ");

double area = Convert.ToDouble(Console.ReadLine());

Console.Write("Местонахождение: ");

string location = Console.ReadLine();

lakes.Add(new Lake(name, area, location));

}

// Сортировка списка по названию по убыванию

lakes.Sort((x, y) => string.Compare(y.Name, x.Name));

// Вывод отсортированного списка

Console.WriteLine("\nОтсортированный список озер по названию (по убыванию):");

foreach (Lake lake in lakes)

{

Console.WriteLine(lake);

}

}

}

**Билет 15**

1. Опишите класс Array: назначение, свойства, методы. Приведите примеры.

Класс Array в C# — это базовый класс для всех массивов. Он предоставляет методы для создания, изменения, сортировки и поиска в одномерных и многомерных массивах.

Основные свойства:

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| Length | Возвращает общее количество элементов во всех измерениях массива. |
| Rank | Возвращает размерность массива (1 для одномерного, 2 для двумерного и т. д.). |
| IsReadOnly | Возвращает false (массивы в C# по умолчанию изменяемые). |
| IsFixedSize | Всегда true (размер массива фиксирован после создания). |

Основные методы:

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Пример** |
| GetValue(int index) | Получает элемент по индексу. | arr.GetValue(0) |
| SetValue(object value, int index) | Устанавливает значение элемента. | arr.SetValue(10, 0) |
| Sort(Array array) | Сортирует элементы в одномерном массиве. | Array.Sort(arr) |
| BinarySearch(Array array, object value) | Ищет элемент в отсортированном массиве. | Array.BinarySearch(arr, 5) |
| Copy(Array source, Array dest, int length) | Копирует часть массива в другой. | Array.Copy(arr1, arr2, 3) |
| IndexOf(Array array, object value) | Возвращает индекс первого вхождения элемента. | Array.IndexOf(arr, "A") |
| Reverse(Array array) | Переворачивает массив. | Array.Reverse(arr) |
| Resize<T>(ref T[] array, int newSize) | Изменяет размер одномерного массива. | Array.Resize(ref arr, 10) |

Примеры использования Array в C#

1. Создание и заполнение массива

int[] numbers = new int[5] { 10, 20, 30, 40, 50 };

2. Доступ к элементам

Console.WriteLine(numbers[0]); // Выводит 10

numbers[1] = 25; // Изменяет второй элемент

3. Сортировка

Array.Sort(numbers); // По возрастанию: [10, 20, 25, 30, 40, 50]

Array.Reverse(numbers); // По убыванию: [50, 40, 30, 25, 20, 10]

4. Поиск элемента

int index = Array.IndexOf(numbers, 30); // Возвращает 2

bool contains = Array.Exists(numbers, x => x == 25); // true

5. Копирование массива

int[] copy = new int[numbers.Length];

Array.Copy(numbers, copy, numbers.Length);

6. Изменение размера

Array.Resize(ref numbers, 10); // Теперь массив вмещает 10 элементов

1. Данные о температуре воздуха за декабрь хранятся в вещественном массиве. Определить, сколько раз температура была выше средней за этот месяц.

using System;

using System.Linq;

class Program

{

static void Main()

{

// Создаем массив для хранения температур за декабрь (31 день)

double[] temperatures = new double[31];

Random rand = new Random();

// Заполняем массив случайными значениями от -10 до +10 градусов

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

{

temperatures[i] = Math.Round(rand.NextDouble() \* 20 - 10, 1); // округляем до 1 знака после запятой

}

// Выводим все температуры для наглядности

Console.WriteLine("Температуры за декабрь:");

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

{

Console.WriteLine($"День {i + 1}: {temperatures[i]}°C");

}

// Находим среднюю температуру

double average = Math.Round(temperatures.Average(), 1);

Console.WriteLine($"\nСредняя температура за декабрь: {average}°C");

// Подсчитываем дни с температурой выше средней

int count = temperatures.Count(t => t > average);

// без Linq Подсчитываем дни с температурой выше средней

int count = 0;

foreach (var t in temperatures)

{

if (t > average)

{

count++;

}

}

Console.WriteLine($"\nТемпература была выше средней {count} раз");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ТОВАР (наименование – string, количество – int, стоимость – float). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: наименование. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

class Product

{

// Свойства

public string Name { get; set; } // Наименование

public int Quantity { get; set; } // Количество

public float Price { get; set; } // Стоимость

// Конструктор

public Product(string name, int quantity, float price)

{

Name = name;

Quantity = quantity;

Price = price;

}

// Метод для вывода информации

public override string ToString()

{

return $"Название: {Name}, Количество: {Quantity}, Цена: {Price:F2} руб.";

}

}

class Program

{

static void Main()

{

List<Product> products = new List<Product>();

Console.Write("Введите количество товаров: ");

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

// Ввод данных

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

{

Console.WriteLine($"\nТовар {i + 1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Количество: ");

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

Console.Write("Цена: ");

float price = float.Parse(Console.ReadLine());

products.Add(new Product(name, quantity, price));

}

// Сортировка по убыванию названия (без IComparable, через лямбда-выражение)

products.Sort((p1, p2) => p2.Name.CompareTo(p1.Name));

// Вывод отсортированного списка

Console.WriteLine("\nОтсортированный список товаров (по убыванию названия):");

foreach (Product product in products)

{

Console.WriteLine(product);

}

}

}

**Билет 16**

1. Опишите класс Regex: назначение, свойства, методы. Приведите примеры

Класс Regex в C# предназначен для работы с регулярными выражениями. Он позволяет выполнять поиск, замену и другие операции с текстом по заданному шаблону.

Свойства:

Options - возвращает параметры, которые были переданы в конструктор Regex

RightToLeft - указывает, выполняется ли поиск справа налево

ToString() - возвращает строку регулярного выражения

Основные методы:

IsMatch() - проверяет, содержит ли строка соответствие шаблону

Match() - находит первое соответствие шаблону

Matches() - находит все соответствия шаблону

Replace() - заменяет соответствия шаблону на указанную строку

Split() - разбивает строку на части по соответствиям шаблону

Примеры:

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

// Проверка соответствия шаблону

string pattern = @"^\d{3}-\d{2}-\d{4}$"; // формат SSN

Regex regex = new Regex(pattern);

string ssn = "123-45-6789";

Console.WriteLine(regex.IsMatch(ssn)); // True

// Замена текста

string text = "Today is 2023-10-15";

string newText = Regex.Replace(text, @"\d{4}-\d{2}-\d{2}", "DATE");

Console.WriteLine(newText); // Today is DATE

// Поиск всех соответствий

MatchCollection matches = Regex.Matches("A1B2C3", @"\d");

foreach (Match match in matches)

{

Console.WriteLine(match.Value); // 1, 2, 3

}

}

}

1. В одномерном массиве, состоящем из N вещественных элементов, вычислить максимальный элемент массива.

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

// Ввод элементов массива

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

{

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

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

}

// Поиск максимального элемента

double max = array[0];

for (int i = 1; i < N; i++)

{

if (array[i] > max)

{

max = array[i];

}

}

Console.WriteLine($"Максимальный элемент массива: {max}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: АВТОБУС (модель – string, мощность – double, количество мест – int). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: модель. Отсортированный список вывести на экран.

using System;

using System.Collections.Generic;

class Bus

{

public string Model { get; set; }

public double Power { get; set; }

public int Seats { get; set; }

// Конструктор

public Bus(string model, double power, int seats)

{

Model = model;

Power = power;

Seats = seats;

}

// Метод для вывода информации

public void PrintInfo()

{

Console.WriteLine($"Модель: {Model}, Мощность: {Power} л.с., Мест: {Seats}");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество автобусов N: ");

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

List<Bus> buses = new List<Bus>();

// Ввод данных

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

{

Console.WriteLine($"\nВведите данные автобуса {i + 1}:");

Console.Write("Модель: ");

string model = Console.ReadLine();

Console.Write("Мощность: ");

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

Console.Write("Количество мест: ");

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

buses.Add(new Bus(model, power, seats));

}

// Сортировка по убыванию по полю Model

buses.Sort((bus1, bus2) => string.Compare(bus2.Model, bus1.Model));

// Вывод отсортированного списка

Console.WriteLine("\nОтсортированный список автобусов:");

foreach (var bus in buses)

{

bus.PrintInfo();

}

}

}

**Пояснения**

В данном коде bus1 и bus2 - это параметры лямбда-выражения, используемого для сортировки списка автобусов. Давайте разберем подробнее:

В строке:

buses.Sort((bus1, bus2) => string.Compare(bus2.Model, bus1.Model));

bus1 и bus2 - это два объекта класса Bus, которые сравниваются в процессе сортировки.

Это часть лямбда-выражения, которое передается в метод Sort() для определения порядка сортировки.

Метод Sort() автоматически передает пары объектов из списка в эту функцию сравнения, чтобы определить их относительный порядок.

Лямбда-выражение возвращает:

Отрицательное число, если bus1 должен идти перед bus2

Положительное число, если bus1 должен идти после bus2

Ноль, если они равны

В данном случае мы используем string.Compare(bus2.Model, bus1.Model) вместо обычного string.Compare(bus1.Model, bus2.Model), что дает нам сортировку по убыванию (от Z до A).

Альтернативный вариант без лямбда-выражения мог бы выглядеть так:

buses.Sort(CompareBuses);

static int CompareBuses(Bus x, Bus y)

{

return string.Compare(y.Model, x.Model);

}

Где x и y выполняют ту же роль, что и bus1 и bus2 - это просто имена параметров для двух сравниваемых объектов.

**Билет 17**

1. Опишите синтаксис и использование структуры, отличие класса от структуры. Приведите примеры

Структура (struct) в C# — это тип значения, который обычно используется для небольших объектов, не требующих наследования.

Класс (class) — это ссылочный тип, поддерживающий наследование и более сложное поведение.

Основные отличия:

Тип данных: структуры — value type (хранятся в стеке), классы — reference type (хранятся в куче).

Наследование: структуры не поддерживают наследование (кроме System.ValueType).

Инициализация: в структурах нельзя определить конструктор по умолчанию (он всегда есть и инициализирует поля нулями).

Присваивание: при копировании структуры создается копия всех данных, а при копировании класса — копируется только ссылка.

Структуры - значимые типы, классы - ссылочные

Структуры хранятся в стеке, классы - в куче

Структуры не поддерживают наследование

Структуры нельзя создать без инициализации полей

Пример структуры:

struct Point

{

public int X;

public int Y;

public Point(int x, int y)

{

X = x;

Y = y;

}

public void Print()

{

Console.WriteLine($"X: {X}, Y: {Y}");

}

}

// Использование

Point p1 = new Point(10, 20);

p1.Print();

Пример класса:

class PointClass

{

public int X { get; set; }

public int Y { get; set; }

public PointClass(int x, int y)

{

X = x;

Y = y;

}

public void Print()

{

Console.WriteLine($"X: {X}, Y: {Y}");

}

}

// Использование

PointClass pc = new PointClass(5, 15);

pc.Print();

1. Задать одномерный массив, состоящий из N целых элементов. Найти номер минимального и максимального элементов массива

using System;

class Program

{

static void Main()

{

int[] array = { 5, 2, 9, 1, 5, 6 };

int minIndex = 0, maxIndex = 0;

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

{

if (array[i] < array[minIndex])

minIndex = i;

if (array[i] > array[maxIndex])

maxIndex = i;

}

Console.WriteLine($"Минимальный элемент: индекс {minIndex}, значение {array[minIndex]}");

Console.WriteLine($"Максимальный элемент: индекс {maxIndex}, значение {array[maxIndex]}");

}

}

1. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ФИРМА (название – string, адрес – string, год основания – int). Условие: посчитать и вывести на экран количество фирм имеющих заданный год основания (год основания ввести с клавиатуры).

using System;

using System.Collections.Generic;

class Firm

{

public string Name { get; set; }

public string Address { get; set; }

public int YearFounded { get; set; }

public Firm(string name, string address, int yearFounded)

{

Name = name;

Address = address;

YearFounded = yearFounded;

}

public void DisplayInfo()

{

Console.WriteLine($"Название: {Name}, Адрес: {Address}, Год основания: {YearFounded}");

}

}

class Program

{

static void Main()

{

List<Firm> firms = new List<Firm>();

Console.Write("Введите количество фирм: ");

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

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

{

Console.WriteLine($"Фирма {i + 1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Адрес: ");

string address = Console.ReadLine();

Console.Write("Год основания: ");

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

firms.Add(new Firm(name, address, year));

}

Console.Write("Введите год для поиска: ");

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

int count = 0;

foreach (var firm in firms)

{

if (firm.YearFounded == searchYear)

{

firm.DisplayInfo();

count++;

}

}

Console.WriteLine($"Количество фирм с годом основания {searchYear}: {count}");

}

}

**Билет 18**

1. Опишите синтаксис и использование перечисления. Приведите примеры

Перечисление (enum) в C# — это тип данных, который представляет собой набор именованных констант.

Синтаксис:

enum ИмяПеречисления

{

Значение1,

Значение2,

...

}

Пример использования:

using System;

enum DayOfWeek

{

Monday, // 0 (по умолчанию)

Tuesday, // 1

Wednesday, // 2

Thursday, // 3

Friday, // 4

Saturday, // 5

Sunday // 6

}

class Program

{

static void Main()

{

DayOfWeek today = DayOfWeek.Friday;

Console.WriteLine($"Сегодня: {today} ({(int)today})");

if (today == DayOfWeek.Saturday || today == DayOfWeek.Sunday)

Console.WriteLine("Выходной!");

else

Console.WriteLine("Рабочий день.");

}

}

Вывод:

Сегодня: Friday (4)

Рабочий день.

Особенности:

Можно задавать свои числовые значения:

enum Status { Active = 1, Inactive = 0, Pending = 2 }

Можно использовать в switch-case.

По умолчанию хранятся как int, но можно изменить (byte, long и др.):

enum SmallEnum : byte { Low = 1, Medium = 2, High = 3 }

2. В одномерном массиве, состоящем из N вещественных элементов, вычислить количество элементов массива, лежащих в диапазоне от А до В.

using System;

class Program

{

static void Main()

{

double[] array = { 1.5, 2.3, 4.0, 5.7, 3.2, 6.1 };

Console.Write("Введите A: ");

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

Console.Write("Введите B: ");

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

int count = 0;

foreach (double num in array)

{

if (num >= A && num <= B)

count++;

}

Console.WriteLine($"Количество элементов от {A} до {B}: {count}");

}

}

3. Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ПРИБОР (наименование – string, класс – string, мощность – double). Условие: посчитать и вывести на экран количество приборов имеющих мощность больше заданного значения (мощность прибора ввести с клавиатуры).

using System;

using System.Collections.Generic;

class Device

{

public string Name { get; set; }

public string Class { get; set; }

public double Power { get; set; }

public Device(string name, string deviceClass, double power)

{

Name = name;

Class = deviceClass;

Power = power;

}

public void DisplayInfo()

{

Console.WriteLine($"Прибор: {Name}, Класс: {Class}, Мощность: {Power} Вт");

}

}

class Program

{

static void Main()

{

List<Device> devices = new List<Device>();

Console.Write("Введите количество приборов: ");

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

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

{

Console.WriteLine($"Прибор {i + 1}:");

Console.Write("Наименование: ");

string name = Console.ReadLine();

Console.Write("Класс: ");

string deviceClass = Console.ReadLine();

Console.Write("Мощность (Вт): ");

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

devices.Add(new Device(name, deviceClass, power));

}

Console.Write("Введите минимальную мощность для поиска: ");

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

int count = 0;

foreach (var device in devices)

{

if (device.Power > minPower)

{

device.DisplayInfo();

count++;

}

}

Console.WriteLine($"Количество приборов с мощностью > {minPower} Вт: {count}");

}

}

**Вариант 19**

**1 Раскройте сущность понятия «делегат». Опишите синтаксис делегата, действия над ними. Приведите примеры.**

**Делегат** — это тип, представляющий собой ссылку на метод. Делегаты позволяют вызывать методы, не зная их имени во время компиляции. Они особенно полезны для реализации событий и обратных вызовов (callbacks).

**Синтаксис объявления делегата:**

delegate void MyDelegate(string message);

**Пример использования делегата:**

using System;

delegate void MyDelegate(string message);

class Program

{

static void Hello(string msg)

{

Console.WriteLine("Hello: " + msg);

}

static void Bye(string msg)

{

Console.WriteLine("Bye: " + msg);

}

static void Main()

{

MyDelegate del = Hello; // Присваиваем метод Hello

del += Bye; // Добавляем метод Bye

del("World"); // Вызываются оба метода

del -= Hello; // Удаляем метод Hello

del("Again"); // Остался только Bye

}

}

**Операции над делегатами:**

* += — добавление метода к списку вызова.
* -= — удаление метода.
* Invoke() или просто delegateName() — вызов методов, связанных с делегатом.

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить количество элементов массива, больших 100.**

using System;

class Program

{

static void Main()

{

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

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

double[] arr = new double[n];

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

{

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

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

}

int count = 0;

foreach (double val in arr)

{

if (val > 100)

count++;

}

Console.WriteLine($"Количество элементов больше 100: {count}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: МОНИТОР (фирма – string, модель – string, диагональ – float). Условие: посчитать и вывести на экран количество объектов, имеющих заданную модель (модель ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class MONITOR

{

private string firm;

private string model;

private float diagonal;

// Конструктор

public MONITOR(string firm, string model, float diagonal)

{

this.firm = firm;

this.model = model;

this.diagonal = diagonal;

}

// Методы для установки и получения значений

public string GetFirm() => firm;

public string GetModel() => model;

public float GetDiagonal() => diagonal;

public void SetFirm(string f) => firm = f;

public void SetModel(string m) => model = m;

public void SetDiagonal(float d) => diagonal = d;

}

class Program

{

static void Main()

{

Console.Write("Введите количество мониторов: ");

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

List<MONITOR> monitors = new List<MONITOR>();

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

{

Console.WriteLine($"\nМонитор {i + 1}:");

Console.Write("Фирма: ");

string firm = Console.ReadLine();

Console.Write("Модель: ");

string model = Console.ReadLine();

Console.Write("Диагональ: ");

float diagonal = float.Parse(Console.ReadLine());

monitors.Add(new MONITOR(firm, model, diagonal));

}

Console.Write("\nВведите модель для поиска: ");

string searchModel = Console.ReadLine();

int count = 0;

foreach (MONITOR m in monitors)

{

if (m.GetModel() == searchModel)

count++;

}

Console.WriteLine($"\nКоличество мониторов модели {searchModel}: {count}");

}

}

**Вариант 20**

**1 Раскройте сущность понятия «файл». Опишите виды файлов, алгоритм чтения и записи файлов. Приведите примеры.**

**Файл** — это именованная область хранения данных на долговременном носителе (например, жёстком диске), содержащая информацию, которую можно прочитать или записать.

**Виды файлов:**

* **Текстовые файлы** — содержат символьную информацию (например, .txt, .csv, .xml).
* **Бинарные файлы** — содержат данные в формате, понятном программе, но не читаемом человеком напрямую (например, .exe, .jpg, .dat).

**Алгоритм работы с файлами в C#:**

Чтение файла:

1. Открыть файл.
2. Прочитать содержимое.
3. Закрыть файл.

Запись в файл:

1. Открыть или создать файл.
2. Записать данные.
3. Закрыть файл

**Примеры:**

using System;

using System.IO;

class Program

{

static void Main()

{

string path = "example.txt";

// Запись в файл

File.WriteAllText(path, "Привет, файл!");

// Чтение из файла

string content = File.ReadAllText(path);

Console.WriteLine("Содержимое файла: " + content);

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить количество элементов массива, не равных нулю.**

using System;

class Program

{

static void Main()

{

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

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

double[] arr = new double[n];

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

{

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

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

}

int count = 0;

foreach (double val in arr)

{

if (val != 0)

count++;

}

Console.WriteLine($"Количество элементов, не равных нулю: {count}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов, удовлетворяющих заданному условию. Имя класса: КНИГА (автор – string, название книги – string, год издания – int). Условие: посчитать и вывести на экран количество книг, имеющих заданного автора (имя автора ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class KNIGA

{

private string author;

private string title;

private int year;

public KNIGA(string author, string title, int year)

{

this.author = author;

this.title = title;

this.year = year;

}

public string GetAuthor() => author;

public string GetTitle() => title;

public int GetYear() => year;

public void SetAuthor(string a) => author = a;

public void SetTitle(string t) => title = t;

public void SetYear(int y) => year = y;

}

class Program

{

static void Main()

{

Console.Write("Введите количество книг: ");

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

List<KNIGA> books = new List<KNIGA>();

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

{

Console.WriteLine($"\nКнига {i + 1}:");

Console.Write("Автор: ");

string author = Console.ReadLine();

Console.Write("Название: ");

string title = Console.ReadLine();

Console.Write("Год издания: ");

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

books.Add(new KNIGA(author, title, year));

}

Console.Write("\nВведите имя автора для поиска: ");

string searchAuthor = Console.ReadLine();

int count = 0;

foreach (KNIGA book in books)

{

if (book.GetAuthor().Equals(searchAuthor, StringComparison.OrdinalIgnoreCase))

count++;

}

Console.WriteLine($"\nКоличество книг автора \"{searchAuthor}\": {count}");

}

}

**Вариант 21**

**1 Дайте определение исключительной ситуации, изложите способы реагирования на исключительные ситуации на языке C#. Приведите примеры.**

**Исключительная ситуация (исключение)** — это ошибка, возникающая во время выполнения программы, которая нарушает нормальный ход выполнения (например, деление на ноль, выход за границы массива, ошибка при чтении файла).

**Обработка исключений в C#** выполняется с помощью конструкции **try-catch-finally**.

**Синтаксис:**

try

{

// Код, где может возникнуть исключение

}

catch (ExceptionType e)

{

// Обработка исключения

}

finally

{

// Код, который выполнится в любом случае

}

**Пример:**

using System;

class Program

{

static void Main()

{

try

{

Console.Write("Введите число: ");

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

Console.WriteLine("Квадрат: " + (num \* num));

}

catch (FormatException)

{

Console.WriteLine("Ошибка: введено не число.");

}

catch (Exception ex)

{

Console.WriteLine("Общая ошибка: " + ex.Message);

}

finally

{

Console.WriteLine("Работа завершена.");

}

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить сумму элементов массива до первого нулевого элемента.**

using System;

class Program

{

static void Main()

{

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

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

double[] arr = new double[n];

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

{

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

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

}

double sum = 0;

foreach (double val in arr)

{

if (val == 0)

break;

sum += val;

}

Console.WriteLine($"Сумма элементов до первого нулевого: {sum}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов, удовлетворяющих заданному условию. Имя класса: ЧЕЛОВЕК (имя – string, пол – char, год рождения – int). Условие: посчитать и вывести на экран количество человек, имеющих заданное имя (имя ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class CHELOVEK

{

private string name;

private char gender;

private int birthYear;

public CHELOVEK(string name, char gender, int birthYear)

{

this.name = name;

this.gender = gender;

this.birthYear = birthYear;

}

public string GetName() => name;

public char GetGender() => gender;

public int GetBirthYear() => birthYear;

public void SetName(string n) => name = n;

public void SetGender(char g) => gender = g;

public void SetBirthYear(int y) => birthYear = y;

}

class Program

{

static void Main()

{

Console.Write("Введите количество людей: ");

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

List<CHELOVEK> people = new List<CHELOVEK>();

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

{

Console.WriteLine($"\nЧеловек {i + 1}:");

Console.Write("Имя: ");

string name = Console.ReadLine();

Console.Write("Пол (м/ж): ");

char gender = char.Parse(Console.ReadLine());

Console.Write("Год рождения: ");

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

people.Add(new CHELOVEK(name, gender, year));

}

Console.Write("\nВведите имя для поиска: ");

string searchName = Console.ReadLine();

int count = 0;

foreach (CHELOVEK person in people)

{

if (person.GetName().Equals(searchName, StringComparison.OrdinalIgnoreCase))

count++;

}

Console.WriteLine($"\nКоличество людей с именем \"{searchName}\": {count}");

}

}

**Вариант 22**

**1 Раскройте сущность понятия «многопоточность». Опишите основы многопоточной обработки, класс Thread, работу с потоками. Приведите примеры.**

**Многопоточность** — это способность программы выполнять несколько операций одновременно с помощью **параллельного выполнения потоков**. Каждый поток — это независимая последовательность инструкций.

**Класс Thread:**

System.Threading.Thread — основной класс для создания потоков в .NET. Потоки позволяют выполнять задачи параллельно, что особенно полезно для задач с высокой нагрузкой (например, вычисления, ввод-вывод, ожидание).

**Создание и запуск потока:**

using System;

using System.Threading;

class Program

{

static void PrintNumbers()

{

for (int i = 1; i <= 5; i++)

{

Console.WriteLine($"Поток: {i}");

Thread.Sleep(500); // Пауза 0.5 сек

}

}

static void Main()

{

Thread t = new Thread(PrintNumbers);

t.Start();

for (int i = 1; i <= 5; i++)

{

Console.WriteLine($"Главный: {i}");

Thread.Sleep(700);

}

t.Join(); // Ожидание завершения потока

}

}

**2 В одномерном массиве, состоящем из N целочисленных элементов, вычислить количество элементов массива, больших 8 и кратных 7.**

using System;

class Program

{

static void Main()

{

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

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

int[] arr = new int[n];

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

{

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

arr[i] = int.Parse(Console.ReadLine());

}

int count = 0;

foreach (int val in arr)

{

if (val > 8 && val % 7 == 0)

count++;

}

Console.WriteLine($"Количество элементов >8 и кратных 7: {count}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ТОВАР (наименование – string, количество – int, стоимость – double). Условие: посчитать и вывести на экран количество товаров, имеющих заданное наименование (наименование товара ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class TOVAR

{

private string name;

private int quantity;

private double price;

public TOVAR(string name, int quantity, double price)

{

this.name = name;

this.quantity = quantity;

this.price = price;

}

public string GetName() => name;

public int GetQuantity() => quantity;

public double GetPrice() => price;

public void SetName(string n) => name = n;

public void SetQuantity(int q) => quantity = q;

public void SetPrice(double p) => price = p;

}

class Program

{

static void Main()

{

Console.Write("Введите количество товаров: ");

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

List<TOVAR> goods = new List<TOVAR>();

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

{

Console.WriteLine($"\nТовар {i + 1}:");

Console.Write("Наименование: ");

string name = Console.ReadLine();

Console.Write("Количество: ");

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

Console.Write("Стоимость: ");

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

goods.Add(new TOVAR(name, quantity, price));

}

Console.Write("\nВведите наименование для поиска: ");

string targetName = Console.ReadLine();

int count = 0;

foreach (TOVAR item in goods)

{

if (item.GetName().Equals(targetName, StringComparison.OrdinalIgnoreCase))

count++;

}

Console.WriteLine($"\nКоличество товаров с наименованием \"{targetName}\": {count}");

}

}

**Вариант 23**

**1 Раскройте сущность технологии ADO.NET. Опишите подключение к базе данных, выполнение операторов SQL. Приведите примеры.**

**ADO.NET** — это технология доступа к данным в платформе .NET. Она позволяет работать с различными источниками данных (например, SQL Server, SQLite, MySQL) через объекты .NET.

**Основные компоненты ADO.NET:**

* SqlConnection — подключение к базе данных.
* SqlCommand — выполнение SQL-команд.
* SqlDataReader — чтение данных построчно.
* SqlDataAdapter и DataSet — для работы с данными в памяти.

**Пример подключения и выполнения запроса:**

using System;

using System.Data.SqlClient;

class Program

{

static void Main()

{

string connectionString = "Data Source=localhost;Initial Catalog=MyDatabase;Integrated Security=True";

string sql = "SELECT Name FROM Students";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand(sql, connection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine(reader["Name"]);

}

reader.Close();

}

}

}

**2 В одномерном массиве, состоящем из N целочисленных элементов, вычислить сумму элементов, кратных 8.**

using System;

class Program

{

static void Main()

{

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

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

int[] arr = new int[n];

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

{

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

arr[i] = int.Parse(Console.ReadLine());

}

int sum = 0;

foreach (int val in arr)

{

if (val % 8 == 0)

sum += val;

}

Console.WriteLine($"Сумма элементов, кратных 8: {sum}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов, удовлетворяющих заданному условию. Имя класса: УЧАЩИЙСЯ (фамилия – string, группа – string, средний балл – float). Условие: вывести фамилии учащихся, средний балл которых больше заданного (средний балл ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class UCHASHCHIYSYA

{

private string lastName;

private string group;

private float averageMark;

public UCHASHCHIYSYA(string lastName, string group, float averageMark)

{

this.lastName = lastName;

this.group = group;

this.averageMark = averageMark;

}

public string GetLastName() => lastName;

public string GetGroup() => group;

public float GetAverageMark() => averageMark;

public void SetLastName(string ln) => lastName = ln;

public void SetGroup(string g) => group = g;

public void SetAverageMark(float am) => averageMark = am;

}

class Program

{

static void Main()

{

Console.Write("Введите количество учащихся: ");

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

List<UCHASHCHIYSYA> students = new List<UCHASHCHIYSYA>();

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

{

Console.WriteLine($"\nУчащийся {i + 1}:");

Console.Write("Фамилия: ");

string lastName = Console.ReadLine();

Console.Write("Группа: ");

string group = Console.ReadLine();

Console.Write("Средний балл: ");

float avg = float.Parse(Console.ReadLine());

students.Add(new UCHASHCHIYSYA(lastName, group, avg));

}

Console.Write("\nВведите минимальный средний балл: ");

float threshold = float.Parse(Console.ReadLine());

Console.WriteLine("\nУчащиеся с баллом выше " + threshold + ":");

foreach (UCHASHCHIYSYA s in students)

{

if (s.GetAverageMark() > threshold)

Console.WriteLine(s.GetLastName());

}

}

}

**Вариант 24**

**1 Раскройте сущность понятия «делегат». Опишите синтаксис делегата, действия над ними. Приведите примеры.**

**Делегат** — это тип, представляющий собой ссылку на метод. Делегаты позволяют вызывать методы, не зная их имени во время компиляции. Они особенно полезны для реализации событий и обратных вызовов (callbacks).

**Синтаксис объявления делегата:**

delegate void MyDelegate(string message);

**Пример использования делегата:**

using System;

delegate void MyDelegate(string message);

class Program

{

static void Hello(string msg)

{

Console.WriteLine("Hello: " + msg);

}

static void Bye(string msg)

{

Console.WriteLine("Bye: " + msg);

}

static void Main()

{

MyDelegate del = Hello; // Присваиваем метод Hello

del += Bye; // Добавляем метод Bye

del("World"); // Вызываются оба метода

del -= Hello; // Удаляем метод Hello

del("Again"); // Остался только Bye

}

}

**Операции над делегатами:**

* += — добавление метода к списку вызова.
* -= — удаление метода.
* Invoke() или просто delegateName() — вызов методов, связанных с делегатом.

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить количество элементов массива, больших 100.**

using System;

class Program

{

static void Main()

{

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

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

double[] arr = new double[n];

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

{

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

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

}

int count = 0;

foreach (double val in arr)

{

if (val > 100)

count++;

}

Console.WriteLine($"Количество элементов больше 100: {count}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: МОНИТОР (фирма – string, модель – string, диагональ – float). Условие: посчитать и вывести на экран количество объектов, имеющих заданную модель (модель ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class MONITOR

{

private string firm;

private string model;

private float diagonal;

// Конструктор

public MONITOR(string firm, string model, float diagonal)

{

this.firm = firm;

this.model = model;

this.diagonal = diagonal;

}

// Методы для установки и получения значений

public string GetFirm() => firm;

public string GetModel() => model;

public float GetDiagonal() => diagonal;

public void SetFirm(string f) => firm = f;

public void SetModel(string m) => model = m;

public void SetDiagonal(float d) => diagonal = d;

}

class Program

{

static void Main()

{

Console.Write("Введите количество мониторов: ");

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

List<MONITOR> monitors = new List<MONITOR>();

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

{

Console.WriteLine($"\nМонитор {i + 1}:");

Console.Write("Фирма: ");

string firm = Console.ReadLine();

Console.Write("Модель: ");

string model = Console.ReadLine();

Console.Write("Диагональ: ");

float diagonal = float.Parse(Console.ReadLine());

monitors.Add(new MONITOR(firm, model, diagonal));

}

Console.Write("\nВведите модель для поиска: ");

string searchModel = Console.ReadLine();

int count = 0;

foreach (MONITOR m in monitors)

{

if (m.GetModel() == searchModel)

count++;

}

Console.WriteLine($"\nКоличество мониторов модели {searchModel}: {count}");

}

}

**Вариант 25**

**1 Опишите основные элементы класса LinkedList. Приведите примеры.**

Класс LinkedList<T> в C# представляет собой двусвязный список, где каждый элемент (узел) содержит:

* значение (Value)
* ссылку на следующий элемент (Next)
* ссылку на предыдущий элемент (Previous)

Основные элементы LinkedList<T>:

1. LinkedListNode<T> — узел списка, содержащий:
   * Value — хранимое значение типа T.
   * Next — ссылка на следующий узел (или null, если это последний элемент).
   * Previous — ссылка на предыдущий узел (или null, если это первый элемент).
2. Свойства:
   * First — возвращает первый узел списка.
   * Last — возвращает последний узел списка.
   * Count — возвращает количество элементов в списке.
3. Основные методы:
   * AddFirst(T value) / AddFirst(LinkedListNode<T> node) — добавляет элемент в начало списка.
   * AddLast(T value) / AddLast(LinkedListNode<T> node) — добавляет элемент в конец списка.
   * RemoveFirst() — удаляет первый элемент.
   * RemoveLast() — удаляет последний элемент.
   * Contains(T value) — проверяет наличие элемента.
   * Find(T value) — находит первый узел с указанным значением.
   * Remove(T value) — удаляет первое вхождение элемента.

**Пример использования LinkedList<T>:**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

LinkedList<string> linkedList = new LinkedList<string>();

linkedList.AddFirst("First");

linkedList.AddLast("Second");

linkedList.AddLast("Third");

Console.WriteLine("Элементы списка:");

foreach (var item in linkedList)

{

Console.WriteLine(item);

}

Console.WriteLine($"\nПервый элемент: {linkedList.First.Value}");

Console.WriteLine($"Последний элемент: {linkedList.Last.Value}");

linkedList.Remove("Second");

Console.WriteLine("\nПосле удаления 'Second':");

foreach (var item in linkedList)

{

Console.WriteLine(item);

}

Console.WriteLine($"\nСписок содержит 'Third'? {linkedList.Contains("Third")}");

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить среднее арифметическое положительных элементов.**

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

double sum = 0;

int count = 0;

// Заполнение массива

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

{

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

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

if (array[i] > 0)

{

sum += array[i];

count++;

}

}

if (count > 0)

{

double average = sum / count;

Console.WriteLine($"Среднее арифметическое положительных элементов: {average:F2}");

}

else

{

Console.WriteLine("В массиве нет положительных элементов");

}

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ПРЕПОДАВАТЕЛЬ (фамилия – string, цикловая комиссия – string, педагогический стаж – float). Условие: посчитать и вывести на экран количество объектов имеющих педагогический стаж больше заданного значения (педагогический стаж ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class Teacher

{

// Поля данных

public string LastName { get; set; }

public string Commission { get; set; }

public float TeachingExperience { get; set; }

// Конструктор

public Teacher(string lastName, string commission, float experience)

{

LastName = lastName;

Commission = commission;

TeachingExperience = experience;

}

public void DisplayInfo()

{

Console.WriteLine($"Фамилия: {LastName}, Комиссия: {Commission}, Стаж: {TeachingExperience} лет");}}

class Program

{

static void Main()

{

Console.Write("Введите количество преподавателей N: ");

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

List<Teacher> teachers = new List<Teacher>();

// Заполнение списка

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

{

Console.WriteLine($"\nПреподаватель {i+1}:");

Console.Write("Фамилия: ");

string lastName = Console.ReadLine();

Console.Write("Цикловая комиссия: ");

string commission = Console.ReadLine();

Console.Write("Педагогический стаж (лет): ");

float experience = float.Parse(Console.ReadLine());

teachers.Add(new Teacher(lastName, commission, experience));}

Console.Write("\nВведите минимальный стаж для поиска: ");

float minExperience = float.Parse(Console.ReadLine());

// Поиск преподавателей с стажем больше заданного

int count = 0;

Console.WriteLine("\nПреподаватели со стажем больше {0} лет:", minExperience);

foreach (var teacher in teachers)

{

if (teacher.TeachingExperience > minExperience)

{

teacher.DisplayInfo();

count++;

}

}

Console.WriteLine($"\nОбщее количество: {count}");

}

}

**Вариант 26**

**1 Опишите основные элементы класса List. Приведите примеры.**

**Класс**List<T> в C# представляет собой **динамический массив**, который автоматически увеличивает свою емкость при добавлении элементов.

**Основные элементы**List<T>**:**

1. **Свойства:**
   * Count — возвращает количество элементов в списке.
   * Capacity — возвращает или задает текущую емкость списка.
   * Item[int index] — доступ к элементу по индексу (например, list[0]).
2. **Основные методы:**
   * Add(T item) — добавляет элемент в конец списка.
   * Insert(int index, T item) — вставляет элемент по указанному индексу.
   * Remove(T item) — удаляет первое вхождение элемента.
   * RemoveAt(int index) — удаляет элемент по индексу.
   * Clear() — очищает список.
   * Contains(T item) — проверяет наличие элемента.
   * IndexOf(T item) — возвращает индекс первого вхождения элемента.
   * Sort() — сортирует элементы списка.

**Пример использования List<T>:**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Создание списка целых чисел

List<int> numbers = new List<int>();

// Добавление элементов

numbers.Add(10);

numbers.Add(20);

numbers.Add(30);

// Вставка элемента

numbers.Insert(1, 15); // [10, 15, 20, 30]

// Удаление элемента

numbers.Remove(20); // [10, 15, 30]

// Доступ по индексу

Console.WriteLine($"Первый элемент: {numbers[0]}");

Console.WriteLine($"Количество элементов: {numbers.Count}");

// Проверка наличия элемента

Console.WriteLine($"Список содержит 15? {numbers.Contains(15)}");

// Вывод всех элементов

Console.WriteLine("\nВсе элементы списка:");

foreach (var num in numbers)

{

Console.WriteLine(num);

}

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить номер максимального элемента массива.**

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

// Заполнение массива

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

{

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

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

}

// Поиск индекса максимального элемента

int maxIndex = 0;

for (int i = 1; i < N; i++)

{

if (array[i] > array[maxIndex])

{

maxIndex = i;

}

}

Console.WriteLine($"Индекс максимального элемента: {maxIndex}");

Console.WriteLine($"Значение максимального элемента: {array[maxIndex]}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ФИРМА (название – string, адрес – string, год основания – int). Условие: посчитать и вывести на экран количество объектов имеющих заданный адрес (ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class Company

{

// Поля данных

public string Name { get; set; }

public string Address { get; set; }

public int YearFounded { get; set; }

// Конструктор

public Company(string name, string address, int year)

{

Name = name;

Address = address;

YearFounded = year;

}

// Метод для вывода информации

public void DisplayInfo()

{

Console.WriteLine($"Название: {Name}, Адрес: {Address}, Год основания: {YearFounded}");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество фирм N: ");

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

List<Company> companies = new List<Company>();

// Заполнение списка

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

{

Console.WriteLine($"\nФирма {i + 1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Адрес: ");

string address = Console.ReadLine();

Console.Write("Год основания: ");

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

companies.Add(new Company(name, address, year));

}

Console.Write("\nВведите адрес для поиска: ");

string searchAddress = Console.ReadLine();

// Поиск фирм с заданным адресом

int count = 0;

Console.WriteLine($"\nФирмы с адресом '{searchAddress}':");

foreach (var company in companies)

{

if (company.Address == searchAddress)

{

company.DisplayInfo();

count++;

}

}

Console.WriteLine($"\nОбщее количество: {count}");

}

}

**Вариант 27**

**1 Опишите основные элементы класса Stack. Приведите примеры.**

Класс Stack<T> в C# представляет собой коллекцию, работающую по принципу LIFO (Last In, First Out) - последний вошедший элемент выходит первым.

Основные элементы Stack<T>:

1. Свойства:
   * Count — возвращает количество элементов в стеке.
2. Основные методы:
   * Push(T item) — добавляет элемент на вершину стека.
   * Pop() — удаляет и возвращает элемент с вершины стека.
   * Peek() — возвращает элемент с вершины стека без удаления.
   * Contains(T item) — проверяет наличие элемента в стеке.
   * Clear() — очищает стек.

**Пример использования Stack<T>:**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Создание стека строк

Stack<string> stack = new Stack<string>();

// Добавление элементов в стек

stack.Push("Первый");

stack.Push("Второй");

stack.Push("Третий");

// Вывод количества элементов

Console.WriteLine($"Количество элементов в стеке: {stack.Count}");

// Просмотр верхнего элемента без удаления

Console.WriteLine($"Верхний элемент: {stack.Peek()}");

// Извлечение элементов из стека

Console.WriteLine("\nИзвлечение элементов:");

while (stack.Count > 0)

{

Console.WriteLine(stack.Pop());

}

// Проверка пустоты стека

Console.WriteLine($"\nСтек пуст? {stack.Count == 0}");

}

}

**2 Данные о температуре воздуха за ноябрь хранятся в вещественном массиве. Определить, сколько раз температура опускалась ниже – N градусов.**

using System;

class Program

{

static void Main()

{

// Массив температур за ноябрь (примерные данные)

double[] temperatures = { -2.5, -5.1, -1.8, -7.3, -3.6, -8.0, -4.2, -6.7, -9.1, -3.0 };

Console.Write("Введите пороговую температуру N: ");

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

int count = 0;

foreach (double temp in temperatures)

{

if (temp < -N)

{

count++;

}

}

Console.WriteLine($"Температура опускалась ниже -{N} градусов {count} раз(а).");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОБУВЬ (производитель – string, размер – double, цена – double). Условие: вывести на экран обувь имеющую заданного размера (размер обуви ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class Shoe

{

// Поля данных

public string Manufacturer { get; set; }

public double Size { get; set; }

public double Price { get; set; }

// Конструктор

public Shoe(string manufacturer, double size, double price)

{

Manufacturer = manufacturer;

Size = size;

Price = price;

}

// Метод для вывода информации

public void DisplayInfo()

{

Console.WriteLine($"Производитель: {Manufacturer}, Размер: {Size}, Цена: {Price} руб.");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество пар обуви N: ");

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

List<Shoe> shoes = new List<Shoe>();

// Заполнение списка

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

{

Console.WriteLine($"\nПара обуви {i + 1}:");

Console.Write("Производитель: ");

string manufacturer = Console.ReadLine();

Console.Write("Размер: ");

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

Console.Write("Цена (руб.): ");

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

shoes.Add(new Shoe(manufacturer, size, price));

}

Console.Write("\nВведите размер для поиска: ");

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

// Поиск обуви заданного размера

Console.WriteLine($"\nОбувь размера {searchSize}:");

foreach (var shoe in shoes)

{

if (shoe.Size == searchSize)

{

shoe.DisplayInfo();

}

}

}

}

**Вариант 28**

**1 Опишите основные элементы класса Queue. Приведите примеры.**

Класс Queue<T> в C# представляет собой коллекцию, работающую по принципу FIFO (First In, First Out) - первый вошедший элемент выходит первым.

Основные элементы Queue<T>:

1. Свойства:
   * Count — возвращает количество элементов в очереди.
2. Основные методы:
   * Enqueue(T item) — добавляет элемент в конец очереди.
   * Dequeue() — удаляет и возвращает элемент из начала очереди.
   * Peek() — возвращает элемент из начала очереди без удаления.
   * Contains(T item) — проверяет наличие элемента в очереди.
   * Clear() — очищает очередь.

**Пример использования Queue<T>:**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

// Создание очереди строк

Queue<string> queue = new Queue<string>();

// Добавление элементов в очередь

queue.Enqueue("Первый");

queue.Enqueue("Второй");

queue.Enqueue("Третий");

// Вывод количества элементов

Console.WriteLine($"Количество элементов в очереди: {queue.Count}");

// Просмотр первого элемента без удаления

Console.WriteLine($"Первый элемент: {queue.Peek()}");

// Извлечение элементов из очереди

Console.WriteLine("\nИзвлечение элементов:");

while (queue.Count > 0)

{

Console.WriteLine(queue.Dequeue());

}

// Проверка пустоты очереди

Console.WriteLine($"\nОчередь пуста? {queue.Count == 0}");

}

}

**2 В одномерном массиве, состоящем из N целочисленных элементов, вычислить сумму элементов массива, расположенных до последнего положительного элемента.**

using System;

class Program

{

static void Main()

{

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

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

int[] array = new int[N];

// Заполнение массива

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

{

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

array[i] = int.Parse(Console.ReadLine());

}

// Находим индекс последнего положительного элемента

int lastPositiveIndex = -1;

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

{

if (array[i] > 0)

{

lastPositiveIndex = i;

}

}

// Вычисляем сумму элементов до последнего положительного

int sum = 0;

if (lastPositiveIndex != -1)

{

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

{

sum += array[i];

}

Console.WriteLine($"Сумма элементов до последнего положительного: {sum}");

}

else

{

Console.WriteLine("В массиве нет положительных элементов");

}

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ВОДИТЕЛЬ (фамилия – string, категория – char, стаж вождения – double). Условие: название водителей имеющих заданный стаж вождения (стаж вождения ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class Driver

{

// Поля данных

public string LastName { get; set; }

public char Category { get; set; }

public double DrivingExperience { get; set; }

// Конструктор

public Driver(string lastName, char category, double experience)

{

LastName = lastName;

Category = category;

DrivingExperience = experience;

}

// Метод для вывода информации

public void DisplayInfo()

{

Console.WriteLine($"Фамилия: {LastName}, Категория: {Category}, Стаж: {DrivingExperience} лет");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество водителей N: ");

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

List<Driver> drivers = new List<Driver>();

// Заполнение списка

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

{

Console.WriteLine($"\nВодитель {i + 1}:");

Console.Write("Фамилия: ");

string lastName = Console.ReadLine();

Console.Write("Категория (A/B/C/D): ");

char category = char.Parse(Console.ReadLine());

Console.Write("Стаж вождения (лет): ");

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

drivers.Add(new Driver(lastName, category, experience));

}

Console.Write("\nВведите стаж для поиска: ");

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

// Поиск водителей с заданным стажем

Console.WriteLine($"\nВодители со стажем {searchExperience} лет:");

foreach (var driver in drivers)

{

if (driver.DrivingExperience == searchExperience)

{

driver.DisplayInfo();

}

}

}

}

**Вариант 29**

**1 Опишите, как выполняется описание одномерного массива, обращение к элементам массива, его заполнение, вывод на экран. Приведите примеры.**

**Одномерный массив** в C# - это структура данных, хранящая элементы одного типа с линейной организацией доступа по индексу.

**Основные элементы работы с массивами:**

1. **Объявление массива**

тип\_данных[] имя\_массива; // Объявление

тип\_данных[] имя\_массива = new тип\_данных[размер]; // Объявление с созданием

1. **Инициализация**

int[] numbers = new int[5]; // Массив из 5 нулей

string[] names = { "Иван", "Петр" }; // Массив с начальными значениями

1. **Обращение к элементам**

numbers[0] = 10; // Запись

int x = numbers[1]; // Чтение

1. **Заполнение массива**

// С клавиатуры

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

{

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

numbers[i] = int.Parse(Console.ReadLine());

}

1. **Вывод массива**

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

{

Console.WriteLine($"Элемент {i}: {numbers[i]}");

}

**Пример работы с массивом:**

using System;

class Program

{

static void Main()

{

// Создание массива

double[] temps = new double[3];

// Заполнение

temps[0] = 23.5;

temps[1] = -10.2;

temps[2] = 5.8;

// Вывод

Console.WriteLine("Температуры:");

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

{

Console.WriteLine($"День {i+1}: {temps[i]}°C");

}

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить количество отрицательных элементов массива.**

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

int negativeCount = 0;

// Заполнение массива

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

{

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

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

if (array[i] < 0)

{

negativeCount++;

}

}

Console.WriteLine($"Количество отрицательных элементов: {negativeCount}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОЗЕРО (название – string, площадь – double, местонахождение – string). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: название. Отсортированный список вывести на экран.**

using System;

using System.Collections.Generic;

using System.Linq;

class Lake

{

public string Name { get; set; }

public double Area { get; set; }

public string Location { get; set; }

public Lake(string name, double area, string location)

{

Name = name;

Area = area;

Location = location;

}

public void DisplayInfo()

{

Console.WriteLine($"Название: {Name}, Площадь: {Area} км², Местоположение: {Location}");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество озер N: ");

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

List<Lake> lakes = new List<Lake>();

// Заполнение списка

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

{

Console.WriteLine($"\nОзеро {i + 1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Площадь (км²): ");

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

Console.Write("Местоположение: ");

string location = Console.ReadLine();

lakes.Add(new Lake(name, area, location));

}

// Сортировка по названию (Z-A)

var sortedLakes = lakes.OrderByDescending(lake => lake.Name).ToList();

Console.WriteLine("\nОтсортированный список озер (по убыванию названия):");

foreach (var lake in sortedLakes)

{

lake.DisplayInfo();

}

}

}

**Вариант 30**

**1 Опишите, как выполняется наследование классов, доступ к компонентам базового класса.**

Наследование - это механизм ООП, позволяющий создавать новый класс на основе существующего (базового) с возможностью расширения функциональности.

Основные принципы наследования:

1. Синтаксис наследования:

class БазовыйКласс { ... }

class ПроизводныйКласс : БазовыйКласс { ... }

1. Доступ к компонентам базового класса:

* public - доступен везде
* protected - доступен в производных классах
* private (по умолчанию) - доступен только в базовом классе

1. Ключевые особенности:

* Производный класс получает все поля и методы базового
* Можно добавлять новые члены
* Можно переопределять виртуальные методы (virtual/override)

**Пример наследования:**

using System;

class Animal // Базовый класс

{

protected string name;

public Animal(string name)

{

this.name = name;

}

public virtual void Speak()

{

Console.WriteLine("Животное издает звук");

}

}

class Dog : Animal // Производный класс

{

private string breed;

public Dog(string name, string breed) : base(name)

{

this.breed = breed;

}

public override void Speak()

{

Console.WriteLine($"{name} породы {breed} говорит: Гав!");

}

}

class Program

{

static void Main()

{

Dog myDog = new Dog("Шарик", "Дворняга");

myDog.Speak(); // Вызов переопределенного метода

}

}

**2 В одномерном массиве, состоящем из N целочисленных элементов, вычислить произведение элементов массива с чётными индексами**

using System;

class Program

{

static void Main()

{

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

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

int[] array = new int[N];

long product = 1; // Используем long для больших чисел

// Заполнение массива

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

{

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

array[i] = int.Parse(Console.ReadLine());

// Проверка на четность индекса

if (i % 2 == 0)

{

product \*= array[i];

}

}

Console.WriteLine($"Произведение элементов с четными индексами: {product}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: СОБАКА (кличка – string, порода – string, возраст – float). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: возраст. Отсортированный список вывести на экран.**

using System;

using System.Collections.Generic;

using System.Linq;

class Dog

{

public string Name { get; set; }

public string Breed { get; set; }

public float Age { get; set; }

public Dog(string name, string breed, float age)

{

Name = name;

Breed = breed;

Age = age;

}

public void DisplayInfo()

{

Console.WriteLine($"Кличка: {Name}, Порода: {Breed}, Возраст: {Age} лет");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество собак N: ");

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

List<Dog> dogs = new List<Dog>();

// Заполнение списка

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

{

Console.WriteLine($"\nСобака {i + 1}:");

Console.Write("Кличка: ");

string name = Console.ReadLine();

Console.Write("Порода: ");

string breed = Console.ReadLine();

Console.Write("Возраст (лет): ");

float age = float.Parse(Console.ReadLine());

dogs.Add(new Dog(name, breed, age));

}

// Сортировка по возрасту (по убыванию)

var sortedDogs = dogs.OrderByDescending(dog => dog.Age).ToList();

Console.WriteLine("\nОтсортированный список собак (по возрасту, старшие сначала):");

foreach (var dog in sortedDogs)

{

dog.DisplayInfo();

}

}

}

**Вариант 31**

**1 Опишите, как выполняется описание двумерного массива, обращение к элементам массива , его заполнение ,вывод на экран. Приведите примеры.**

**Двумерный массив** - это структура данных, представляющая собой таблицу элементов с доступом по двум индексам (строка и столбец).

**Основные операции с двумерными массивами:**

1. **Объявление массива**

тип\_данных[,] имя\_массива = new тип\_данных[строки, столбцы];

1. **Инициализация при объявлении**

int[,] matrix = { {1, 2, 3},

{4, 5, 6} };

1. **Обращение к элементам**

элемент = имя\_массива[строка, столбец]; // Чтение

имя\_массива[строка, столбец] = значение; // Запись

1. **Заполнение массива**

Random rnd = new Random();

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

matrix[i,j] = rnd.Next(1, 10);

}

}

1. **Вывод массива**

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

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

}

Console.WriteLine();

}

**Пример работы с двумерным массивом:**

using System;

class Program

{

static void Main()

{

// Создание массива 2x3

double[,] temps = new double[2, 3];

// Заполнение

temps[0,0] = 15.5; temps[0,1] = 16.2; temps[0,2] = 14.8;

temps[1,0] = 18.3; temps[1,1] = 17.9; temps[1,2] = 16.5;

// Вывод

Console.WriteLine("Температуры по дням:");

for (int i = 0; i < temps.GetLength(0); i++)

{

for (int j = 0; j < temps.GetLength(1); j++)

{

Console.Write($"{temps[i,j]}°C\t");

}

Console.WriteLine();

}

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить сумму положительных элементов массива.**

using System;

class Program

{

static void Main()

{

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

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

double[] array = new double[N];

double sum = 0;

// Заполнение массива

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

{

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

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

if (array[i] > 0)

{

sum += array[i];

}

}

Console.WriteLine($"Сумма положительных элементов: {sum:F2}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ДОМ (адрес – string, количество комнат – int, площадь – double). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: площадь домов. Отсортированный список вывести на экран.**

using System;

using System.Collections.Generic;

using System.Linq;

class House

{

public string Address { get; set; }

public int Rooms { get; set; }

public double Area { get; set; }

public House(string address, int rooms, double area)

{

Address = address;

Rooms = rooms;

Area = area;

}

public void DisplayInfo()

{

Console.WriteLine($"Адрес: {Address}, Комнат: {Rooms}, Площадь: {Area} м²");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество домов N: ");

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

List<House> houses = new List<House>();

// Заполнение списка

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

{

Console.WriteLine($"\nДом {i + 1}:");

Console.Write("Адрес: ");

string address = Console.ReadLine();

Console.Write("Количество комнат: ");

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

Console.Write("Площадь (м²): ");

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

houses.Add(new House(address, rooms, area));

}

// Сортировка по площади (по убыванию)

var sortedHouses = houses.OrderByDescending(house => house.Area).ToList();

Console.WriteLine("\nОтсортированный список домов:");

foreach (var house in sortedHouses)

{

house.DisplayInfo();

}

}

}

**Вариант 32**

**1 Опишите основные элементы класса Stack. Приведите примеры.**

Класс Stack<T> в C# представляет коллекцию, работающую по принципу LIFO (Last In, First Out - последний вошел, первый вышел).

Основные элементы:

1. Методы:
   * Push(T item) - добавляет элемент на вершину стека
   * Pop() - удаляет и возвращает элемент с вершины стека
   * Peek() - возвращает элемент с вершины без удаления
   * Clear() - очищает стек
   * Contains(T item) - проверяет наличие элемента
2. Свойства:
   * Count - количество элементов в стеке

**Пример использования:**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

Stack<string> stack = new Stack<string>();

// Добавление элементов

stack.Push("Первый");

stack.Push("Второй");

stack.Push("Третий");

Console.WriteLine($"Элементов в стеке: {stack.Count}");

Console.WriteLine($"Верхний элемент: {stack.Peek()}");

// Извлечение элементов

Console.WriteLine("\nИзвлечение элементов:");

while (stack.Count > 0)

{

Console.WriteLine(stack.Pop());

}

}

}

**2 Данные о температуре воздуха за ноябрь хранятся в вещественном массиве. Определить, сколько раз температура опускалась ниже – N градусов.**

using System;

class Program

{

static void Main()

{

double[] novemberTemps = { -5.2, -3.1, -7.8, -2.5, -10.3, -1.8, -8.6 };

Console.Write("Введите пороговую температуру N: ");

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

int count = 0;

foreach (double temp in novemberTemps)

{

if (temp < -N) count++;

}

Console.WriteLine($"Температура опускалась ниже -{N}°C {count} раз(а)");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОБУВЬ (производитель – string, размер – double, цена – double). Условие: вывести на экран обувь имеющую заданного размера (размер обуви ввести с клавиатуры).**

using System;

using System.Collections.Generic;

class Shoe

{

public string Manufacturer { get; set; }

public double Size { get; set; }

public double Price { get; set; }

public Shoe(string manufacturer, double size, double price)

{

Manufacturer = manufacturer;

Size = size;

Price = price;

}

public void DisplayInfo()

{

Console.WriteLine($"Производитель: {Manufacturer}, Размер: {Size}, Цена: {Price} руб.");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество пар обуви: ");

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

List<Shoe> shoes = new List<Shoe>();

// Ввод данных

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

{

Console.WriteLine($"\nПара {i+1}:");

Console.Write("Производитель: ");

string manuf = Console.ReadLine();

Console.Write("Размер: ");

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

Console.Write("Цена: ");

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

shoes.Add(new Shoe(manuf, size, price));

}

// Поиск по размеру

Console.Write("\nВведите размер для поиска: ");

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

Console.WriteLine("\nНайденная обувь:");

bool found = false;

foreach (Shoe shoe in shoes)

{

if (shoe.Size == searchSize)

{

shoe.DisplayInfo();

found = true;

}

}

if (!found) Console.WriteLine("Обувь такого размера не найдена");

}

}

**Вариант 33**

**1 Опишите, как выполняется описание одномерного массива, обращение к элементам массива, его заполнение, вывод на экран. Приведите примеры.**

**Одномерный массив** - это структура данных, содержащая элементы одного типа с линейной организацией.

**Основные операции:**

1. **Объявление массива:**

тип[] имяМассива; // Объявление

тип[] имяМассива = new тип[размер]; // Создание

1. **Инициализация:**

double[] temps = new double[5]; // Массив из 5 элементов

string[] cities = {"Москва", "СПб"}; // С начальными значениями

1. **Обращение к элементам:**

temps[0] = 23.5; // Запись

double x = temps[1]; // Чтение

1. **Заполнение массива:**

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

{

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

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

}

1. **Вывод массива:**

Console.WriteLine("Температуры:");

foreach (double t in temps)

{

Console.Write(t + " ");

}

**Полный пример:**

using System;

class Program

{

static void Main()

{

double[] temps = new double[3];

// Заполнение

temps[0] = 15.5;

temps[1] = -3.2;

temps[2] = 7.8;

// Вывод

Console.WriteLine("Температуры:");

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

{

Console.WriteLine($"День {i+1}: {temps[i]}°C");

}

}

}

**2 В одномерном массиве, состоящем из N вещественных элементов, вычислить количество отрицательных элементов массива.**

using System;

class Program

{

static void Main()

{

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

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

double[] arr = new double[N];

int negativeCount = 0;

// Заполнение и подсчет

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

{

Console.Write($"Элемент {i+1}: ");

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

if (arr[i] < 0) negativeCount++;

}

Console.WriteLine($"Отрицательных элементов: {negativeCount}");

}

}

**3 Определить пользовательский класс. Определить в классе методы для просмотра и установки полей данных. Создать список, состоящий из N элементов, каждый элемент которого является объектом заданного класса. Заполнить список значениями, введенными с клавиатуры. Выполнить обработку объектов удовлетворяющих заданному условию. Имя класса: ОЗЕРО (название – string, площадь – double, местонахождение – string). Условие: отсортировать список по убыванию. Поле для выполнения сравнения объектов списка: название. Отсортированный список вывести на экран.**

using System;

using System.Collections.Generic;

using System.Linq;

class Lake

{

public string Name { get; set; }

public double Area { get; set; }

public string Location { get; set; }

public Lake(string name, double area, string location)

{

Name = name;

Area = area;

Location = location;

}

public void DisplayInfo()

{

Console.WriteLine($"{Name} ({Area} км²) - {Location}");

}

}

class Program

{

static void Main()

{

Console.Write("Введите количество озер: ");

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

List<Lake> lakes = new List<Lake>();

// Ввод данных

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

{

Console.WriteLine($"\nОзеро {i+1}:");

Console.Write("Название: ");

string name = Console.ReadLine();

Console.Write("Площадь (км²): ");

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

Console.Write("Местоположение: ");

string loc = Console.ReadLine();

lakes.Add(new Lake(name, area, loc));

}

// Сортировка по убыванию названия

var sortedLakes = lakes.OrderByDescending(l => l.Name).ToList();

// Вывод результатов

Console.WriteLine("\nОзера (отсортировано по названию Z-A):");

foreach (Lake lake in sortedLakes)

{

lake.DisplayInfo();

}

}

}