**Постфиксы**

float b = 30.6f; decimal c = 1005.8M; uint a = 10U; long b = 20L; ulong c = 30UL;

**Конвертация**

short \_short = Convert.ToInt16(Console.ReadLine());

**Nullable**

bool? \_null1 = null;

Nullable<bool> \_null2 = null;

**Упаковка\распаковка**

int box = 5;

object obj = box; // упаковка

int Unbox = (int)obj; // распаковка

**Операции со строками**

text = text.Replace("хороший", "плохой");

// индекс последнего символа

int ind = text.Length - 1;

// вырезаем последний символ

text = text.Remove(ind);

// вставка  
text = text.Insert(8, subString);

// Разделение  
string[] words = text.Split(new char[] { ' ' });

// Поиск  
string s1 = "hello world";

char ch = 'o';

int indexOfChar = s1.IndexOf(ch); // равно 4

// сравнение  
int result = String.Compare(s1, s2);  
//соединение  
string s4 = String.Concat(s3, "!!!"); // результат: строка "hello world!!!"

**Перегрузка операторов**

public static bool operator >(Variant23 first, Variant23 second )

{

return first.length > second.length;

}

**Переопределение методов**

public override int GetHashCode()

{return pname.GetHashCode();}

public override bool Equals(object obj)

{if (obj.GetType() != this.GetType()) return false;

produser person = (produser)obj;

return (this.pname == person.pname);}

public override string ToString()

{return "Person: " + Name + " " + Age; }

**Исключения**

if (name== null || name.Length < 2)

    {throw new Exception("Длина имени меньше 2 символов");}

catch (Exception e)

{Console.WriteLine($"Ошибка: {e.Message}");}

**Делегаты**

delegate void AccountHandler(string message);

event AccountHandler Notify;

В классе в методе - if(Notify !=null) Notify("Произошло"); или Notify?.Invoke("Произошло");

Дальше в мэин - Account account = new Account(100);

account.Notify += DisplayMessage;   // Добавляем обработчик для события Notify

void DisplayRedMessage(string message)

{

    // Устанавливаем красный цвет символов

    Console.ForegroundColor = ConsoleColor.Red;

    Console.WriteLine(message);

    // Сбрасываем настройки цвета

    Console.ResetColor();

}

**Коллекции**

Console.WriteLine("Stack<T>");

Stack<double> stack = new Stack<double>();

stack.Push(0.54);

stack.Push(6.345);

stack.Push(8.76543);

Console.WriteLine("LinkedList<T>");

LinkedList<double> spisok = new LinkedList<double>();

spisok.AddFirst(2.53);

spisok.AddLast(2.34);

Console.WriteLine("Stack<Student>");

Stack<Student> sstack = new Stack<Student>();

sstack.Push(new Student("Влад"));

sstack.Push(new Student("Костя"));

sstack.Push(new Student("Никита"));

**LINQ**

Select: определяет проекцию выбранных значений

Where: определяет фильтр выборки

OrderBy: упорядочивает элементы по возрастанию

OrderByDescending: упорядочивает элементы по убыванию

ThenBy: задает дополнительные критерии для упорядочивания элементов возрастанию

ThenByDescending: задает дополнительные критерии для упорядочивания элементов по убыванию

Join: соединяет две коллекции по определенному признаку

GroupBy: группирует элементы по ключу

ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь

GroupJoin: выполняет одновременно соединение коллекций и группировку элементов по ключу

Reverse: располагает элементы в обратном порядке

All: определяет, все ли элементы коллекции удовлятворяют определенному условию

Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию

Contains: определяет, содержит ли коллекция определенный элемент

Distinct: удаляет дублирующиеся элементы из коллекции

Except: возвращает разность двух коллекцию, то есть те элементы, которые создаются только в одной коллекции

Union: объединяет две однородные коллекции

Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях

Count: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию

Sum: подсчитывает сумму числовых значений в коллекции

Average: подсчитывает cреднее значение числовых значений в коллекции

Min: находит минимальное значение

Max: находит максимальное значение

Take: выбирает определенное количество элементов

Skip: пропускает определенное количество элементов

TakeWhile: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно

SkipWhile: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы

Concat: объединяет две коллекции

Zip: объединяет две коллекции в соответствии с определенным условием

First: выбирает первый элемент коллекции

FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

Single: выбирает единственный элемент коллекции, если коллекция содердит больше или меньше одного элемента, то генерируется исключение

SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

ElementAt: выбирает элемент последовательности по определенному индексу

ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона

Last: выбирает последний элемент коллекции

LastOrDefault: выбирает последний элемент коллекции или возвращает значение по умолчанию

+ в лабе 11 примеры

**Работа с дисками**

MVSLog.MVSWriter(streamWriter, "Создание MVSInspect");//пишем в файл MVSlogfile.txt о создании

//string pathtofile = @"D:\Что-то\Где-то\Laba-13";

Directory.CreateDirectory("MVSInspect");//создаем необходимый каталог

FileStream fs = File.Create("MVSInspect\\MVSdirinfo.txt");//создаем файл

fs.Close();

MVSLog.MVSWriter(streamWriter, "Создание MVSdirinfo.txt");//пишем в файл создание MVSdirinfo.txt

StreamWriter sw = new StreamWriter("MVSInspect\\MVSdirinfo.txt");// в каталог в файл MVSdirinfo.txt открываем поток

DirectoryInfo dir = new DirectoryInfo(str);//получаем информацию о директории

if (dir.Exists)

{

DirectoryInfo[] d = dir.GetDirectories();//если существует получаем каталоги и файлы

FileInfo[] f = dir.GetFiles();

for (int i = 0; i < d.Length; i++)//выводим все каталоги и файлы

{

Console.WriteLine(d[i].Name);

sw.WriteLine(d[i].Name);

}

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

{

Console.WriteLine(f[i].Name);

sw.WriteLine(f[i].Name);

}

MVSLog.MVSWriter(streamWriter, "Скопировано из MVSdirinfo в MVSdirinfocopy");//пишем в файл "копирование каталога"

if (File.Exists("MVSInspect\\MVSdirinfocopy.txt"))//перемещаем файл создавая его копию, после удаляем

{

File.Delete("MVSInspect\\MVSdirinfocopy.txt");

}

FileInfo q = new FileInfo("MVSInspect\\MVSdirinfo.txt");

q.CopyTo("MVSInspect\\MVSdirinfocopy.txt");

Directory.CreateDirectory("MVSFiles");

q.CopyTo("MVSFiles\\MVSmove.txt");

File.Delete("MVSInspect\\MVSdirinfo.txt");

public static StreamWriter CreateStream(string str)//Создает экземпляр класса записи в файл

{

StreamWriter streamWriter = new StreamWriter(str);

return streamWriter;

}

public static StreamReader CreateStreamR(string str)//Создает экземпляр класса чтение из файла

{

StreamReader streamReader = new StreamReader(str);

return streamReader;

}

public static void MVSWriter(StreamWriter streamWriter, string info)//Запись в файл время создания записи и запись

{

streamWriter.WriteLine(DateTime.Now.Hour + ":" + DateTime.Now.Minute + ":"

+ DateTime.Now.Day + ":" + DateTime.Now.Month + ":" + DateTime.Now.Year + " ");

streamWriter.WriteLine(info);

}

public static void CloseStream(StreamWriter streamWriter)//Закрываем файл после работы с ним

{

streamWriter.Close();

}

public static string MVSReader(StreamReader streamReader)

{

return streamReader.ReadToEnd();

}

public static void MVSSearcher(StreamReader streamReader, string info)//Поиск в файле конкретную строку

{

string text = MVSReader(streamReader);

if (text.Contains(info))

{

Console.WriteLine("Файл содержит необходимую информацию");

}

else

{

Console.WriteLine("Файл не содержит необходимую информацию");

}

}

}

**Потоки**

public static void Even()

{

int n = 100;

for (int i = 2; i < n; i += 2)

{

Thread.Sleep(30);

Console.WriteLine(i);

}

}

public static void Noteven()

{

int n = 100;

for (int i = 1; i < n; i += 2)

{

Thread.Sleep(60);

Console.WriteLine(i);

}

}

int n = 100;

Thread th = new Thread(new ThreadStart(Operation));

Console.WriteLine(th.ThreadState);

Console.WriteLine(th.Priority);

th.Name = "SecondThread";

Console.WriteLine(th.Name);

Console.WriteLine("id: " + th.ManagedThreadId);

**TPL**

static async void Async()

{

WriteLine("Начало асинхронного метода");

await Task.Run(() => Meth());

WriteLine("Конец асинхронного метода");

}

WriteLine($"Генарация двух массивов в for: {st.ElapsedMilliseconds}мс");

int[] arr3 = new int[1000000];

int[] arr4 = new int[1000000];

st.Restart();

Parallel.For(0, arr3.Length, i => { arr3[i] = random.Next(0, 300); arr4[i] = random.Next(0, 300); });

st.Stop(); // за сколько создался второй массив

WriteLine($"Генарация двух массивов в ParallelFor: {st.ElapsedMilliseconds}мс");

int[] arr5 = new int[1000000];

int[] arr6 = new int[1000000];

st.Restart();

Parallel.ForEach<int>(arr5, i => { arr5[i] = random.Next(0, 300); });

st.Stop(); // за сколько создался третий массив

long t1 = st.ElapsedMilliseconds;

st.Restart();

Parallel.ForEach<int>(arr6, i => { arr6[i] = random.Next(0, 300); });

st.Stop();

WriteLine($"Генерация двух массивов в ParallelForEach: {st.ElapsedMilliseconds + t1}мс");

WriteLine();

var task11 = Task.Factory.StartNew<int>(() => Task11(15)); // создание 1 задачи

var task22 = Task.Factory.StartNew<int>(() => Task22(15));// создание 2 задачи

task1 = new Task(() => Task4(task11.Result, task22.Result)); // создание 3 задачи, что ринимает задачи 1 и 2

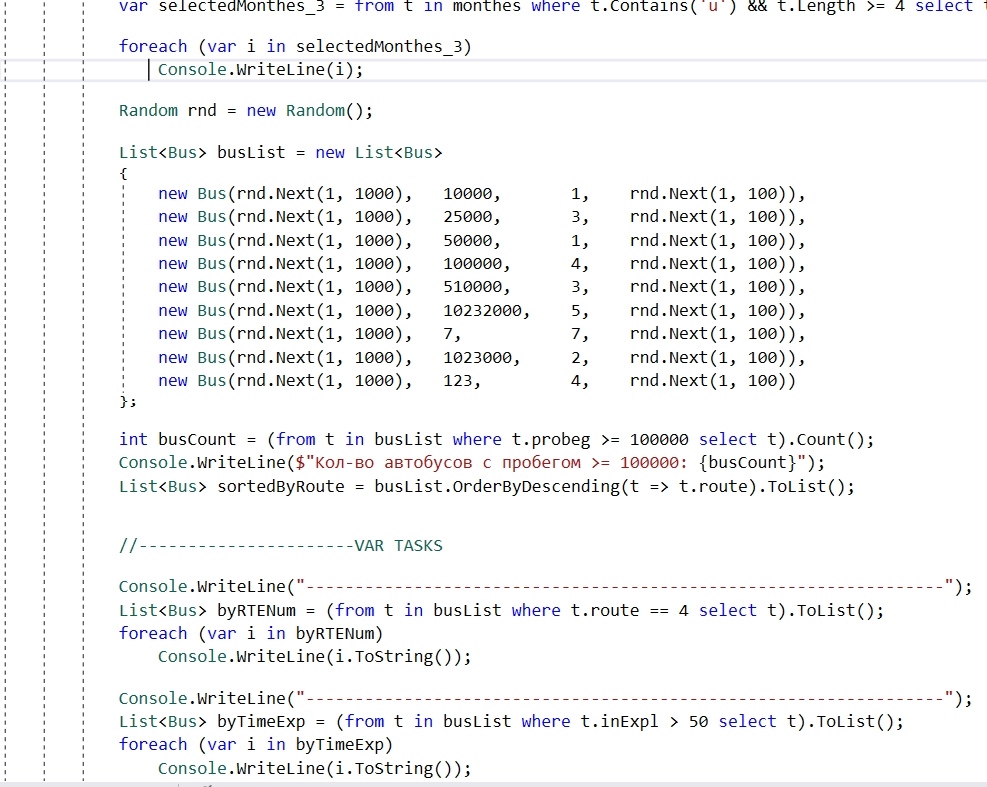
task1.Start();

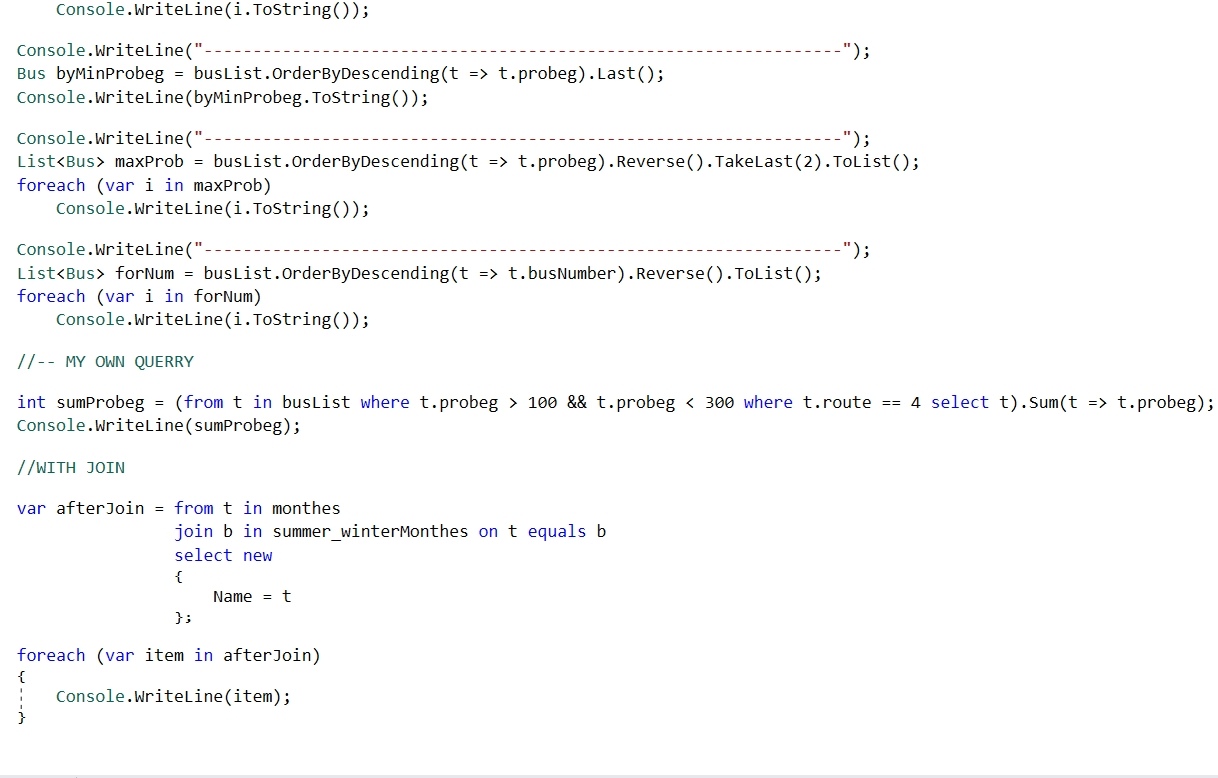
task1.Wait();

task11.Dispose();

task22.Dispose();

WriteLine();





**Запись в файл**

static void Main(string[] args)

        {

            string writePath = @"C:\SomeDir\hta.txt";

            string text = "Привет мир!\nПока мир...";

            try

            {

                using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))

                {

                    sw.WriteLine(text);

                }

                using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default))

                {

                    sw.WriteLine("Дозапись");

                    sw.Write(4.5);

                }

                Console.WriteLine("Запись выполнена");

            }

**ArrayFunc**

public delegate string MyDelegate();

**class ArrayFunc<t>**

{

public static string ToString()

{

return "Work";

}

new public static string GetHashCode()

{

return "Work";

}

public static Func<int, int>[] funcs = new Func<int, int>[3];

**class ArrayIndex<T>**

{

ArrayFunc<double>[] data;

public ArrayIndex()

{

data = new ArrayFunc<double>[5];

}

public ArrayFunc<double> this[int index]

{

get

{

return data[index];

}

set

{

data[index] = value;

}

}

**abstract class Function**

{

public double X { get; set; }

public virtual double Func(double x, double c, double a, double b)

{

return a \* x \* x + b \* x + c;

}

public virtual double Func(double x, double a, double b)

{

return a \* x + b;

}

}

}

**class Liner : Function**

{

public double A { get; set; }

public double B { get; set; }

}

}

**static void Main(string[] args)**

{

MyDelegate[] delegates = new MyDelegate[2];

delegates[0] = ArrayFunc<string>.ToString;

delegates[1] = ArrayFunc<string>.GetHashCode;

ArrayFunc<string>.funcs[0] = i => i + 1;

ArrayFunc<string>.funcs[1] = i => i \* 2;

ArrayFunc<string>.funcs[2] = i => i - 3;

foreach (Func<int, int> func in ArrayFunc<string>.funcs)

{

Console.WriteLine(func(2));

}

**using System;**

**using System.Collections.Generic;**

**using System.IO;**

**using System.Linq;**

**using System.Runtime.Serialization.Formatters.Binary;**

**using System.Xml;**

**using System.Xml.Linq;**

**using System.Runtime.Serialization.Formatters.Soap;**

**using System.Xml.Serialization;**

**using System.Text.Json;**

**using System.Threading.Tasks;**

**using System.Text;**