МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

ОТЧЕТ ПО учебной

вычислительной (ознакомительной)

практике

(Практическая часть)

Новикова Егора Андреевича

студента 1 курса 8 группы,

специальность «Прикладная математика»

Руководитель практики:

старший преподаватель

О.О. Колб

[Практическая часть курса 3](#_Toc108007609)

[1 Выбранный курс 3](#_Toc108007610)

[2 Решение задач: .Net C#. 4](#_Toc108007611)

3 Итоги…………………………………………………………………155

1. **Выбранный курс**

В ходе выполнения практической части учебной практики был пройден курс “.Net Development (2021/Q4/1)” на платформе AutoCode (<https://autocode-next.lab.epam.com/courses/6>).

Курс содержит 33 задания на такие основы .Net как:

- работа с различными типами данных

- работа с вводом-выводом

- ветвление (if-else statements)

- работа с исключениями

- работа с различными конструкциями циклов (for, while, do-while)

- работа с массивами

- работа со строками

- форматирование и приведение строк

**2. Решение задач .Net C#**

Задача №1

Название: Fixing Style Issues in "Hello, world!" app

Условие: Отформатировать и исправить ошибки в исходном файле

Код:

﻿namespace HelloWorldStyle

{

public static

class HelloWorld

{

public static string ReturnGreeting(string hello, string addressee)

{

return hello + ", " + addressee + "!";

}

public static int ReturnSum(int a, int b)

{

return

a

+ b;

}

public static int[] ReturnArray(int length)

{

var array = new

int[

length];

for

(

int i = 0; i < length; i++)

{

array[i] = i;

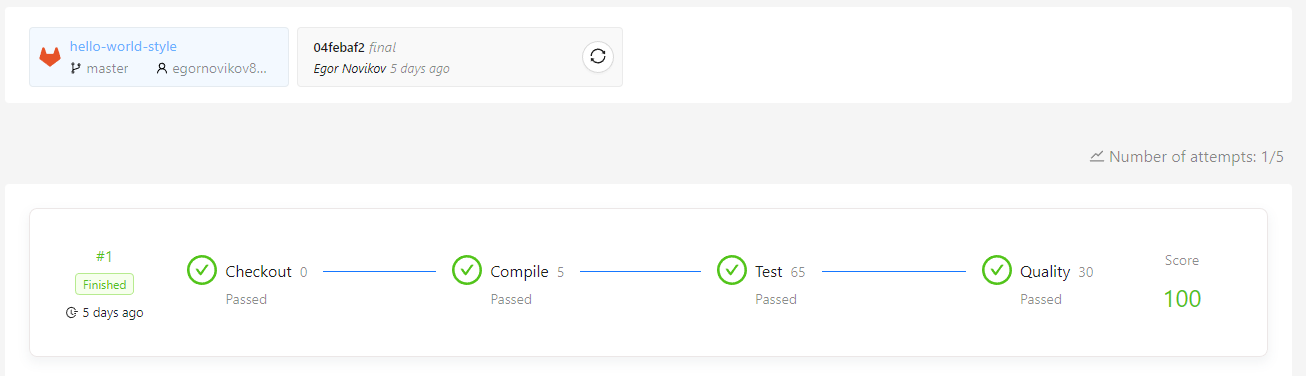
}

return array;

}

}

}

Результаты теста:

Задача №2

Название: Fancy Calculator

Условие: Исправить методы в исходном документе, чтоб они работали корректно.

Код:

﻿ ﻿using System;

namespace FancyCalc

{

public static class Operations

{

public static int Plus(int x, int y)

{

return x + y;

}

public static int Minus(int x, int y)

{

return x - y;

}

public static int Multiply(int x, int y)

{

int result = x \* y;

return result;

}

public static int Sum(int x1, int x2, int x3)

{

int sum1 = x1 + x2;

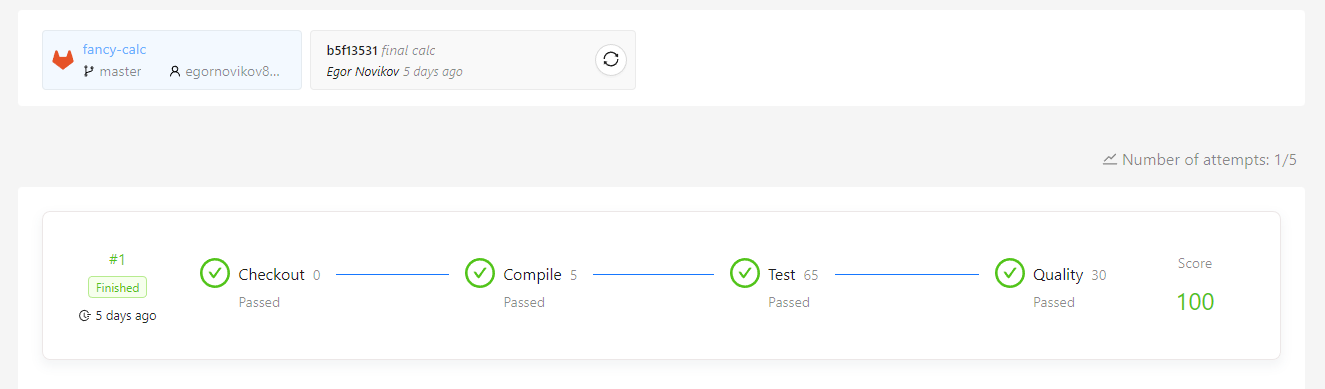
int sum2 = sum1 + x3;

return sum2;

}

}

}

Результаты теста:

Задача №3

Название: Print face

Условие: Используя вывод в консоль, нарисовать лицо

Код:

﻿ ﻿using System;

namespace PrintFace

{

public static class Program

{

/// <summary>

/// Print "Hello, world!" in a separate line.

/// </summary>

public static void Main()

{

Console.WriteLine(" Hello, world! ");

}

/// <summary>

/// Print "Hello, userName!" in a separate line.

/// </summary>

/// <param name="userName">Name of user.</param>

public static void SayHelloUser(string userName)

{

Console.WriteLine($"Hello, {userName}!");

}

/// <summary>

/// Print this face.

/// +"""""+

/// (| o o |)

/// | ^ |

/// | '-' |

/// +-----+.

/// </summary>

public static void PrintFace()

{

Console.WriteLine($" +\"\"\"\"\"+");

Console.WriteLine($"(| o o |)");

Console.WriteLine($" | ^ |");

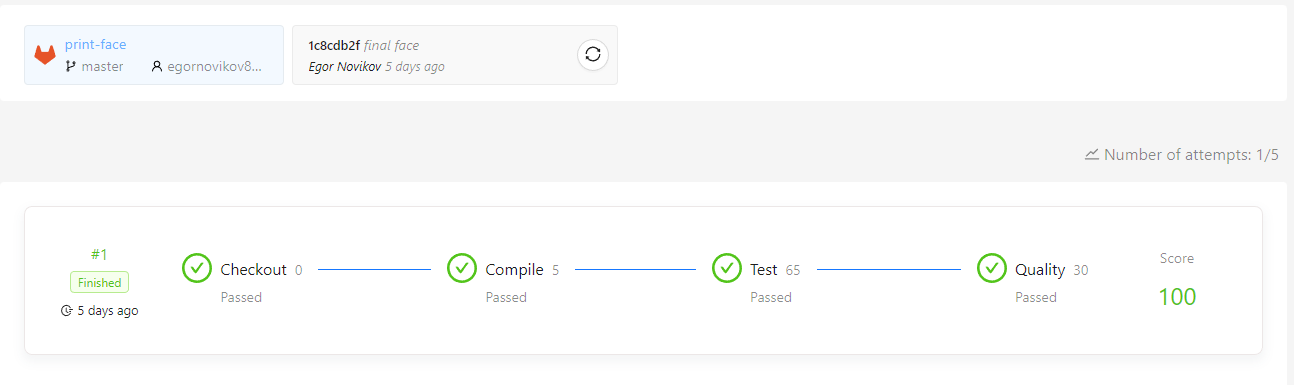
Console.WriteLine($" | '-' |");

Console.WriteLine($" +-----+");

}

}

}

Результаты теста:

Задача №4

Название: Creating Static Methods

Условие: Научиться создавать методы с различными параметрами (out, ref и обычные)

Код:

﻿MethodsWithOutParameters.cs:

﻿namespace CreatingMethods

{

public static class MethodsWithOutParameters

{

public static void ReturnValues(out bool trueValue, out bool falseValue)

{

trueValue = true;

falseValue = false;

}

public static void ReturnValues(out char lowerCaseA, out char upperCaseA)

{

lowerCaseA = 'a';

upperCaseA = 'A';

}

public static void ReturnValues(out float minFloatValue, out float maxFloatValue)

{

minFloatValue = -3.40282347E+38f;

maxFloatValue = 3.40282347E+38f;

}

public static void ReturnValues(out int minIntValue, out int maxIntValue)

{

minIntValue = -2147483648;

maxIntValue = 2147483647;

}

public static void ReturnValues(out long minLongValue, out long maxLongValue)

{

minLongValue = -9223372036854775808;

maxLongValue = 9223372036854775807;

}

}

}

MethodsWithRefParameters.cs:

﻿namespace CreatingMethods

{

public static class MethodsWithRefParameters

{

public static bool ReturnParameterValueAndSetParameterToDefaultValue(ref bool boolValue)

{

bool value = boolValue;

boolValue = default(bool);

return value;

}

public static char ReturnParameterValueAndSetParameterToDefaultValue(ref char charValue)

{

char value = charValue;

charValue = default(char);

return value;

}

public static float ReturnParameterValueAndSetParameterToDefaultValue(ref float floatValue)

{

float value = floatValue;

floatValue = default(float);

return value;

}

public static int ReturnParameterValueAndSetParameterToDefaultValue(ref int intValue)

{

int value = intValue;

intValue = default(int);

return value;

}

public static long ReturnParameterValueAndSetParameterToDefaultValue(ref long longValue)

{

long value = longValue;

longValue = default(long);

return value;

}

}

}

PassThroughtMethods:

namespace CreatingMethods

{

// ReSharper disable once IdentifierTypo

public static class PassthroughMethods

{

public static bool ReturnValue(bool boolValue)

{

return boolValue;

}

public static byte ReturnValue(byte byteValue)

{

return byteValue;

}

public static sbyte ReturnValue(sbyte sbyteValue)

{

return sbyteValue;

}

public static char ReturnValue(char charValue)

{

return charValue;

}

public static decimal ReturnValue(decimal decimalValue)

{

return decimalValue;

}

public static double ReturnValue(double doubleValue)

{

return doubleValue;

}

public static float ReturnValue(float floatValue)

{

return floatValue;

}

public static int ReturnValue(int intValue)

{

return intValue;

}

public static uint ReturnValue(uint uintValue)

{

return uintValue;

}

public static long ReturnValue(long longValue)

{

return longValue;

}

public static ulong ReturnValue(ulong ulongValue)

{

return ulongValue;

}

public static short ReturnValue(short shortValue)

{

return shortValue;

}

public static ushort ReturnValue(ushort ushortValue)

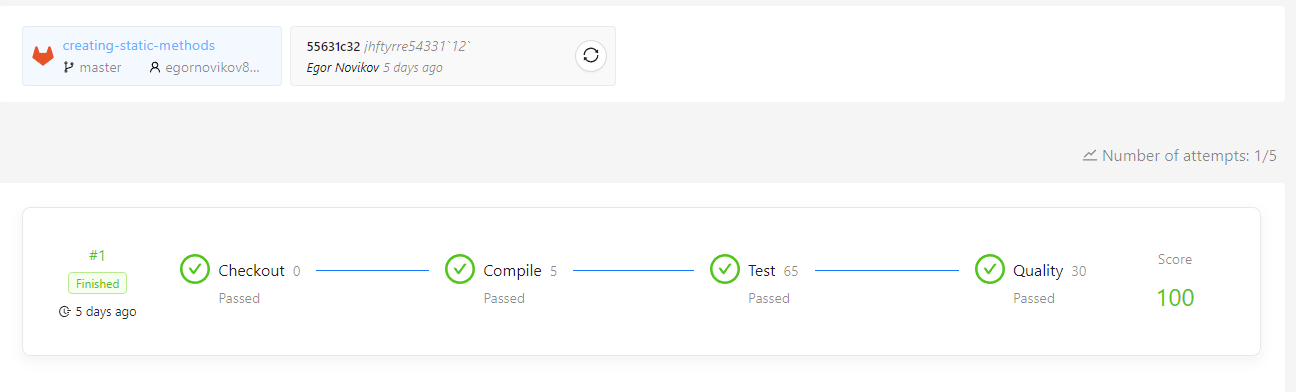
{

return ushortValue;

}

}

}

Результаты теста:

Задача №5

Название: Numeric Conversions

Условие: Научиться приведения различных типов данных

Код:

ExplicitConversations.cs:

﻿namespace NumericConversions

{

public static class ExplicitConversion

{

public static int LongToInt(long longValue)

{

return (int)longValue;

}

public static int FloatToInt(float floatValue)

{

return (int)floatValue;

}

public static int DoubleToInt(double doubleValue)

{

return (int)doubleValue;

}

public static int DecimalToInt(decimal decimalValue)

{

return (int)decimalValue;

}

public static long FloatToLong(float floatValue)

{

return (long)floatValue;

}

public static long DoubleToLong(double doubleValue)

{

return (long)doubleValue;

}

public static long DecimalToLong(decimal decimalValue)

{

return (long)decimalValue;

}

public static byte ShortToByte(short shortValue)

{

return (byte)shortValue;

}

public static byte IntToByte(int intValue)

{

return (byte)intValue;

}

public static short IntToShort(int intValue)

{

return (short)intValue;

}

}

}

ImplicitConversations.cs:

﻿namespace NumericConversions

{

public static class ImplicitConversion

{

public static long IntToLong(int intValue)

{

return intValue;

}

public static float IntToFloat(int intValue)

{

return intValue;

}

public static double IntToDouble(int intValue)

{

return intValue;

}

public static decimal IntToDecimal(int intValue)

{

return intValue;

}

public static float LongToFloat(long longValue)

{

return longValue;

}

public static double LongToDouble(long longValue)

{

return longValue;

}

public static decimal LongToDecimal(long longValue)

{

return longValue;

}

public static short ByteToShort(byte byteValue)

{

return byteValue;

}

public static int ByteToInt(byte byteValue)

{

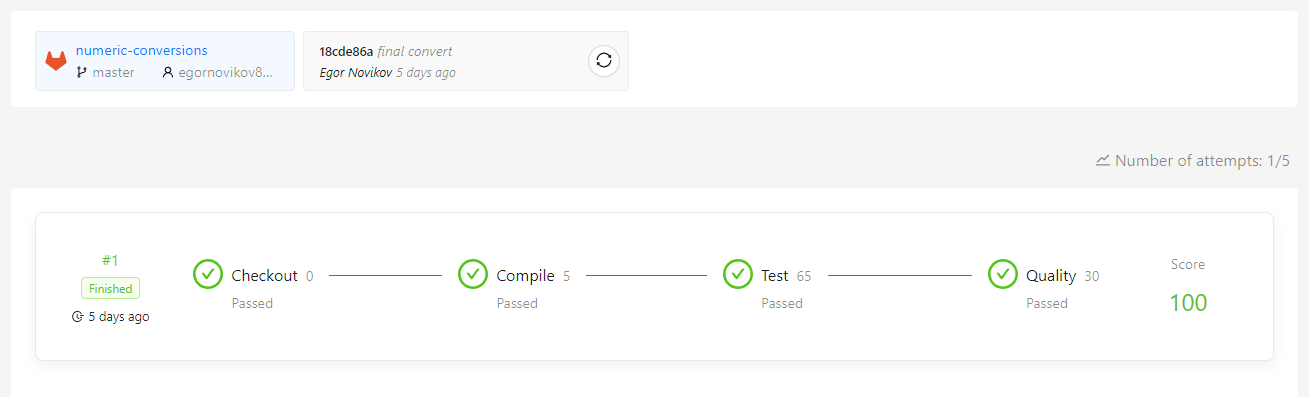
return byteValue;

}

}

}

Результаты теста:



Задача №6

Название: Morse Code Alphabet

Условие: Написать программу, которая переводит латинский алфавит в азбуку Морзе

Код:

UsingIf.cs:

﻿#pragma warning disable S3776

namespace MorseCodeAlphabet

{

public static class UsingIf

{

public static string GetMorseCode(char c)

{

if (c == 'A' || c == 'a')

{

return ".-";

}

if (c == 'B' || c == 'b')

{

return "-...";

}

if (c == 'C' || c == 'c')

{

return "-.-.";

}

if (c == 'D' || c == 'd')

{

return "-..";

}

if (c == 'E' || c == 'e')

{

return ".";

}

if (c == 'F' || c == 'f')

{

return "..-.";

}

if (c == 'G' || c == 'g')

{

return "--.";

}

if (c == 'H' || c == 'h')

{

return "....";

}

if (c == 'I' || c == 'i')

{

return "..";

}

if (c == 'J' || c == 'j')

{

return ".---";

}

if (c == 'K' || c == 'k')

{

return "-.-";

}

if (c == 'L' || c == 'l')

{

return ".-..";

}

if (c == 'M' || c == 'm')

{

return "--";

}

if (c == 'N' || c == 'n')

{

return "-.";

}

if (c == 'O' || c == 'o')

{

return "---";

}

if (c == 'P' || c == 'p')

{

return ".--.";

}

if (c == 'Q' || c == 'q')

{

return "--.-";

}

if (c == 'R' || c == 'r')

{

return ".-.";

}

if (c == 'S' || c == 's')

{

return "...";

}

if (c == 'T' || c == 't')

{

return "-";

}

if (c == 'U' || c == 'u')

{

return "..-";

}

if (c == 'V' || c == 'v')

{

return "...-";

}

if (c == 'W' || c == 'w')

{

return ".--";

}

if (c == 'X' || c == 'x')

{

return "-..-";

}

if (c == 'Y' || c == 'y')

{

return "-.--";

}

if (c == 'Z' || c == 'z')

{

return "--..";

}

return string.Empty;

}

}

}

UsingSwitch.cs:

﻿namespace MorseCodeAlphabet

{

public static class UsingSwitch

{

public static byte GetMorseCode(char c)

{

switch (c)

{

case 'A':

case 'a':

return 0b0011\_0001;

case 'B':

case 'b':

return 0b1111\_1110;

case 'C':

case 'c':

return 0b1111\_1010;

case 'D':

case 'd':

return 0b0111\_0110;

case 'E':

case 'e':

return 0b0001\_0001;

case 'F':

case 'f':

return 0b1111\_1011;

case 'G':

case 'g':

return 0b0111\_0100;

case 'H':

case 'h':

return 0b1111\_1111;

case 'I':

case 'i':

return 0b0011\_0011;

case 'J':

case 'j':

return 0b1111\_0001;

case 'K':

case 'k':

return 0b0111\_0010;

case 'L':

case 'l':

return 0b1111\_1101;

case 'M':

case 'm':

return 0b0011\_0000;

case 'N':

case 'n':

return 0b0011\_0010;

case 'O':

case 'o':

return 0b0111\_0000;

case 'P':

case 'p':

return 0b1111\_1001;

case 'Q':

case 'q':

return 0b1111\_0100;

case 'R':

case 'r':

return 0b0111\_0101;

case 'S':

case 's':

return 0b0111\_0111;

case 'T':

case 't':

return 0b0001\_0000;

case 'U':

case 'u':

return 0b0111\_0011;

case 'V':

case 'v':

return 0b1111\_0111;

case 'W':

case 'w':

return 0b0111\_0001;

case 'X':

case 'x':

return 0b1111\_0110;

case 'Y':

case 'y':

return 0b1111\_0010;

case 'Z':

case 'z':

return 0b1111\_1100;

default:

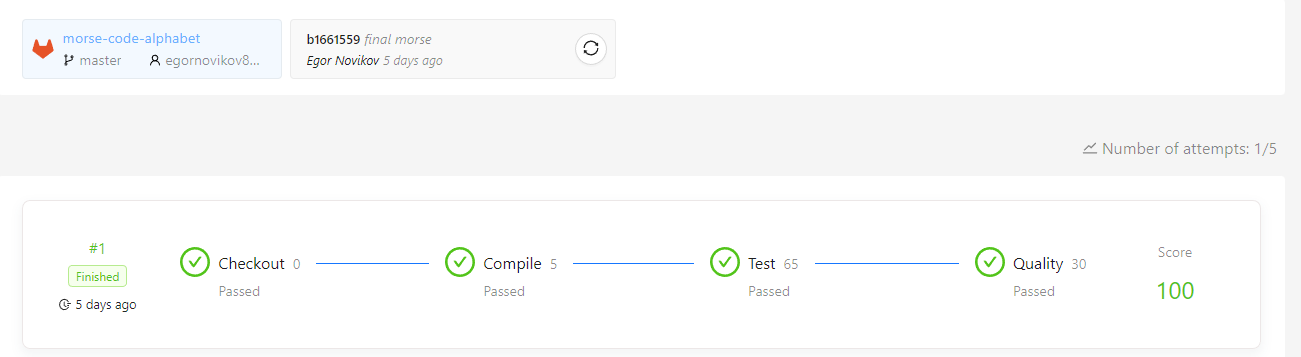
return 0b0000\_0000;

}

}

}

}

Результаты теста:

Задача №7

Название: Selection Statements

Условие: Решить задачи на тренировку if-else, switch statements

Код:

﻿using System;

using System.Reflection.Metadata.Ecma335;

namespace SelectionStatments

{

public static class Statements

{

/// <summary>

/// Writes the largest of three numbers in a separate line in format "Number {0} is the largest".

/// </summary>

/// <param name="first">First number.</param>

/// <param name="second">Second number.</param>

/// <param name="third">Third number.</param>

public static void WriteLargestWithNestedIfElse(int first, int second, int third)

{

if (first > third && first > second)

{

Console.WriteLine($"Number {first} is the largest");

}

else if (second < third && first < third)

{

Console.WriteLine($"Number {third} is the largest");

}

else if (third < second && first < second)

{

Console.WriteLine($"Number {second} is the largest");

}

}

/// <summary>

/// Writes the largest of three numbers in a separate line in format "Number {0} is the largest".

/// </summary>

/// <param name="first">First number.</param>

/// <param name="second">Second number.</param>

/// <param name="third">Third number.</param>

public static void WriteLargestWithIfElseAndTernaryOperator(int first, int second, int third)

{

if (first > third && first > second)

{

Console.WriteLine($"Number {first} is the largest");

}

else if (second < third && first < third)

{

Console.WriteLine($"Number {third} is the largest");

}

else if (third < second && first < second)

{

Console.WriteLine($"Number {second} is the largest");

}

}

/// <summary>

/// Writes the largest of three numbers in a separate line in format "Number {0} is the largest".

/// </summary>

/// <param name="first">First number.</param>

/// <param name="second">Second number.</param>

/// <param name="third">Third number.</param>

public static void WriteLargestWithIfElseAndConditionLogicalOperators(int first, int second, int third)

{

if (first > third && first > second)

{

Console.WriteLine($"Number {first} is the largest");

}

else if (second < third && first < third)

{

Console.WriteLine($"Number {third} is the largest");

}

else if (third < second && first < second)

{

Console.WriteLine($"Number {second} is the largest");

}

}

/// <summary>

/// Writes the reaction to the user's age:

/// - writes "Enjoy your retirement!" if user's age is more or equal 65;

/// - writes "Fancy an alcoholic beverage?" if user's is age more or equal 21;

/// - writes "You're old enough to drive." if user's is age more or equal 18;

/// - writes "You are too young to drive, drink, or retire." otherwise.

/// </summary>

/// <param name="userAge">User's age is more or equals zero.</param>

public static void HowOldAreYouReactionWithCascadedIfElse(int userAge)

{

if (userAge >= 65)

{

Console.WriteLine($"Enjoy your retirement!");

}

else if (userAge >= 21)

{

Console.WriteLine($"Fancy an alcoholic beverage?");

}

else if (userAge >= 18)

{

Console.WriteLine($"You're old enough to drive.");

}

else

{

Console.WriteLine($"You are too young to drive, drink, or retire.");

}

}

/// <summary>

/// Writes the message with information about count of daily downloads:

/// - writes "No downloads." if countOfDailyDownloads is less or equals 0;

/// - writes "Daily downloads: 1-100." if countOfDailyDownloads is less than 100;

/// - writes "Daily downloads: 100-1,000." if countOfDailyDownloads is less than 1000;

/// - writes "Daily downloads: 1,000-10,000." if countOfDailyDownloads is less than 10000;

/// - writes "Daily downloads: 10,000-100,000." if countOfDailyDownloads is less than 100000;

/// - writes "Daily downloads: 100,000+." otherwise.

/// </summary>

/// <param name="countOfDailyDownloads">Count of daily downloads more or equals zero.</param>

public static void WriteInformationAboutDailyDownloadsWithCascadedIfElse(int countOfDailyDownloads)

{

if (countOfDailyDownloads <= 0)

{

Console.WriteLine("No downloads.");

}

else if (countOfDailyDownloads <= 100)

{

Console.WriteLine("Daily downloads: 1-100.");

}

else if (countOfDailyDownloads <= 1000)

{

Console.WriteLine("Daily downloads: 100-1,000.");

}

else if (countOfDailyDownloads <= 10000)

{

Console.WriteLine("Daily downloads: 1,000-10,000.");

}

else if (countOfDailyDownloads <= 100000)

{

Console.WriteLine("Daily downloads: 10,000-100,000.");

}

else

{

Console.WriteLine("Daily downloads: 100,000+.");

}

}

/// <summary>

/// Writes on base on the DayOfWeek enumeration whether a particular date is

/// - a weekend, writes "The weekend.";

/// - the first day of the work week, writes "The first day of the work week.";

/// - the last day of the work week, writes "The last day of the work week.";

/// - the middle of the work week, "The middle of the work week.".

/// </summary>

/// <param name="dayOfWeek">Day of week.</param>

public static void WriteTheInformationAboutDayWithIfElse(DayOfWeek dayOfWeek)

{

if (dayOfWeek == DayOfWeek.Sunday)

{

Console.WriteLine("The weekend.");

}

else if (dayOfWeek == DayOfWeek.Monday)

{

Console.WriteLine("The first day of the work week.");

}

else if (dayOfWeek == DayOfWeek.Tuesday)

{

Console.WriteLine("The middle of the work week.");

}

else if (dayOfWeek == DayOfWeek.Wednesday)

{

Console.WriteLine("The middle of the work week.");

}

else if (dayOfWeek == DayOfWeek.Thursday)

{

Console.WriteLine("The middle of the work week.");

}

else if (dayOfWeek == DayOfWeek.Friday)

{

Console.WriteLine("The last day of the work week.");

}

else if (dayOfWeek == DayOfWeek.Saturday)

{

Console.WriteLine("The weekend.");

}

}

/// <summary>

/// Determines on base on the DayOfWeek enumeration whether a particular date is

/// - a weekend, writes "The weekend.";

/// - the first day of the work week, writes "The first day of the work week.";

/// - the last day of the work week, writes "The last day of the work week.";

/// - the middle of the work week, "The middle of the work week.".

/// </summary>

/// <param name="dayOfWeek">Day of week.</param>

public static void WriteTheInformationAboutDayWithSwitchStatement(DayOfWeek dayOfWeek)

{

switch (dayOfWeek)

{

case DayOfWeek.Sunday:

Console.WriteLine("The weekend.");

break;

case DayOfWeek.Monday:

Console.WriteLine("The first day of the work week.");

break;

case DayOfWeek.Tuesday:

Console.WriteLine("The middle of the work week.");

break;

case DayOfWeek.Wednesday:

Console.WriteLine("The middle of the work week.");

break;

case DayOfWeek.Thursday:

Console.WriteLine("The middle of the work week.");

break;

case DayOfWeek.Friday:

Console.WriteLine("The last day of the work week.");

break;

case DayOfWeek.Saturday:

Console.WriteLine("The weekend.");

break;

}

}

/// <summary>

/// Gets the message with information about the type of integer in format:

/// - "{arg} is sbyte.", if arg is sbyte;

/// - "{arg} is byte.", if arg is byte;

/// - "{arg} is short.", if arg is short;

/// - "{arg} is int.", if arg is int;

/// - "{arg} is long.", if arg is long;

/// - "{arg} is ushort.", if arg is ushort;

/// - "{arg} is uint.", if arg is uint;

/// - "{arg} is ulong.", if arg is ulong.

/// - "{arg} is not integer.", otherwise.

/// </summary>

/// <param name="arg">Presents some integer.</param>

/// <returns>The message with information about the type of integer.</returns>

public static string GetTypeOfIntegerWithCascadedIfElse(object arg)

{

if (arg is sbyte)

{

return $"{arg} is sbyte.";

}

else if (arg is byte)

{

return $"{arg} is byte.";

}

else if (arg is short)

{

return $"{arg} is short.";

}

else if (arg is int)

{

return $"{arg} is int.";

}

else if (arg is long)

{

return $"{arg} is long.";

}

else if (arg is ushort)

{

return $"{arg} is ushort.";

}

else if (arg is uint)

{

return $"{arg} is uint.";

}

else if (arg is ulong)

{

return $"{arg} is ulong.";

}

else

{

return $"{arg} is not integer.";

}

}

/// <summary>

/// Gets the message with information about the type of integer in format:

/// - "{arg} is sbyte.", if arg is sbyte;

/// - "{arg} is byte.", if arg is byte;

/// - "{arg} is short.", if arg is short;

/// - "{arg} is int.", if arg is int;

/// - "{arg} is long.", if arg is long;

/// - "{arg} is ushort.", if arg is ushort;

/// - "{arg} is uint.", if arg is uint;

/// - "{arg} is ulong.", if arg is ulong.

/// - "{arg} is not integer.", otherwise.

/// </summary>

/// <param name="arg">Presents some integer.</param>

/// <returns>The message with information about the type of integer.</returns>

public static string GetTypeOfIntegerWithSwitchStatement(object arg)

{

switch (arg)

{

case sbyte \_:

return $"{arg} is sbyte.";

case byte \_:

return $"{arg} is byte.";

case short \_:

return $"{arg} is short.";

case int \_:

return $"{arg} is int.";

case long \_:

return $"{arg} is long.";

case ushort \_:

return $"{arg} is ushort.";

case uint \_:

return $"{arg} is uint.";

case ulong \_:

return $"{arg} is ulong.";

default:

return $"{arg} is not integer.";

}

}

/// <summary>

/// Gets the message with information about the type of integer in format:

/// - "{arg} is sbyte.", if arg is sbyte;

/// - "{arg} is byte.", if arg is byte;

/// - "{arg} is short.", if arg is short;

/// - "{arg} is int.", if arg is int;

/// - "{arg} is long.", if arg is long;

/// - "{arg} is ushort.", if arg is ushort;

/// - "{arg} is uint.", if arg is uint;

/// - "{arg} is ulong.", if arg is ulong.

/// - "{arg} is not integer.", otherwise.

/// </summary>

/// <param name="arg">Presents some integer.</param>

/// <returns>The message with information about the type of integer.</returns>

public static string GetTypeOfIntegerWithSwitchExpression(object arg)

{

return arg switch

{

sbyte \_ => $"{arg} is sbyte.",

byte \_ => $"{arg} is byte.",

short \_ => $"{arg} is short.",

int \_ => $"{arg} is int.",

long \_ => $"{arg} is long.",

ushort \_ => $"{arg} is ushort.",

uint \_ => $"{arg} is uint.",

ulong \_ => $"{arg} is ulong.",

\_ => $"{arg} is not integer."

};

}

/// <summary>

/// Writes the season that corresponds to the given month:

/// - writes "It's winter now." if month is December, January or February;

/// - writes "It's spring now." if month is March, April or May;

/// - writes "It's summer now." if month is June, July or August;

/// - writes "It's autumn now." if month is September, October or November;

/// - writes "Sorry, the month was entered incorrectly." otherwise.

/// </summary>

/// <param name="month">Source month.</param>

public static void WriteTheInformationAboutSeasonsWithSwitchStatement(Month month)

{

switch (month)

{

case Month.December:

Console.WriteLine("It's winter now.");

break;

case Month.January:

Console.WriteLine("It's winter now.");

break;

case Month.February:

Console.WriteLine("It's winter now.");

break;

case Month.March:

Console.WriteLine("It's spring now.");

break;

case Month.May:

Console.WriteLine("It's spring now.");

break;

case Month.April:

Console.WriteLine("It's spring now.");

break;

case Month.June:

Console.WriteLine("It's summer now.");

break;

case Month.July:

Console.WriteLine("It's summer now.");

break;

case Month.August:

Console.WriteLine("It's summer now.");

break;

case Month.September:

Console.WriteLine("It's autumn now.");

break;

case Month.October:

Console.WriteLine("It's autumn now.");

break;

case Month.November:

Console.WriteLine("It's autumn now.");

break;

default:

Console.WriteLine("Sorry, the month was entered incorrectly.");

break;

}

}

/// <summary>

/// Returns the length of the number (number of digits).

/// </summary>

/// <param name="number">Source integer.</param>

/// <returns>The length of the number.</returns>

public static byte GetLengthWithCascadedIfElse(int number)

{

if ((number >= 0 && number < 10) || (number <= 0 && number > -10))

{

return 1;

}

else if ((number >= 10 && number < 100) || (number <= -10 && number > -100))

{

return 2;

}

else if ((number >= 100 && number < 1000) || (number <= -100 && number > -1000))

{

return 3;

}

else if ((number >= 1000 && number < 10000) || (number <= -1000 && number > -10000))

{

return 4;

}

else if ((number >= 10000 && number < 100000) || (number <= -10000 && number > -100000))

{

return 5;

}

else if ((number >= 100000 && number < 1000000) || (number <= -100000 && number > -1000000))

{

return 6;

}

else if ((number >= 1000000 && number < 10000000) || (number <= -1000000 && number > -10000000))

{

return 7;

}

else if ((number >= 10000000 && number < 100000000) || (number <= -10000000 && number > -100000000))

{

return 8;

}

else if ((number >= 100000000 && number < 1000000000) || (number <= -100000000 && number > -1000000000))

{

return 9;

}

else

{

return 10;

}

}

/// <summary>

/// Returns the length of the number (number of digits).

/// </summary>

/// <param name="number">Source integer.</param>

/// <returns>The length of the number.</returns>

public static byte GetLengthWithSwitchExpression(int number)

{

if ((number >= 0 && number < 10) || (number <= 0 && number > -10))

{

return 1;

}

else if ((number >= 10 && number < 100) || (number <= -10 && number > -100))

{

return 2;

}

else if ((number >= 100 && number < 1000) || (number <= -100 && number > -1000))

{

return 3;

}

else if ((number >= 1000 && number < 10000) || (number <= -1000 && number > -10000))

{

return 4;

}

else if ((number >= 10000 && number < 100000) || (number <= -10000 && number > -100000))

{

return 5;

}

else if ((number >= 100000 && number < 1000000) || (number <= -100000 && number > -1000000))

{

return 6;

}

else if ((number >= 1000000 && number < 10000000) || (number <= -1000000 && number > -10000000))

{

return 7;

}

else if ((number >= 10000000 && number < 100000000) || (number <= -10000000 && number > -100000000))

{

return 8;

}

else if ((number >= 100000000 && number < 1000000000) || (number <= -100000000 && number > -1000000000))

{

return 9;

}

else

{

return 10;

}

}

/// <summary>

/// Returns the value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or

/// null, if integer less than 1 or more than 12.

/// </summary>

/// <param name="month">Source integer.</param>

/// <returns>The value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or null, if integer less than 1 or more than 12.

/// </returns>

public static Month? GetMonthWithCascadedIfElse(int month)

{

if (month == 1)

{

return Month.January;

}

else if (month == 2)

{

return Month.February;

}

else if (month == 3)

{

return Month.March;

}

else if (month == 4)

{

return Month.April;

}

else if (month == 5)

{

return Month.May;

}

else if (month == 6)

{

return Month.June;

}

else if (month == 7)

{

return Month.July;

}

else if (month == 8)

{

return Month.August;

}

else if (month == 9)

{

return Month.September;

}

else if (month == 10)

{

return Month.October;

}

else if (month == 11)

{

return Month.November;

}

else if (month == 12)

{

return Month.December;

}

else

{

return null;

}

}

/// <summary>

/// Returns the value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or

/// null, if integer less than 1 or more than 12.

/// </summary>

/// <param name="month">Source integer.</param>

/// <returns>The value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or null, if integer less than 1 or more than 12.

/// </returns>

public static Month? GetMonthWithSwitchStatement(int month)

{

switch (month)

{

case 1:

return Month.January;

case 2:

return Month.February;

case 3:

return Month.March;

case 4:

return Month.April;

case 5:

return Month.May;

case 6:

return Month.June;

case 7:

return Month.July;

case 8:

return Month.August;

case 9:

return Month.September;

case 10:

return Month.October;

case 11:

return Month.November;

case 12:

return Month.December;

default:

return null;

}

}

/// <summary>

/// Returns the value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or

/// null, if integer less than 1 or more than 12.

/// </summary>

/// <param name="month">Source integer.</param>

/// <returns>The value of enum <see cref="SelectionStatments.Month"/> that corresponds to the given integer

/// - or null, if integer less than 1 or more than 12.

/// </returns>

public static Month? GetMonthWithSwitchExpression(int month)

{

return month switch

{

1 => Month.January,

2 => Month.February,

3 => Month.March,

4 => Month.April,

5 => Month.May,

6 => Month.June,

7 => Month.July,

8 => Month.August,

9 => Month.September,

10 => Month.October,

11 => Month.November,

12 => Month.December,

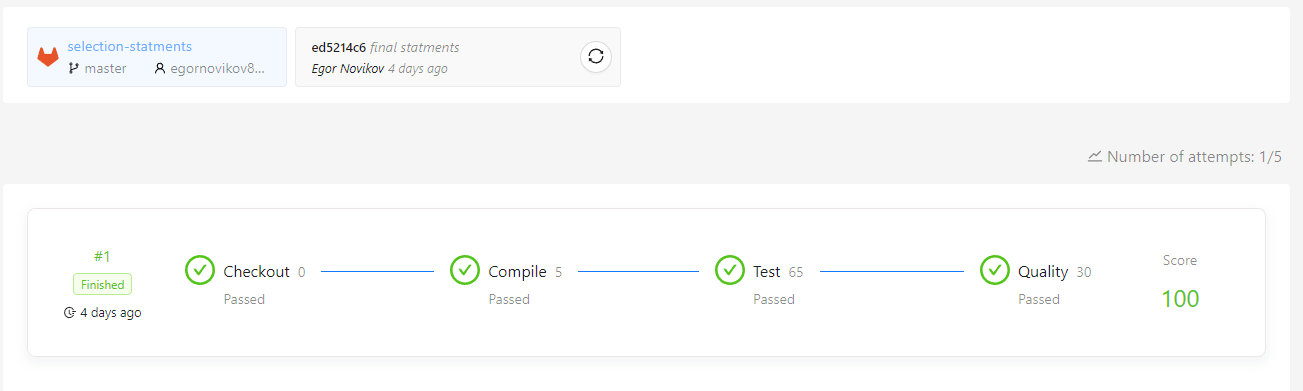
\_ => null

};

}

}

}

Результаты теста:

Задача №8

Название: Darts game

Условие: Решить задачи на тренировку if-else, switch statements

Код:

﻿using System;

namespace DartsGame

{

public static class Darts

{

/// <summary>

/// Calculates the earned points in a single toss of a Darts game.

/// </summary>

/// <param name="x">x-coordinate of dart.</param>

/// <param name="y">y-coordinate of dart.</param>

/// <returns>The earned points.</returns>

public static int GetScore(double x, double y)

{

if (Math.Sqrt((x \* x) + (y \* y)) <= 1)

{

return 10;

}

else if (Math.Sqrt((x \* x) + (y \* y)) <= 5)

{

return 5;

}

else if (Math.Sqrt((x \* x) + (y \* y)) <= 10)

{

return 1;

}

else

{

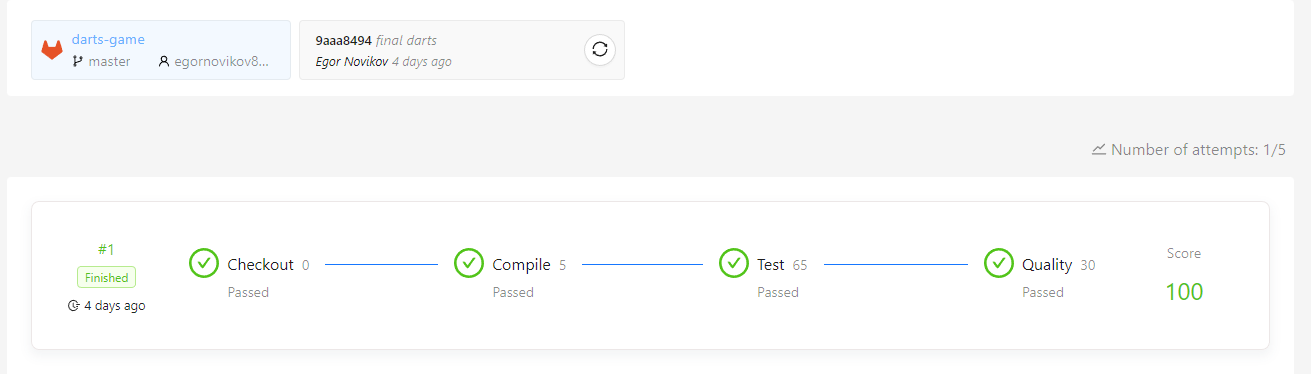
return 0;

}

}

}

}

Результаты теста:

Задача №9

Название: Exception handling

Условие: Научиться обрабатывать исключения

Код:

HandlingExceptions.cs:

﻿using System;

namespace ExceptionHandling

{

public static class HandlingExceptions

{

public static bool CatchException(object obj)

{

try

{

ThrowException(obj);

}

catch

{

return true;

}

return false;

}

public static bool CatchArgumentNullException(object obj, out string exceptionMessage)

{

exceptionMessage = string.Empty;

try

{

ThrowException(obj);

}

catch (ArgumentNullException ex)

{

exceptionMessage = ex.Message;

return true;

}

return false;

}

public static bool CatchArgumentException(int i, out string exceptionMessage)

{

exceptionMessage = string.Empty;

try

{

ThrowException(new object(), i);

}

catch (ArgumentException ex)

{

exceptionMessage = ex.Message;

return true;

}

return false;

}

public static bool CatchArgumentOutOfRangeException(int j, out string exceptionMessage)

{

exceptionMessage = string.Empty;

try

{

ThrowException(new object(), 1, j);

}

catch (ArgumentOutOfRangeException ex)

{

exceptionMessage = ex.Message;

return true;

}

return false;

}

public static bool CatchExceptions(object obj, int i, int j, bool throwException, out string exceptionMessage)

{

exceptionMessage = string.Empty;

try

{

ThrowException(obj, i, j, throwException);

}

catch (ArgumentNullException ex)

{

exceptionMessage = ex.Message;

return true;

}

catch (ArgumentOutOfRangeException ex)

{

exceptionMessage = ex.Message;

return true;

}

catch (ArgumentException ex)

{

exceptionMessage = ex.Message;

return true;

}

catch (Exception ex)

{

exceptionMessage = ex.Message;

return true;

}

return false;

}

private static void ThrowException(object obj, int i = 1, int j = 1, bool throwException = false)

{

if (obj is null)

{

throw new ArgumentNullException(nameof(obj), "obj is null.");

}

if (i == 0)

{

throw new ArgumentException("i parameter is invalid.", nameof(i));

}

if (j < 0 || j > 10)

{

throw new ArgumentOutOfRangeException(nameof(j), "j is out of range.");

}

if (throwException)

{

var exception = new Exception("exception is thrown.");

throw exception;

}

}

}

}

ThrowingExceptions.cs:

using System;

namespace ExceptionHandling

{

public static class ThrowingExceptions

{

public static void CheckParameterAndThrowException(object obj)

{

if (obj == null)

{

throw new ArgumentNullException(nameof(obj));

}

}

public static void CheckBothParametersAndThrowException(object obj1, object obj2)

{

if (obj1 == null || obj2 == null)

{

throw new ArgumentNullException(nameof(obj1));

}

}

public static string CheckStringAndThrowException(string str)

{

if (string.IsNullOrEmpty(str))

{

throw new ArgumentNullException(nameof(str));

}

return str;

}

public static string CheckBothStringsAndThrowException(string str1, string str2)

{

if (string.IsNullOrEmpty(str1))

{

throw new ArgumentNullException(nameof(str1));

}

if (string.IsNullOrEmpty(str2))

{

throw new ArgumentNullException(nameof(str2));

}

string str = string.Concat(str1, str2);

return str;

}

public static int CheckEvenNumberAndThrowException(int evenNumber)

{

if (evenNumber % 2 != 0)

{

throw new ArgumentException(nameof(evenNumber));

}

return evenNumber;

}

public static int CheckCandidateAgeAndThrowException(int candidateAge, bool isCandidateWoman)

{

if (!isCandidateWoman && (candidateAge < 16 || candidateAge > 63))

{

throw new ArgumentOutOfRangeException(nameof(candidateAge));

}

if (isCandidateWoman && (candidateAge < 16 || candidateAge > 58))

{

throw new ArgumentOutOfRangeException(nameof(candidateAge));

}

return candidateAge;

}

public static string GenerateUserCode(int day, int month, string username)

{

if (day < 1 || day > 31)

{

throw new ArgumentOutOfRangeException(nameof(day));

}

if (month < 1 || month > 12)

{

throw new ArgumentOutOfRangeException(nameof(month));

}

if (string.IsNullOrEmpty(username))

{

throw new ArgumentNullException(nameof(username));

}

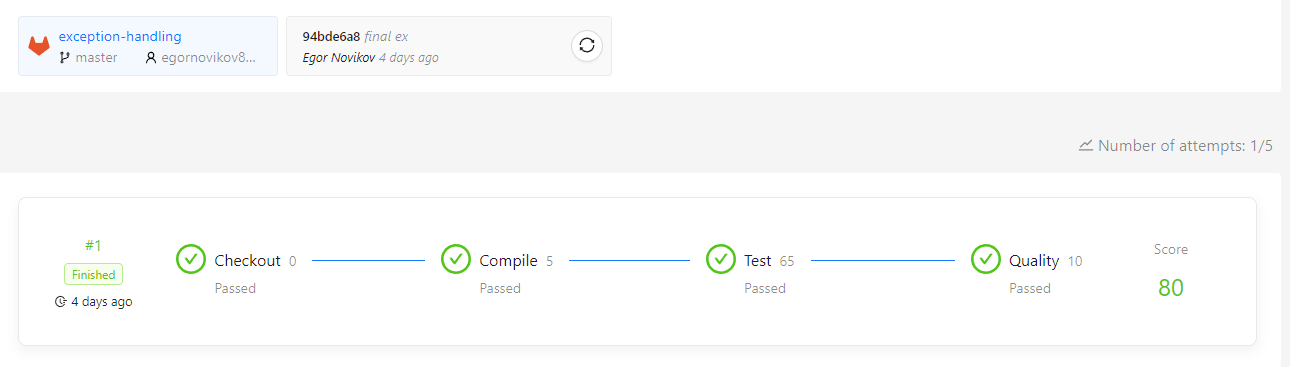
return $"{username}-{day}{month}";

}

}

}

Результаты теста:



Задача №10

Название: Refactoring Exception Guard Classes

Условие: Решить задачи на тренировку обработки исключений

Код:

﻿using System;

using System.Globalization;

namespace ExceptionGuardClauses

{

public static class ThrowingExceptions

{

public static int ConvertHexCharToInteger(char c)

{

if (int.TryParse($"{c}", NumberStyles.HexNumber, CultureInfo.InvariantCulture, out int result))

{

return result;

}

throw new ArgumentException("c is not a hex char.", nameof(c));

}

public static char GetLastCharacter(string str)

{

if (string.IsNullOrEmpty(str))

{

throw new ArgumentNullException(nameof(str));

}

return str[^1];

}

public static string GenerateUserCode(string code, int day)

{

if (string.IsNullOrEmpty(code))

{

throw new ArgumentNullException(nameof(code));

}

if (code.Length != 4)

{

throw new ArgumentException($"{nameof(code)} has {code.Length} chars.", nameof(code));

}

if (day <= 0 || day >= 100)

{

throw new ArgumentOutOfRangeException(nameof(day));

}

return code + day.ToString(CultureInfo.InvariantCulture);

}

public static string GenerateGreeting(string hello, string[] addressee, int index)

{

if (string.IsNullOrEmpty(hello))

{

throw new ArgumentNullException(nameof(hello));

}

if (addressee is null)

{

throw new ArgumentNullException(nameof(addressee));

}

if (addressee.Length == 0)

{

throw new ArgumentException($"{nameof(addressee)} is empty.", nameof(addressee));

}

if (index < 0 || index >= addressee.Length)

{

throw new ArgumentOutOfRangeException(nameof(index));

}

return $"{hello}, {addressee[index]}!";

}

public static string GetArrayValue(int[] indexArray, int indexArrayPosition, string[] valueArray)

{

if (indexArray is null)

{

throw new ArgumentNullException(nameof(indexArray));

}

if (indexArray.Length == 0)

{

throw new ArgumentException($"{nameof(indexArray)} is empty.", nameof(indexArray));

}

if (valueArray is null)

{

throw new ArgumentNullException(nameof(valueArray));

}

if (valueArray.Length == 0)

{

throw new ArgumentException($"{nameof(valueArray)} is empty.", nameof(valueArray));

}

if (indexArrayPosition < 0 || indexArrayPosition >= indexArray.Length)

{

throw new ArgumentOutOfRangeException(nameof(indexArrayPosition));

}

int position = indexArray[indexArrayPosition];

if (position < 0 || position >= valueArray.Length)

{

throw new IndexOutOfRangeException(nameof(position));

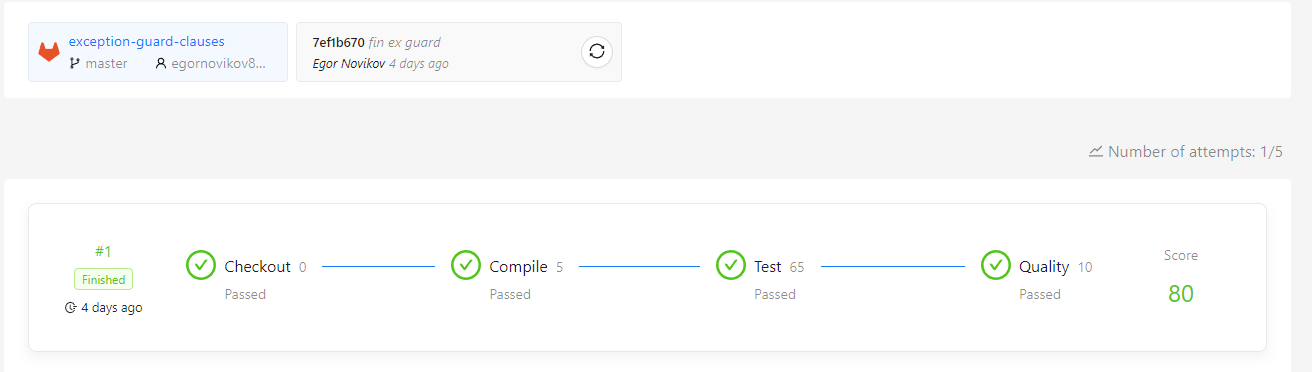
}

return valueArray[position];

}

}

}

Результаты теста:

Задача №11

Название: Leap Year

Условие: Решить задачу на закрепление пройденного материала

Код:

﻿ ﻿using System;

namespace LeapYearTask

{

public static class Year

{

/// <summary>

/// Report if given year is a leap year.

/// </summary>

/// <param name="year">Given year.</param>

/// <returns>True if given year is leap, false otherwise.</returns>

public static bool IsLeapYear(int year)

{

if (year % 4 == 0)

{

if ((year % 100) == 0 && (year % 400) != 0)

{

return false;

}

return true;

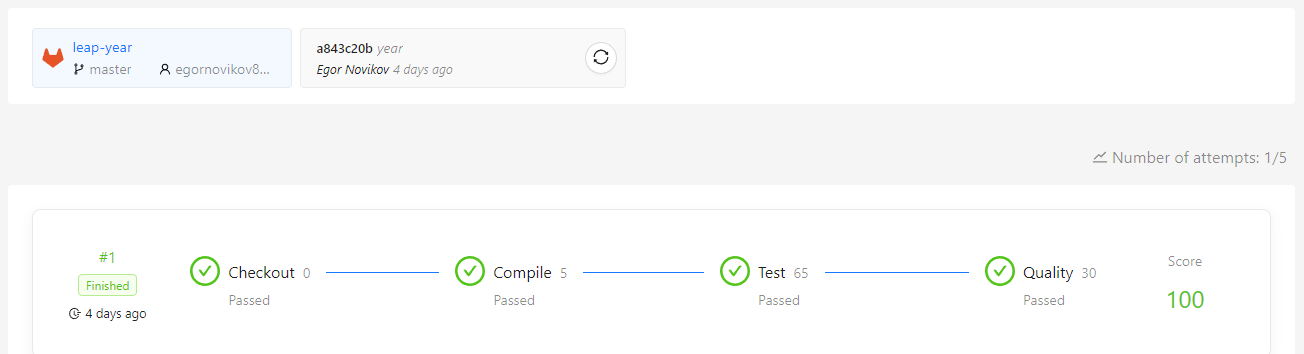
}

return false;

}

}

}

Результаты теста:

Задача №12

Название: Population

Условие: Решить задачу с использованием цикла

Код:

﻿ using System;

namespace PopulationTask

{

public static class Population

{

/// <summary>

/// Calculates the count of years which the town need to see its population greater or equal to currentPopulation inhabitants.

/// </summary>

/// <param name="initialPopulation">The population at the beginning of a year.</param>

/// <param name="percent">The percentage of growth per year.</param>

/// <param name="visitors">The visitors (new inhabitants per year) who come to live in the town.</param>

/// <param name="currentPopulation">The population at present.</param>

/// <returns>The count of years which the town need to see its population greater or equal to currentPopulation inhabitants.</returns>

/// <exception cref="ArgumentException">

/// Thrown when initial population is less or equals 0

/// - or -

/// the count of visitors cannot be less 0

/// - or -

/// the current population is less or equals 0

/// - or -

/// the current population is less than initial population.

/// </exception>

/// <exception cref="ArgumentOutOfRangeException">Throw if the value of percents is less then 0% or more then 100%.</exception>

public static int GetYears(int initialPopulation, double percent, int visitors, int currentPopulation)

{

if (initialPopulation <= 0)

{

throw new ArgumentException("initial population is less or equals 0");

}

if (visitors < 0)

{

throw new ArgumentException("the count of visitors cannot be less 0");

}

if (currentPopulation <= 0)

{

throw new ArgumentException("the current population is less or equals 0");

}

if (currentPopulation <= initialPopulation)

{

throw new ArgumentException("the current population is less than initial population.");

}

if (percent < 0 || percent > 100)

{

throw new ArgumentOutOfRangeException(nameof(percent));

}

int years = 0;

double population = initialPopulation;

while (population < currentPopulation)

{

population += percent / 100 \* population;

population += visitors;

years++;

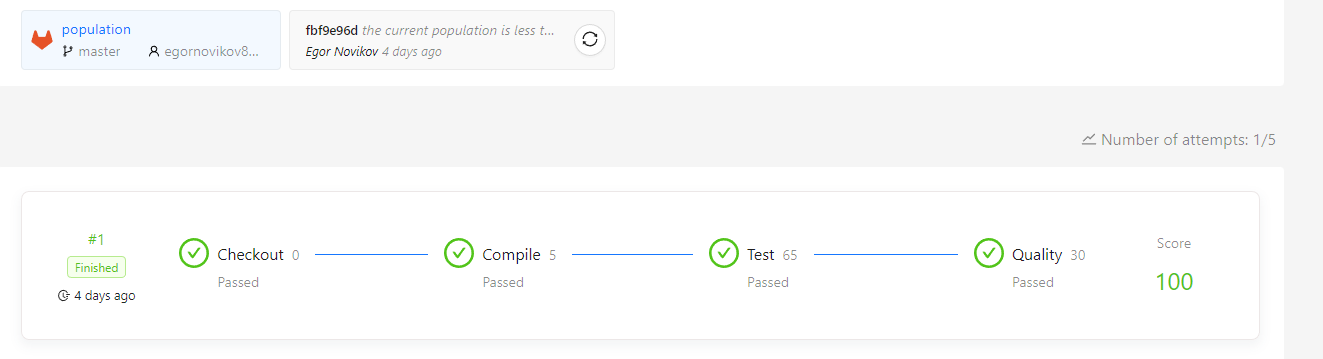
}

return years;

}

}

}

Результаты теста:

Задача №13

Название: Arithmetic Sequence

Условие: Решить задачу с использованием цикла

Код:

using System;

namespace ArithmeticSequenceTask

{

public static class ArithmeticSequence

{

/// <summary>

/// Calculates the sum of the first 'count' elements of a sequence in which each element is the sum of the given integer 'number'

/// and number of occurrences of the given integer 'add', based on the element's position within the sequence.

/// </summary>

/// <param name="number">Source number.</param>

/// <param name="add">The term.</param>

/// <param name="count">The number of occurrences.</param>

/// <returns>Calculated sum.</returns>

/// <exception cref="OverflowException">

/// Thrown when number is int.MaxValue and term more then 0

/// - or -

/// number is int.MinValue and term less then 0.

/// </exception>

/// <exception cref="ArgumentException">Throw if count less than zero.</exception>

public static int Calculate(int number, int add, int count)

{

if (number == int.MinValue)

{

throw new OverflowException();

}

if (number == int.MaxValue)

{

throw new OverflowException();

}

if (count < 0)

{

throw new ArgumentException("count less than zero.");

}

int result = 0;

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

{

result += number + (i \* add);

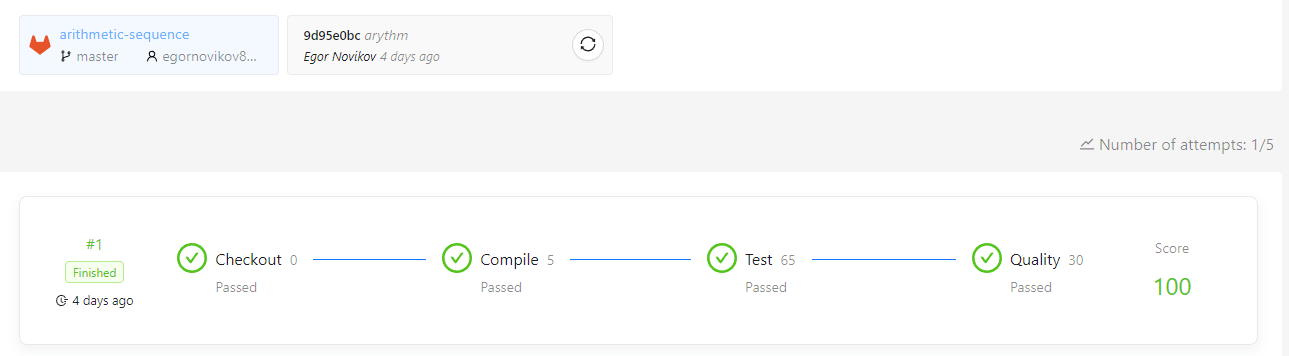
}

return result;

}

}

}

Результаты теста:

Задача №14

Название: Calculations

Условие: Решить задачи с использованием циклов

Код:

﻿using System;

using System.Data;

namespace Calculations

{

public static class Calculator

{

/// <summary>

/// Calculate the following sum 1/1 + 1/2 + 1/3 + ... + 1/n, where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumOne(int n)

{

double sum = 0;

for (double i = 1; i < n + 1; i++)

{

sum += 1 / i;

}

return sum;

}

/// <summary>

/// Calculate the following sum

/// 1/(1\*2) - 1/(2\*3) + 1/(3\*4) + ... + (-1)^(n+1) / (n \* (n + 1)), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumTwo(int n)

{

double sum = 0;

for (double i = 1; i < n + 1; i++)

{

sum += Math.Pow(-1, i + 1) / (i \* (i + 1));

}

return sum;

}

/// <summary>

/// Calculate the following sum

/// 1/1^5 + 1/2^5 + 1/3^5 + ... + 1/n^5, where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumThree(int n)

{

double sum = 0;

for (double i = 1; i < n + 1; i++)

{

sum += 1 / Math.Pow(i, 5);

}

return sum;

}

/// <summary>

/// Calculate the following sum

/// 1/(3 \* 3) + 1/(5 \* 5) + 1/(7 \* 7) + ... + 1/((2 \* n + 1) \* (2 \* n + 1)), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumFour(int n)

{

double sum = 0;

for (double i = 3; i <= (2 \* n) + 1; i += 2)

{

sum += 1 / (i \* i);

}

return sum;

}

/// <summary>

/// Calculate the following product

/// (1 + 1/(1 \* 1)) \* (1 + 1/(2 \* 2)) \* (1 + 1/(3 \* 3)) \* ... \* (1 + 1/(n \* n)), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Product of elements.</returns>

public static double GetProductOne(int n)

{

double sum = 1;

for (double i = 1; i <= n; i += 1)

{

sum \*= 1 + (1 / (i \* i));

}

return sum;

}

/// <summary>

/// Calculate the following sum

/// -1/3 + 1/5 - 1/7 + ... + (-1)^n / (2 \* n + 1), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumFive(int n)

{

double sum = 0;

for (double i = 3; i <= (2 \* n) + 1; i += 2)

{

sum += Math.Pow(-1, (int)(i / 2)) / i;

}

return sum;

}

/// <summary>

/// Calculate the following sum

/// 1!/1 + 2!/(1+1/2) + 3!/(1+1/2+1/3) + ... + 1\*2\*...\* n/ (1+1/2+1/3+...+1/n), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumSix(int n)

{

double factorial = 1;

double denominator = 1;

double rez = 1;

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

{

factorial \*= i;

denominator += 1.0d / i;

rez += factorial / denominator;

}

return rez;

}

/// <summary>

/// Calculate the following sum

/// Sqrt(2 + Sqrt(2 + ... + Sqrt(2))), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumSeven(int n)

{

double first = Math.Sqrt(2);

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

{

first = Math.Sqrt(2 + first);

}

return first;

}

/// <summary>

/// Calculate the following sum

/// 1/sin(1) + 1/(sin(1)+sin(2)) + ...+ 1/(sin(1)+sin(2)+...+sin(n)), where n > 0.

/// </summary>

/// <param name="n">Number of elements.</param>

/// <returns>Sum of elements.</returns>

public static double GetSumEight(int n)

{

double sum = 0;

double sumsum = 0;

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

{

sum += Math.Sin(i \* (Math.PI / 180));

sumsum += 1 / sum;

}

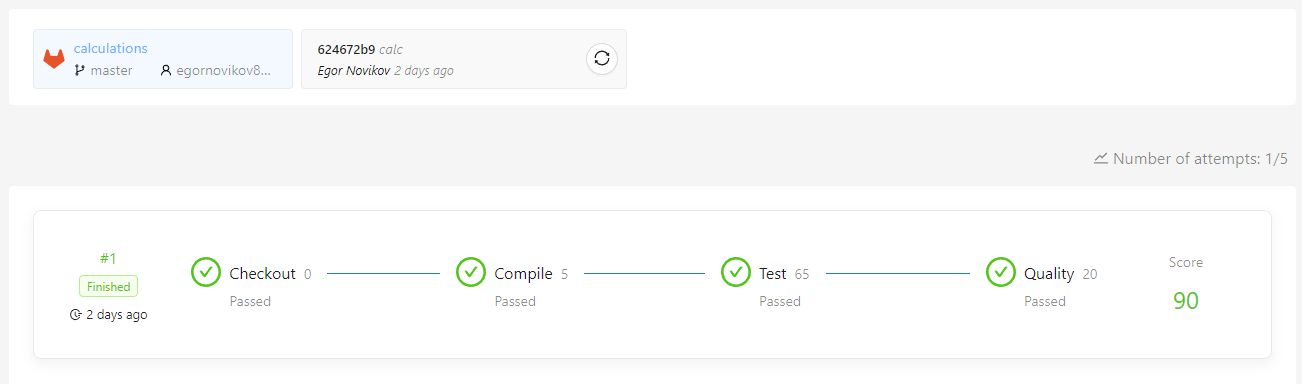
return sumsum;

}

}

}

Результаты теста:



Задача №15

Название: Next bigger

Условие: Решить задачу с использованием цикла

Код:

﻿ using System;

using System.Globalization;

namespace NextBiggerTask

{

public static class NumberExtension

{

/// <summary>

/// Finds the nearest largest integer consisting of the digits of the given positive integer number and null if no such number exists.

/// </summary>

/// <param name="number">Source number.</param>

/// <returns>

/// The nearest largest integer consisting of the digits of the given positive integer and null if no such number exists.

/// </returns>

/// <exception cref="ArgumentException">Thrown when source number is less than 0.</exception>

public static int? NextBiggerThan(int number)

{

var nums = number.ToString(CultureInfo.InvariantCulture).ToCharArray();

if (number < 0 || number == int.MinValue)

{

throw new ArgumentException($"Value of {nameof(number)} cannot be less zero.");

}

else if (number == int.MaxValue)

{

return null;

}

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

{

if (nums[i - 1] < nums[i])

{

int temp = nums[i];

nums[i] = nums[i - 1];

nums[i - 1] = (char)temp;

for (int k = i; k < nums.Length - 1; k++)

{

for (int j = k + 1; j < nums.Length; j++)

{

if (nums[k] > nums[j])

{

int t = nums[k];

nums[k] = nums[j];

nums[j] = (char)t;

}

}

}

return int.Parse(nums);

}

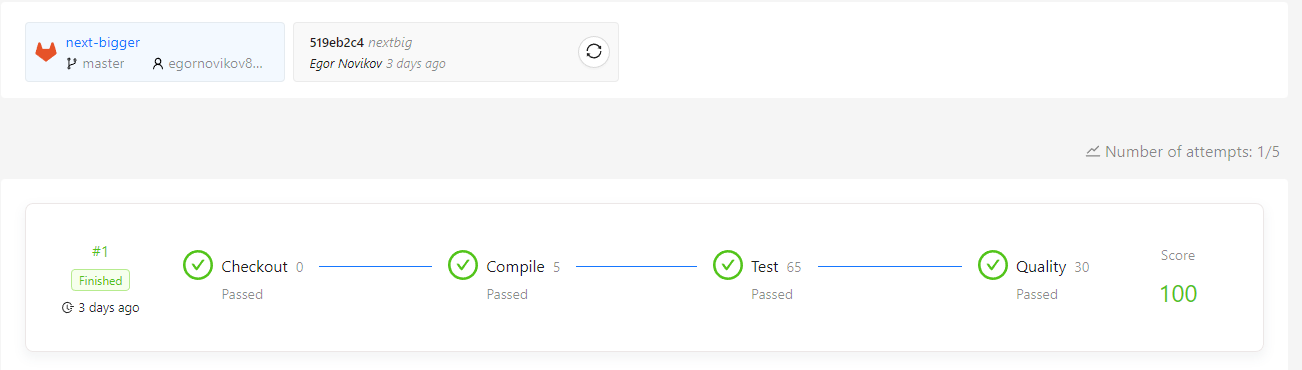
}

return null;

}

}

}

Результаты теста:

Задача №16

Название: Gcd

Условие: Решить задачу на поиск НОД с использованием цикла

Код:

﻿using System;

namespace GcdTask

{

public static class IntegerExtensions

{

/// <summary>

/// Calculates GCD of two integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or two numbers are int.MinValue.</exception>

public static int FindGcd(int a, int b)

{

if (a == 0 && b == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

int result = 0;

int minoftwo = 0;

if (a == b)

{

return a;

}

if (a == 0)

{

return Math.Abs(b);

}

if (b == 0)

{

return Math.Abs(a);

}

if (a < 0)

{

a = Math.Abs(a);

}

if (b < 0)

{

b = Math.Abs(b);

}

if (a > b)

{

minoftwo = b;

}

else

{

minoftwo = a;

}

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

{

if (a % i == 0 && b % i == 0)

{

result = i;

}

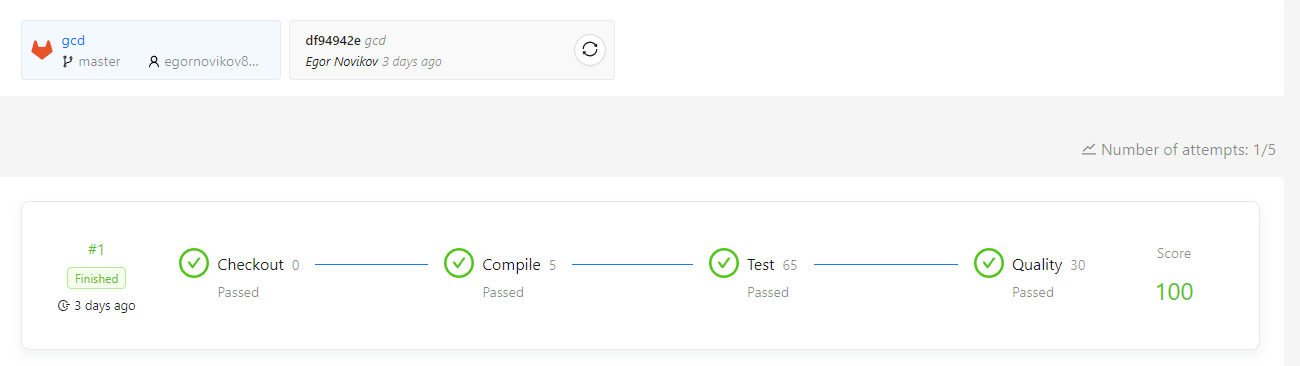
}

return result;

}

}

}

Результаты теста:

Задача №17

Название: Gcd. Variable parameter’s number

Условие: Решить задачи на поиск НОД с использованием определённых алгоритмов

Код:

﻿using System;

namespace Gcd

{

/// <summary>

/// Provide methods with integers.

/// </summary>

public static class IntegerExtensions

{

/// <summary>

/// Calculates GCD of two integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or two numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(int a, int b)

{

if (a == 0 && b == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

}

return a + b;

}

/// <summary>

/// Calculates GCD of three integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="c">Third integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(int a, int b, int c)

{

if (a == 0 && b == 0 && c == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue || c == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0 || c < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

c = Math.Abs(c);

}

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

}

int temp = a + b;

while (temp != 0 && c != 0)

{

if (temp > c)

{

temp = temp % c;

}

else

{

c = c % temp;

}

}

return temp + c;

}

/// <summary>

/// Calculates the GCD of integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="other">Other integers.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(int a, int b, params int[] other)

{

if (a == 0 && b == 0)

{

bool allisnull = true;

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

{

if (other[i] != 0)

{

allisnull = false;

break;

}

}

if (allisnull)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

bool isanymin = false;

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

{

if (other[i] == int.MinValue)

{

isanymin = true;

break;

}

}

if (isanymin)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

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

{

other[i] = Math.Abs(other[i]);

}

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

}

int temp = a + b;

int size = other.Length;

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

{

while (temp != 0 && other[i] != 0)

{

if (temp > other[i])

{

temp = temp % other[i];

}

else

{

other[i] = other[i] % temp;

}

}

temp = temp + other[i];

}

return temp;

}

/// <summary>

/// Calculates GCD of two integers [-int.MaxValue;int.MaxValue] by the Stein algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or two numbers are int.MinValue.</exception>

public static int GetGcdByStein(int a, int b)

{

if (a == 0 && b == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

if (a == 0)

{

return b;

}

if (b == 0)

{

return a;

}

if (a == b)

{

return a;

}

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

return GetGcdByStein(a >> 1, b >> 1) << 1;

}

else if (aIsEven && !bIsEven)

{

return GetGcdByStein(a >> 1, b);

}

else if (bIsEven)

{

return GetGcdByStein(a, b >> 1);

}

else if (a > b)

{

return GetGcdByStein((a - b) >> 1, b);

}

else

{

return GetGcdByStein(a, (b - a) >> 1);

}

}

/// <summary>

/// Calculates GCD of three integers [-int.MaxValue;int.MaxValue] by the Stein algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="c">Third integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByStein(int a, int b, int c)

{

int tmp = a;

a = c;

c = tmp;

if (a == 0 && b == 0 && c == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue || c == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0 || c < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

c = Math.Abs(c);

}

int temp;

if (a == 0)

{

temp = b;

}

else if (b == 0)

{

temp = a;

}

else if (a == b)

{

temp = a;

}

else

{

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

temp = GetGcdByStein(a >> 1, b >> 1) << 1;

}

else if (aIsEven && !bIsEven)

{

temp = GetGcdByStein(a >> 1, b);

}

else if (bIsEven)

{

temp = GetGcdByStein(a, b >> 1);

}

else if (a > b)

{

temp = GetGcdByStein((a - b) >> 1, b);

}

else

{

temp = GetGcdByStein(a, (b - a) >> 1);

}

}

bool tempIsEven = (temp & 1u) == 0;

bool cIsEven = (c & 1u) == 0;

if (tempIsEven && cIsEven)

{

return GetGcdByStein(temp >> 1, c >> 1) << 1;

}

else if (tempIsEven && !cIsEven)

{

return GetGcdByStein(temp >> 1, c);

}

else if (cIsEven)

{

return GetGcdByStein(temp, c >> 1);

}

else if (temp > c)

{

return GetGcdByStein((temp - c) >> 1, c);

}

else

{

return GetGcdByStein(temp, (c - temp) >> 1);

}

}

/// <summary>

/// Calculates the GCD of integers [-int.MaxValue;int.MaxValue] by the Stein algorithm.

/// </summary>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="other">Other integers.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByStein(int a, int b, params int[] other)

{

int tmp = a;

a = other[other.Length - 1];

other[other.Length - 1] = tmp;

if (a == 0 && b == 0)

{

bool allisnull = true;

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

{

if (other[i] != 0)

{

allisnull = false;

break;

}

}

if (allisnull)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

bool isanymin = false;

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

{

if (other[i] == int.MinValue)

{

isanymin = true;

break;

}

}

if (isanymin)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

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

{

other[i] = Math.Abs(other[i]);

}

int temp;

if (a == 0)

{

temp = b;

}

else if (b == 0)

{

temp = a;

}

else if (a == b)

{

temp = a;

}

else

{

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

temp = GetGcdByStein(a >> 1, b >> 1) << 1;

}

else if (aIsEven && !bIsEven)

{

temp = GetGcdByStein(a >> 1, b);

}

else if (bIsEven)

{

temp = GetGcdByStein(a, b >> 1);

}

else if (a > b)

{

temp = GetGcdByStein((a - b) >> 1, b);

}

else

{

temp = GetGcdByStein(a, (b - a) >> 1);

}

}

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

{

bool tempIsEven = (temp & 1u) == 0;

bool otherIsEven = (other[i] & 1u) == 0;

if (tempIsEven && otherIsEven)

{

temp = GetGcdByStein(temp >> 1, other[i] >> 1) << 1;

}

else if (tempIsEven && !otherIsEven)

{

temp = GetGcdByStein(temp >> 1, other[i]);

}

else if (otherIsEven)

{

temp = GetGcdByStein(temp, other[i] >> 1);

}

else if (temp > other[i])

{

temp = GetGcdByStein((temp - other[i]) >> 1, other[i]);

}

else

{

temp = GetGcdByStein(temp, (other[i] - temp) >> 1);

}

}

return temp;

}

/// <summary>

/// Calculates GCD of two integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or two numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(out long elapsedTicks, int a, int b)

{

if (a == 0 && b == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

elapsedTicks = 0;

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

elapsedTicks++;

}

return a + b;

}

/// <summary>

/// Calculates GCD of three integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="c">Third integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(out long elapsedTicks, int a, int b, int c)

{

if (a == 0 && b == 0 && c == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue || c == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0 || c < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

c = Math.Abs(c);

}

elapsedTicks = 0;

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

elapsedTicks++;

}

int temp = a + b;

while (temp != 0 && c != 0)

{

if (temp > c)

{

temp = temp % c;

}

else

{

c = c % temp;

}

elapsedTicks++;

}

return temp + c;

}

/// <summary>

/// Calculates the GCD of integers from [-int.MaxValue;int.MaxValue] by the Euclidean algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in Ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="other">Other integers.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByEuclidean(out long elapsedTicks, int a, int b, params int[] other)

{

if (a == 0 && b == 0)

{

bool allisnull = true;

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

{

if (other[i] != 0)

{

allisnull = false;

break;

}

}

if (allisnull)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

bool isanymin = false;

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

{

if (other[i] == int.MinValue)

{

isanymin = true;

break;

}

}

if (isanymin)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

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

{

other[i] = Math.Abs(other[i]);

}

elapsedTicks = 0;

while (a != 0 && b != 0)

{

if (a > b)

{

a = a % b;

}

else

{

b = b % a;

}

elapsedTicks++;

}

int temp = a + b;

int size = other.Length;

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

{

while (temp != 0 && other[i] != 0)

{

if (temp > other[i])

{

temp = temp % other[i];

}

else

{

other[i] = other[i] % temp;

}

elapsedTicks++;

}

temp = temp + other[i];

}

return temp;

}

/// <summary>

/// Calculates GCD of two integers from [-int.MaxValue;int.MaxValue] by the Stein algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or two numbers are int.MinValue.</exception>

public static int GetGcdByStein(out long elapsedTicks, int a, int b)

{

if (a == 0 && b == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

elapsedTicks = 0;

if (a == 0)

{

elapsedTicks++;

return b;

}

if (b == 0)

{

elapsedTicks++;

return a;

}

if (a == b)

{

elapsedTicks++;

return a;

}

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

elapsedTicks++;

return GetGcdByStein(a >> 1, b >> 1) << 1;

}

else if (aIsEven && !bIsEven)

{

elapsedTicks++;

return GetGcdByStein(a >> 1, b);

}

else if (bIsEven)

{

elapsedTicks++;

return GetGcdByStein(a, b >> 1);

}

else if (a > b)

{

elapsedTicks++;

return GetGcdByStein((a - b) >> 1, b);

}

else

{

elapsedTicks++;

return GetGcdByStein(a, (b - a) >> 1);

}

}

/// <summary>

/// Calculates GCD of three integers from [-int.MaxValue;int.MaxValue] by the Stein algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="c">Third integer.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByStein(out long elapsedTicks, int a, int b, int c)

{

int tmp = a;

a = c;

c = tmp;

if (a == 0 && b == 0 && c == 0)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

if (a == int.MinValue || b == int.MinValue || c == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0 || c < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

c = Math.Abs(c);

}

int temp;

elapsedTicks = 0;

if (a == 0)

{

temp = b;

elapsedTicks++;

}

else if (b == 0)

{

temp = a;

elapsedTicks++;

}

else if (a == b)

{

temp = a;

}

else

{

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

elapsedTicks++;

temp = GetGcdByStein(a >> 1, b >> 1) << 1;

}

else if (aIsEven && !bIsEven)

{

elapsedTicks++;

temp = GetGcdByStein(a >> 1, b);

}

else if (bIsEven)

{

elapsedTicks++;

temp = GetGcdByStein(a, b >> 1);

}

else if (a > b)

{

elapsedTicks++;

temp = GetGcdByStein((a - b) >> 1, b);

}

else

{

elapsedTicks++;

temp = GetGcdByStein(a, (b - a) >> 1);

}

}

bool tempIsEven = (temp & 1u) == 0;

bool cIsEven = (c & 1u) == 0;

if (tempIsEven && cIsEven)

{

elapsedTicks++;

return GetGcdByStein(temp >> 1, c >> 1) << 1;

}

else if (tempIsEven && !cIsEven)

{

elapsedTicks++;

return GetGcdByStein(temp >> 1, c);

}

else if (cIsEven)

{

elapsedTicks++;

return GetGcdByStein(temp, c >> 1);

}

else if (temp > c)

{

elapsedTicks++;

return GetGcdByStein((temp - c) >> 1, c);

}

else

{

elapsedTicks++;

return GetGcdByStein(temp, (c - temp) >> 1);

}

}

/// <summary>

/// Calculates the GCD of integers from [-int.MaxValue;int.MaxValue] by the Stein algorithm with elapsed time.

/// </summary>

/// <param name="elapsedTicks">Method execution time in Ticks.</param>

/// <param name="a">First integer.</param>

/// <param name="b">Second integer.</param>

/// <param name="other">Other integers.</param>

/// <returns>The GCD value.</returns>

/// <exception cref="ArgumentException">Thrown when all numbers are 0 at the same time.</exception>

/// <exception cref="ArgumentOutOfRangeException">Thrown when one or more numbers are int.MinValue.</exception>

public static int GetGcdByStein(out long elapsedTicks, int a, int b, params int[] other)

{

int tmp = a;

a = other[other.Length - 1];

other[other.Length - 1] = tmp;

if (a == 0 && b == 0)

{

bool allisnull = true;

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

{

if (other[i] != 0)

{

allisnull = false;

break;

}

}

if (allisnull)

{

throw new ArgumentException("all numbers are 0 at the same time.");

}

}

if (a == int.MinValue || b == int.MinValue)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

bool isanymin = false;

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

{

if (other[i] == int.MinValue)

{

isanymin = true;

break;

}

}

if (isanymin)

{

throw new ArgumentOutOfRangeException(nameof(a));

}

if (a < 0 || b < 0)

{

a = Math.Abs(a);

b = Math.Abs(b);

}

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

{

other[i] = Math.Abs(other[i]);

}

elapsedTicks = 0;

int temp;

if (a == 0)

{

temp = b;

elapsedTicks++;

}

else if (b == 0)

{

temp = a;

elapsedTicks++;

}

else if (a == b)

{

temp = a;

}

else

{

bool aIsEven = (a & 1u) == 0;

bool bIsEven = (b & 1u) == 0;

if (aIsEven && bIsEven)

{

temp = GetGcdByStein(a >> 1, b >> 1) << 1;

elapsedTicks++;

}

else if (aIsEven && !bIsEven)

{

temp = GetGcdByStein(a >> 1, b);

elapsedTicks++;

}

else if (bIsEven)

{

temp = GetGcdByStein(a, b >> 1);

elapsedTicks++;

}

else if (a > b)

{

temp = GetGcdByStein((a - b) >> 1, b);

elapsedTicks++;

}

else

{

temp = GetGcdByStein(a, (b - a) >> 1);

elapsedTicks++;

}

}

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

{

bool tempIsEven = (temp & 1u) == 0;

bool otherIsEven = (other[i] & 1u) == 0;

if (tempIsEven && otherIsEven)

{

temp = GetGcdByStein(temp >> 1, other[i] >> 1) << 1;

elapsedTicks++;

}

else if (tempIsEven && !otherIsEven)

{

temp = GetGcdByStein(temp >> 1, other[i]);

elapsedTicks++;

}

else if (otherIsEven)

{

temp = GetGcdByStein(temp, other[i] >> 1);

elapsedTicks++;

}

else if (temp > other[i])

{

temp = GetGcdByStein((temp - other[i]) >> 1, other[i]);

elapsedTicks++;

}

else

{

temp = GetGcdByStein(temp, (other[i] - temp) >> 1);

elapsedTicks++;

}

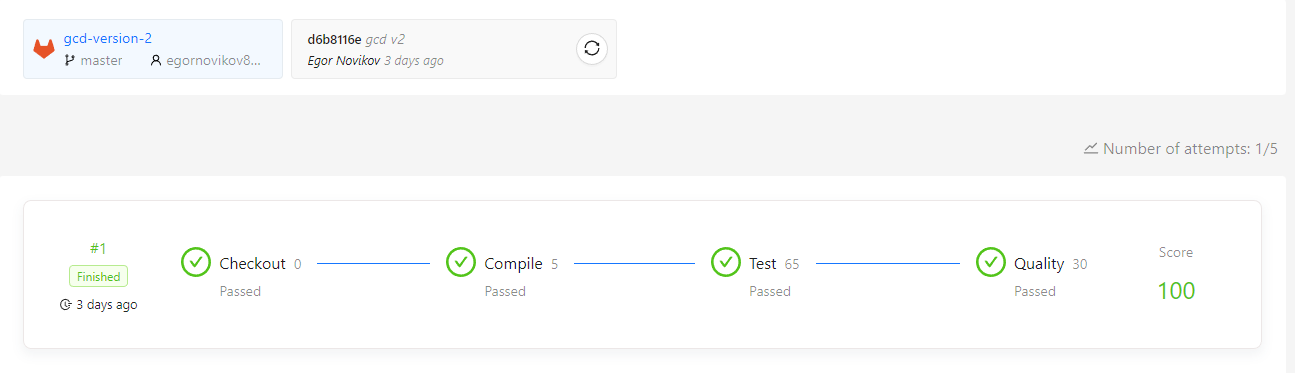
}

return temp;

}

}

}

Результаты теста:

Задача №18

Название: Working with arrays

Условие: Научиться работать с массивами

Код:

﻿CreatingArray.cs:

using System;

namespace WorkingWithArrays

{

public static class CreatingArray

{

public static int[] CreateEmptyArrayOfIntegers()

{

return Array.Empty<int>();

}

public static bool[] CreateEmptyArrayOfBooleans()

{

return Array.Empty<bool>();

}

public static string[] CreateEmptyArrayOfStrings()

{

return Array.Empty<string>();

}

public static char[] CreateEmptyArrayOfCharacters()

{

return Array.Empty<char>();

}

public static double[] CreateEmptyArrayOfDoubles()

{

return Array.Empty<double>();

}

public static float[] CreateEmptyArrayOfFloats()

{

return Array.Empty<float>();

}

public static decimal[] CreateEmptyArrayOfDecimals()

{

return Array.Empty<decimal>();

}

public static int[] CreateArrayOfTenIntegersWithDefaultValues()

{

int[] arr = new int[10];

return arr;

}

public static bool[] CreateArrayOfTwentyBooleansWithDefaultValues()

{

bool[] arr = new bool[20];

return arr;

}

public static string[] CreateArrayOfFiveEmptyStrings()

{

string[] arr = new string[5];

return arr;

}

public static char[] CreateArrayOfFifteenCharactersWithDefaultValues()

{

char[] arr = new char[15];

return arr;

}

public static double[] CreateArrayOfEighteenDoublesWithDefaultValues()

{

double[] arr = new double[18];

return arr;

}

public static float[] CreateArrayOfOneHundredFloatsWithDefaultValues()

{

float[] arr = new float[100];

return arr;

}

public static decimal[] CreateArrayOfOneThousandDecimalsWithDefaultValues()

{

decimal[] arr = new decimal[1000];

return arr;

}

public static int[] CreateIntArrayWithOneElement()

{

var a = new[] { 123456 };

return a;

}

public static int[] CreateIntArrayWithTwoElements()

{

var a = new[] { 1111111, 9999999 };

return a;

}

public static int[] CreateIntArrayWithTenElements()

{

var a = new[] { 0, 4234, 3845, 2942, 1104, 9794, 923943, 7537, 4162, 10134 };

return a;

}

public static bool[] CreateBoolArrayWithOneElement()

{

var a = new[] { true };

return a;

}

public static bool[] CreateBoolArrayWithFiveElements()

{

var a = new[] { true, false, true, false, true };

return a;

}

public static bool[] CreateBoolArrayWithSevenElements()

{

var a = new[] { false, true, true, false, true, true, false };

return a;

}

public static string[] CreateStringArrayWithOneElement()

{

var a = new[] { "one" };

return a;

}

public static string[] CreateStringArrayWithThreeElements()

{

var a = new[] { "one", "two", "three" };

return a;

}

public static string[] CreateStringArrayWithSixElements()

{

var a = new[] { "one", "two", "three", "four", "five", "six" };

return a;

}

public static char[] CreateCharArrayWithOneElement()

{

var a = new[] { 'a' };

return a;

}

public static char[] CreateCharArrayWithThreeElements()

{

var a = new[] { 'a', 'b', 'c' };

return a;

}

public static char[] CreateCharArrayWithNineElements()

{

var a = new[] { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'a' };

return a;

}

public static double[] CreateDoubleArrayWithOneElement()

{

var a = new[] { 1.12 };

return a;

}

public static double[] CreateDoubleWithFiveElements()

{

// TODO #1-28. Add the method implementation. The method should return an array that contains five elements:

// 1.12; 2.23; 3.34; 4.45; 5.56

var a = new[] { 1.12, 2.23, 3.34, 4.45, 5.56 };

return a;

}

public static double[] CreateDoubleWithNineElements()

{

// TODO #1-29. Add the method implementation. The method should return an array that contains nine elements:

// 1.12; 2.23; 3.34; 4.45; 5.56; 6.67; 7.78; 8.89; 9.91

var a = new[] { 1.12, 2.23, 3.34, 4.45, 5.56, 6.67, 7.78, 8.89, 9.91 };

return a;

}

public static float[] CreateFloatArrayWithOneElement()

{

// TODO #1-30. Add the method implementation. The method should return an array that contains one float: 123,456,789.123456.

float[] a = new float[] { 123456789.123456F };

return a;

}

public static float[] CreateFloatWithThreeElements()

{

// TODO #1-31. Add the method implementation. The method should return an array that contains three floats:

// 1,000,000.123456; 2,223,334,444.123456; 9,999.999999

float[] a = new float[] { 1000000.123456F, 2223334444.123456F, 9999.999999F };

return a;

}

public static float[] CreateFloatWithFiveElements()

{

// TODO #1-32. Add the method implementation. The method should return an array that contains five floats:

// 1.0123; 20.012345; 300.01234567; 4,000.01234567; 500,000.01234567

float[] a = new float[] { 1.0123F, 20.012345F, 300.01234567F, 4000.01234567F, 500000.01234567F };

return a;

}

public static decimal[] CreateDecimalArrayWithOneElement()

{

// TODO #1-33. Add the method implementation. The method should return an array that contains one element: 10,000.123456.

decimal[] a = new decimal[] { 10000.123456M };

return a;

}

public static decimal[] CreateDecimalWithFiveElements()

{

// TODO #1-34. Add the method implementation. The method should return an array that contains five elements:

// 1,000.1234; 100,000.2345; 100,000.3456; 1,000,000.456789; 10,000,000.5678901

decimal[] a = new decimal[] { 1000.1234M, 100000.2345M, 100000.3456M, 1000000.456789M, 10000000.5678901M };

return a;

}

public static decimal[] CreateDecimalWithNineElements()

{

// TODO #1-35. Add the method implementation. The method should return an array that contains nine elements:

// 10.122112; 200.233223; 3,000.344334; 40,000.455445; 500,000.566556; 6,000,000.677667; 70,000,000.788778; 800,000,000.899889; 9,000,000,000.911991

decimal[] a = new decimal[] { 10.122112M, 200.233223M, 3000.344334M, 40000.455445M, 500000.566556M, 6000000.677667M, 70000000.788778M, 800000000.899889M, 9000000000.911991M };

return a;

}

}

}

UsingIndexForAccessingArrayElement.cs:

﻿using System;

namespace WorkingWithArrays

{

public static class UsingIndexerForAccessingArrayElement

{

public static int GetFirstArrayElement(int[] array)

{

return array[0];

}

public static int GetSecondArrayElement(int[] array)

{

return array[1];

}

public static int GetThirdArrayElement(int[] array)

{

return array[2];

}

public static int GetLastElement(int[] array)

{

return array[array.Length - 1];

}

public static int GetNextToLastElement(int[] array)

{

return array[array.Length - 2];

}

public static int GetNthArrayElement(int[] array, int n)

{

return array[n - 1];

}

public static bool GetFirstArrayElement(bool[] array)

{

return array[0];

}

public static bool GetSecondArrayElement(bool[] array)

{

return array[1];

}

public static bool GetSixthArrayElement(bool[] array)

{

return array[5];

}

public static bool GetLastElement(bool[] array)

{

return array[array.Length - 1];

}

public static bool GetNextToLastElement(bool[] array)

{

return array[array.Length - 2];

}

public static bool GetNthArrayElement(bool[] array, int n)

{

return array[n - 1];

}

public static string GetFirstArrayElement(string[] array)

{

return array[0];

}

public static string GetForthArrayElement(string[] array)

{

return array[3];

}

public static string GetLastElement(string[] array)

{

return array[array.Length - 1];

}

public static string GetNextToLastElement(string[] array)

{

return array[array.Length - 2];

}

public static char GetFirstArrayElement(char[] array)

{

return array[0];

}

public static char GetSeventhArrayElement(char[] array)

{

return array[6];

}

public static char GetLastElement(char[] array)

{

return array[array.Length - 1];

}

public static char GetNextToLastElement(char[] array)

{

return array[array.Length - 2];

}

public static double GetFirstArrayElement(double[] array)

{

return array[0];

}

public static double GetSeventhArrayElement(double[] array)

{

return array[6];

}

public static double GetLastElement(double[] array)

{

return array[array.Length - 1];

}

public static double GetNextToLastElement(double[] array)

{

return array[array.Length - 2];

}

public static float GetFirstArrayElement(float[] array)

{

return array[0];

}

public static float GetNinthArrayElement(float[] array)

{

return array[8];

}

public static float GetLastElement(float[] array)

{

return array[array.Length - 1];

}

public static float GetNextToLastElement(float[] array)

{

return array[array.Length - 2];

}

public static decimal GetLastElement(decimal[] array)

{

return array[array.Length - 1];

}

public static decimal GetNextToLastElement(decimal[] array)

{

return array[array.Length - 2];

}

public static decimal GetThirdElementFromEnd(decimal[] array)

{

return array[array.Length - 3];

}

public static decimal GetFourthElementFromEnd(decimal[] array)

{

return array[array.Length - 4];

}

}

}

UsingRanges.cs:

﻿using System;

namespace WorkingWithArrays

{

public static class UsingRanges

{

public static int[] GetArrayWithAllElements(int[] array)

{

var a = array[..];

return a;

}

public static int[] GetArrayWithoutFirstElement(int[] array)

{

var a = array[1..^0];

return a;

}

public static int[] GetArrayWithoutTwoFirstElements(int[] array)

{

var a = array[2..^0];

return a;

}

public static int[] GetArrayWithoutThreeFirstElements(int[] array)

{

var a = array[3..^0];

return a;

}

public static int[] GetArrayWithoutLastElement(int[] array)

{

var a = array[0..^1];

return a;

}

public static int[] GetArrayWithoutTwoLastElements(int[] array)

{

var a = array[0..^2];

return a;

}

public static int[] GetArrayWithoutThreeLastElements(int[] array)

{

var a = array[0..^3];

return a;

}

public static bool[] GetArrayWithoutFirstAndLastElements(bool[] array)

{

var a = array[1..^1];

return a;

}

public static bool[] GetArrayWithoutTwoFirstAndTwoLastElements(bool[] array)

{

var a = array[2..^2];

return a;

}

public static bool[] GetArrayWithoutThreeFirstAndThreeLastElements(bool[] array)

{

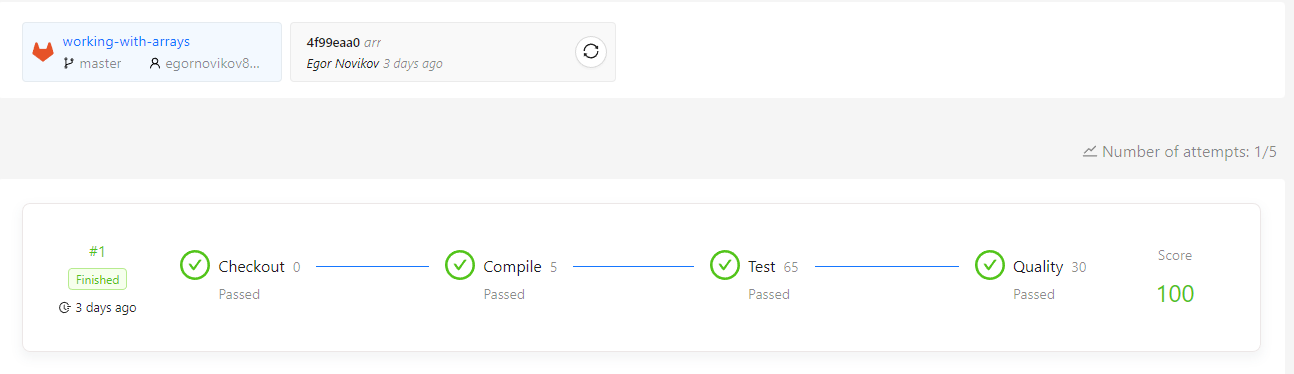
var a = array[3..^3];

return a;

}

}

}

Результаты теста:

Задача №19

Название: Counting array elements

Условие: Решить задачи на поиск количества определённых элементов массива

Код:

﻿DoWhileMethods.cs:

using System;

namespace CountingArrayElements

{

public static class DoWhileMethods

{

/// <summary>

/// Searches an array of booleans for false elements, and returns the number of occurrences of false values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of false values.</returns>

public static int GetFalseValueCount(bool[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int i = 0;

int result = 0;

do

{

if (!arrayToSearch[i])

{

result++;

}

i++;

}

while (i < arrayToSearch.Length);

return result;

}

/// <summary>

/// Searches an array of decimal floating-point numbers for zero elements, and returns the number of occurrences of zero values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of zero values.</returns>

public static int GetZeroDecimalCount(decimal[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int i = 0;

int result = 0;

do

{

if (arrayToSearch[i] == 0)

{

result++;

}

i++;

}

while (i < arrayToSearch.Length);

return result;

}

/// <summary>

/// Searches an array of double-precision floating-point numbers for elements that can be rounded to even, and returns the number of occurrences of numbers that can be rounded to even.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of numbers that can be rounded to even.</returns>

public static int GetRoundedToEvenCount(double[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int i = 0;

int result = 0;

do

{

if (Math.Round(arrayToSearch[i]) % 2 == 0)

{

result++;

}

i++;

}

while (i < arrayToSearch.Length);

return result;

}

/// <summary>

/// Searches an array of booleans for false elements, and returns the number of occurrences of false values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of false values.</returns>

public static int GetFalseValueCountRecursive(bool[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

return GetFalseValueCountRecursive(arrayToSearch, arrayToSearch.Length, 0);

}

/// <summary>

/// Searches an array of decimal floating-point numbers for zero elements, and returns the number of occurrences of zero values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of zero values.</returns>

public static int GetZeroDecimalCountRecursive(decimal[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return 0;

}

int middleIndex = arrayToSearch.Length / 2;

decimal[] leftArrayToSearch = arrayToSearch[..middleIndex];

decimal[] rightArrayToSearch = arrayToSearch[middleIndex..];

int leftArrayCount = GetZeroDecimalCountRecursive(leftArrayToSearch, 0);

int rightArrayCount = GetZeroDecimalCountRecursive(rightArrayToSearch, 0);

return leftArrayCount + rightArrayCount;

}

/// <summary>

/// Searches an array of double-precision floating-point numbers for elements that can be rounded to even, and returns the number of occurrences of numbers that can be rounded to even.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of numbers that can be rounded to even.</returns>

public static int GetRoundedToEvenCountRecursive(double[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

static int ProcessArray(double[] array)

{

if (array.Length <= 0)

{

return 0;

}

double currentElement = array[0];

int currentIncrement = 0;

bool isElementNearEvenNumber = (Math.Round(currentElement, MidpointRounding.ToEven) % 2) == 0;

if (isElementNearEvenNumber)

{

currentIncrement++;

}

if (array.Length > 1)

{

return GetRoundedToEvenCountRecursive(array[1..]) + currentIncrement;

}

return currentIncrement;

}

static int GetRoundedToEvenCountRecursive(double[] array)

{

if (array.Length == 0)

{

return 0;

}

int middleIndex = array.Length / 2;

double[] leftArrayToSearch = array[..middleIndex];

double[] rightArrayToSearch = array[middleIndex..];

return ProcessArray(leftArrayToSearch) + ProcessArray(rightArrayToSearch);

}

return GetRoundedToEvenCountRecursive(arrayToSearch);

}

private static int GetFalseValueCountRecursive(bool[] arrayToSearch, int elementsLeft, int accumulator)

{

if (elementsLeft > 0)

{

accumulator = !arrayToSearch[^elementsLeft] ? ++accumulator : accumulator;

return GetFalseValueCountRecursive(arrayToSearch, --elementsLeft, accumulator);

}

return accumulator;

}

private static int GetZeroDecimalCountRecursive(decimal[] arrayToSearch, int accumulator)

{

if (arrayToSearch.Length == 0)

{

return accumulator;

}

if (arrayToSearch[0] == decimal.Zero)

{

accumulator++;

}

return GetZeroDecimalCountRecursive(arrayToSearch[1..], accumulator);

}

}

}

ForMethods.cs:

﻿using System;

using System.Reflection;

namespace CountingArrayElements

{

public static class ForMethods

{

/// <summary>

/// Searches an array of integers for negative elements, and returns the number of occurrences of negative integers.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of negative integers.</returns>

public static int GetNegativeIntegerCount(int[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int result = 0;

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

{

if (arrayToSearch[i] < 0)

{

result++;

}

}

return result;

}

/// <summary>

/// Searches an array of single-precision floating-point numbers for even elements, and returns the number of occurrences of even numbers.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of even numbers.</returns>

public static int GetEvenNumberCount(float[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int result = 0;

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

{

if (arrayToSearch[i] % 2 == 0)

{

result++;

}

}

return result;

}

/// <summary>

/// Searches an array of bytes for elements with enabled bits in high nibble, and returns the number of occurrences of bytes with enabled bits in high nibble.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of bytes with enabled bits in high nibble.</returns>

public static int GetByteWithBitsInHighNibbleCount(byte[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int currentAccumulator = 0;

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

{

currentAccumulator = (arrayToSearch[i] & 0xF0) > 0 ? currentAccumulator + 1 : currentAccumulator;

}

return currentAccumulator;

}

/// <summary>

/// Searches an array of integers for negative elements, and returns the number of occurrences of negative integers.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of negative integers.</returns>

public static int GetNegativeIntegerCountRecursive(int[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

return GetNegativeIntegerCountRecursive(arrayToSearch, 0);

}

/// <summary>

/// Searches an array of single-precision floating-point numbers for even elements, and returns the number of occurrences of even numbers.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of even numbers.</returns>

public static int GetEvenNumberCountRecursive(float[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return 0;

}

int currentIncrement = arrayToSearch[0] % 2 == 0 ? 1 : 0;

return GetEvenNumberCountRecursive(arrayToSearch[1..]) + currentIncrement;

}

/// <summary>

/// Searches an array of bytes for elements with enabled bits in high nibble, and returns the number of occurrences of bytes with enabled bits in high nibble.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of bytes with enabled bits in high nibble.</returns>

public static int GetByteWithBitsInHighNibbleCountRecursive(byte[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

return GetByteWithBitsInHighNibbleCountRecursive(arrayToSearch, arrayToSearch.Length - 1, 0);

}

private static int GetNegativeIntegerCountRecursive(int[] arrayToSearch, int index)

{

if (index >= arrayToSearch.Length)

{

return 0;

}

int currentIncrement = arrayToSearch[index] < 0 ? 1 : 0;

return GetNegativeIntegerCountRecursive(arrayToSearch, index + 1) + currentIncrement;

}

private static int GetByteWithBitsInHighNibbleCountRecursive(byte[] arrayToSearch, int index, int accumulator)

{

if (index < 0)

{

return accumulator;

}

int currentAccumulator = (arrayToSearch[index] & 0xF0) > 0 ? accumulator + 1 : accumulator;

return GetByteWithBitsInHighNibbleCountRecursive(arrayToSearch, index - 1, currentAccumulator);

}

}

}

WhileMethods.cs:

﻿using System;

namespace CountingArrayElements

{

public static class WhileMethods

{

/// <summary>

/// Searches an array of strings for empty elements, and returns the number of occurrences of empty strings.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of empty strings.</returns>

public static int GetEmptyStringCount(string[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int result = 0;

int i = 0;

while (i < arrayToSearch.Length)

{

if (string.IsNullOrEmpty(arrayToSearch[i]))

{

result++;

}

i++;

}

return result;

}

/// <summary>

/// Searches an array of long integers for elements with minimum and maximum values, and returns the number of occurrences of long integers with minimum and maximum values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of long integers with minimum and maximum values.</returns>

public static int GetMinOrMaxLongCount(long[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int result = 0;

int i = 0;

while (i < arrayToSearch.Length)

{

if (arrayToSearch[i] == long.MinValue || arrayToSearch[i] == long.MaxValue)

{

result++;

}

i++;

}

return result;

}

/// <summary>

/// Searches an array of objects for null elements, and returns the number of occurrences of null values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of null values.</returns>

public static int GetNullObjectCount(object[] arrayToSearch)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int result = 0;

int i = 0;

while (i < arrayToSearch.Length)

{

if (arrayToSearch[i] == null)

{

result++;

}

i++;

}

return result;

}

/// <summary>

/// Searches an array of strings for empty elements, and returns the number of occurrences of empty strings.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of empty strings.</returns>

public static int GetEmptyStringCountRecursive(string[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return 0;

}

int currentIncrement = string.IsNullOrEmpty(arrayToSearch[^1]) ? 1 : 0;

string[] newArrayToSearch = arrayToSearch[..^1];

return GetEmptyStringCountRecursive(newArrayToSearch) + currentIncrement;

}

/// <summary>

/// Searches an array of long integers for elements with minimum and maximum values, and returns the number of occurrences of long integers with minimum and maximum values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of long integers with minimum and maximum values.</returns>

public static int GetMinOrMaxLongCountRecursive(long[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return 0;

}

long currentElement = arrayToSearch[0];

int currentIncrement = (currentElement == long.MinValue || currentElement == long.MaxValue) ? 1 : 0;

long[] newArrayToSearch = arrayToSearch[1..];

return GetMinOrMaxLongCountRecursive(newArrayToSearch) + currentIncrement;

}

/// <summary>

/// Searches an array of objects for null elements, and returns the number of occurrences of null values.

/// </summary>

/// <param name="arrayToSearch">An <see cref="Array"/> to search.</param>

/// <returns>The number of occurrences of null values.</returns>

public static int GetNullObjectCountRecursive(object[] arrayToSearch)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

return GetNullObjectCountRecursive(arrayToSearch, 0);

}

private static int GetNullObjectCountRecursive(object[] arrayToSearch, int accumulator)

{

if (arrayToSearch.Length == 0)

{

return accumulator;

}

if (arrayToSearch[0] is null)

{

accumulator++;

}

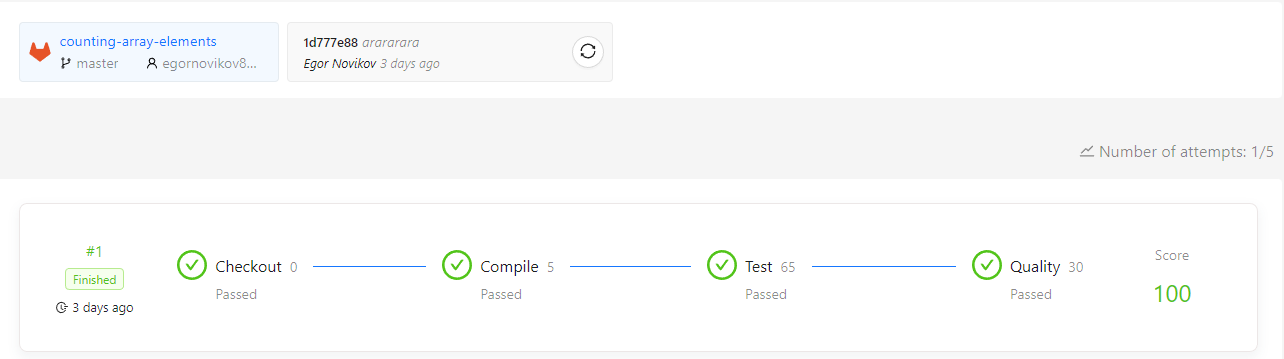
object[] newArrayToSearch = arrayToSearch[1..];

return GetNullObjectCountRecursive(newArrayToSearch, accumulator);

}

}

}

Результаты теста:

Задача №20

Название: Getting an array element index

Условие: Решить задачи на поиск индекса элемента массива по значению

Код:

DoWhileMethods.cs:

﻿using System;

namespace GettingArrayElementIndex

{

public static class DoWhileMethods

{

public static int GetIndexOf(this ulong[] arrayToSearch, ulong value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return -1;

}

int i = 0;

do

{

if (arrayToSearch[i] == value)

{

return i;

}

i++;

}

while (i < arrayToSearch.Length);

return -1;

}

public static int GetIndexOf(this ulong[] arrayToSearch, ulong value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (arrayToSearch.Length == 0)

{

return -1;

}

int i = startIndex;

do

{

if (arrayToSearch[i] == value)

{

return i;

}

i++;

}

while (i < startIndex + count);

return -1;

}

public static int GetLastIndexOf(this ulong[] arrayToSearch, ulong value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int i = arrayToSearch.Length - 1;

if (i < 0)

{

return -1;

}

do

{

if (arrayToSearch[i] == value)

{

return i;

}

}

while (--i >= 0);

return -1;

}

public static int GetLastIndexOf(this ulong[] arrayToSearch, ulong value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

int lastIndex = startIndex + count;

if (lastIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > arrayToSearch.Length");

}

int i = lastIndex - 1;

if (i < 0)

{

return -1;

}

do

{

if (arrayToSearch[i] == value)

{

return i;

}

}

while (--i >= startIndex);

return -1;

}

}

}

ForMethods.cs:

using System;

namespace GettingArrayElementIndex

{

public static class ForMethods

{

public static int GetIndexOf(this uint[] arrayToSearch, uint value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return -1;

}

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

{

if (arrayToSearch[i] == value)

{

return i;

}

}

return -1;

}

public static int GetIndexOf(this uint[] arrayToSearch, uint value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (arrayToSearch.Length == 0)

{

return -1;

}

for (int i = startIndex; i < startIndex + count; i++)

{

if (arrayToSearch[i] == value)

{

return i;

}

}

return -1;

}

public static int GetLastIndexOf(this uint[] arrayToSearch, uint value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

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

{

if (arrayToSearch[i] == value)

{

return i;

}

}

return -1;

}

public static int GetLastIndexOf(this uint[] arrayToSearch, uint value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > arrayToSearch.Length");

}

for (int i = startIndex + count - 1; i >= startIndex; i--)

{

if (arrayToSearch[i] == value)

{

return i;

}

}

return -1;

}

}

}

WhileMethods.cs:

using System;

namespace GettingArrayElementIndex

{

public static class WhileMethods

{

public static int GetIndexOf(this ushort[] arrayToSearch, ushort value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (arrayToSearch.Length == 0)

{

return -1;

}

int i = 0;

do

{

if (arrayToSearch[i] == value)

{

return i;

}

i++;

}

while (i < arrayToSearch.Length);

return -1;

}

public static int GetIndexOf(this ushort[] arrayToSearch, ushort value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (arrayToSearch.Length == 0)

{

return -1;

}

int i = startIndex;

while (i < startIndex + count)

{

if (arrayToSearch[i] == value)

{

return i;

}

i++;

}

return -1;

}

public static int GetLastIndexOf(this ushort[] arrayToSearch, ushort value)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

int i = arrayToSearch.Length - 1;

while (i >= 0)

{

if (arrayToSearch[i] == value)

{

return i;

}

i--;

}

return -1;

}

public static int GetLastIndexOf(this ushort[] arrayToSearch, ushort value, int startIndex, int count)

{

if (arrayToSearch is null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than arrayToSearch.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

int lastPosition = startIndex + count;

if (lastPosition > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > arrayToSearch.Length");

}

int i = lastPosition - 1;

while (i >= startIndex)

{

if (arrayToSearch[i] == value)

{

return i;

}

i--;

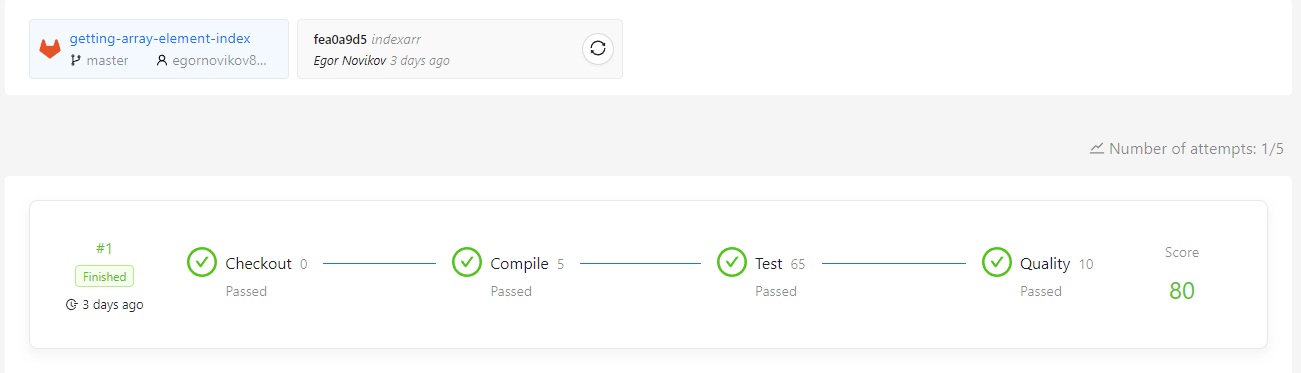
}

return -1;

}

}

}

﻿ Результаты теста:

Задача №21

Название: Looking for an array element

Условие: Решить задачи на поиск количества определённых элементов массива

Код:

IntegersCounter.cs:

﻿using System;

namespace LookingForArrayElements

{

public static class IntegersCounter

{

/// <summary>

/// Searches an array of integers for elements that are in <paramref name="elementsToSearchFor"/> <see cref="Array"/>, and returns the number of occurrences of the elements.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of integers to search.</param>

/// <param name="elementsToSearchFor">One-dimensional, zero-based <see cref="Array"/> that contains integers to search for.</param>

/// <returns>The number of occurrences of the elements that are in <paramref name="elementsToSearchFor"/> <see cref="Array"/>.</returns>

public static int GetIntegersCount(int[] arrayToSearch, int[] elementsToSearchFor)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (elementsToSearchFor == null)

{

throw new ArgumentNullException(nameof(elementsToSearchFor));

}

int result = 0;

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

{

for (int j = 0; j < elementsToSearchFor.Length; j++)

{

if (arrayToSearch[i] == elementsToSearchFor[j])

{

result++;

}

}

}

return result;

}

/// <summary>

/// Searches an array of integers for elements that are in <paramref name="elementsToSearchFor"/> <see cref="Array"/>, and returns the number of occurrences of the elements withing the range of elements in the <see cref="Array"/> that starts at the specified index and contains the specified number of elements.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of integers to search.</param>

/// <param name="elementsToSearchFor">One-dimensional, zero-based <see cref="Array"/> that contains integers to search for.</param>

/// <param name="startIndex">The zero-based starting index of the search.</param>

/// <param name="count">The number of elements in the section to search.</param>

/// <returns>The number of occurrences of the elements that are in <paramref name="elementsToSearchFor"/> <see cref="Array"/>.</returns>

public static int GetIntegersCount(int[] arrayToSearch, int[] elementsToSearchFor, int startIndex, int count)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (elementsToSearchFor == null)

{

throw new ArgumentNullException(nameof(elementsToSearchFor));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count));

}

int result = 0;

for (int i = startIndex; i < startIndex + count; i++)

{

for (int j = 0; j < elementsToSearchFor.Length; j++)

{

if (arrayToSearch[i] == elementsToSearchFor[j])

{

result++;

}

}

}

return result;

}

}

}

FloatCounter.cs:

﻿using System;

namespace LookingForArrayElements

{

public static class FloatCounter

{

/// <summary>

/// Searches an array of floats for elements that are in a specified range, and returns the number of occurrences of the elements that matches the range criteria.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of single-precision floating-point numbers.</param>

/// <param name="rangeStart">One-dimensional, zero-based <see cref="Array"/> of the range starts.</param>

/// <param name="rangeEnd">One-dimensional, zero-based <see cref="Array"/> of the range ends.</param>

/// <returns>The number of occurrences of the <see cref="Array"/> elements that match the range criteria.</returns>

public static int GetFloatsCount(float[] arrayToSearch, float[] rangeStart, float[] rangeEnd)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (rangeEnd == null)

{

throw new ArgumentNullException(nameof(rangeEnd));

}

if (rangeStart == null)

{

throw new ArgumentNullException(nameof(rangeStart));

}

if (rangeStart.Length != rangeEnd.Length)

{

throw new ArgumentException("lol");

}

int result = 0;

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

{

if (rangeStart[i] > rangeEnd[i])

{

throw new ArgumentException("lol");

}

float firstElem = rangeStart[i];

float lastElem = rangeEnd[i];

for (int j = 0; j < arrayToSearch.Length; j++)

{

if (arrayToSearch[j] >= firstElem && arrayToSearch[j] <= lastElem)

{

result++;

}

}

}

return result;

}

/// <summary>

/// Searches an array of floats for elements that are in a specified range, and returns the number of occurrences of the elements that matches the range criteria.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of single-precision floating-point numbers.</param>

/// <param name="rangeStart">One-dimensional, zero-based <see cref="Array"/> of the range starts.</param>

/// <param name="rangeEnd">One-dimensional, zero-based <see cref="Array"/> of the range ends.</param>

/// <param name="startIndex">The zero-based starting index of the search.</param>

/// <param name="count">The number of elements in the section to search.</param>

/// <returns>The number of occurrences of the <see cref="Array"/> elements that match the range criteria.</returns>

public static int GetFloatsCount(float[] arrayToSearch, float[] rangeStart, float[] rangeEnd, int startIndex, int count)

{

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (rangeEnd == null)

{

throw new ArgumentNullException(nameof(rangeEnd));

}

if (rangeStart == null)

{

throw new ArgumentNullException(nameof(rangeStart));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (startIndex > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count));

}

if (rangeStart.Length != rangeEnd.Length)

{

throw new ArgumentException("lol");

}

int result = 0;

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

{

if (rangeStart[i] > rangeEnd[i])

{

throw new ArgumentException("lol");

}

float firstElem = rangeStart[i];

float lastElem = rangeEnd[i];

for (int j = startIndex; j < startIndex + count; j++)

{

if (arrayToSearch[j] >= firstElem && arrayToSearch[j] <= lastElem)

{

result++;

}

}

}

return result;

}

}

}

DecimalCounter.cs:

﻿using System;

#pragma warning disable S2368

namespace LookingForArrayElements

{

public static class DecimalCounter

{

/// <summary>

/// Searches an array of decimals for elements that are in a specified range, and returns the number of occurrences of the elements that matches the range criteria.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of single-precision floating-point numbers.</param>

/// <param name="ranges">One-dimensional, zero-based <see cref="Array"/> of range arrays.</param>

/// <returns>The number of occurrences of the <see cref="Array"/> elements that match the range criteria.</returns>

public static int GetDecimalsCount(decimal[] arrayToSearch, decimal[][] ranges)

{

// #5. Implement the method using "do..while" statements.

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (ranges == null)

{

throw new ArgumentNullException(nameof(ranges));

}

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

{

decimal[] range = ranges[i] ?? throw new ArgumentNullException(nameof(ranges));

if (range.Length == 0)

{

return 0;

}

if (range.Length != 2)

{

throw new ArgumentException("Method throws ArgumentException in case the length of one of the ranges is less or greater than 2.", nameof(ranges));

}

if (range[0] > range[1])

{

throw new ArgumentException("abc", nameof(ranges));

}

}

int cnt = 0;

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

{

for (int j = 0; j < ranges.Length; j++)

{

decimal[] range = ranges[j] ?? throw new ArgumentNullException(nameof(ranges));

if (arrayToSearch[i] >= range[0] && arrayToSearch[i] <= range[1])

{

cnt++;

}

}

}

return cnt;

}

/// <summary>

/// Searches an array of decimals for elements that are in a specified range, and returns the number of occurrences of the elements that matches the range criteria.

/// </summary>

/// <param name="arrayToSearch">One-dimensional, zero-based <see cref="Array"/> of single-precision floating-point numbers.</param>

/// <param name="ranges">One-dimensional, zero-based <see cref="Array"/> of range arrays.</param>

/// <param name="startIndex">The zero-based starting index of the search.</param>

/// <param name="count">The number of elements in the section to search.</param>

/// <returns>The number of occurrences of the <see cref="Array"/> elements that match the range criteria.</returns>

public static int GetDecimalsCount(decimal[] arrayToSearch, decimal[][] ranges, int startIndex, int count)

{

// #6. Implement the method using "for" statement.

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (arrayToSearch == null)

{

throw new ArgumentNullException(nameof(arrayToSearch));

}

if (ranges == null)

{

throw new ArgumentNullException(nameof(ranges));

}

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

{

decimal[] range = ranges[i] ?? throw new ArgumentNullException(nameof(ranges));

if (range.Length == 0)

{

return 0;

}

if (range.Length != 2)

{

throw new ArgumentException("Method throws ArgumentException in case the length of one of the ranges is less or greater than 2.", nameof(ranges));

}

if (range[0] > range[1])

{

throw new ArgumentException("abc", nameof(ranges));

}

}

if (startIndex + count > arrayToSearch.Length)

{

throw new ArgumentOutOfRangeException(nameof(count));

}

int cnt = 0;

for (int i = startIndex; i < startIndex + count; i++)

{

for (int j = 0; j < ranges.Length; j++)

{

decimal[] range = ranges[j] ?? throw new ArgumentNullException(nameof(ranges));

if (arrayToSearch[i] >= range[0] && arrayToSearch[i] <= range[1])

{

cnt++;

}

}

}

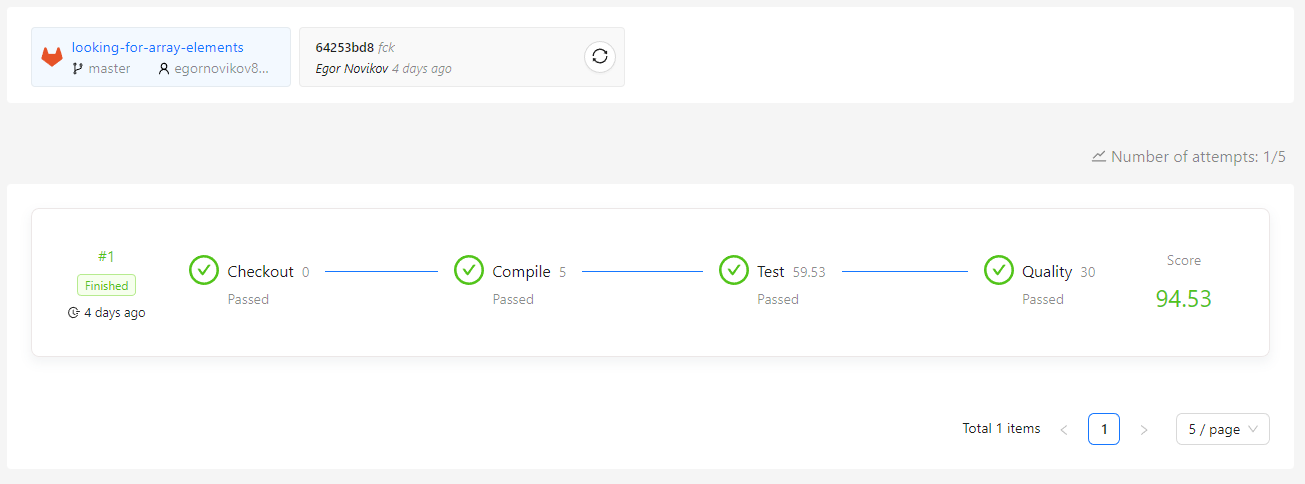
return cnt;

}

}

}

﻿ Результаты теста:



Задача №22

Название: Shift array element

Условие: Решить задачу на перестановки элементов массива

Код:

Shifter.cs:

﻿using System;

namespace ShiftArrayElements

{

public static class Shifter

{

/// <summary>

/// Shifts elements in a <see cref="source"/> array using <see cref="iterations"/> array for getting iterations and iterations (see README.md for detailed instructions).

/// </summary>

/// <param name="source">A source array.</param>

/// <param name="iterations">An array with iterations.</param>

/// <returns>An array with shifted elements.</returns>

/// <exception cref="ArgumentNullException">source array is null.</exception>

/// <exception cref="ArgumentNullException">iterations array is null.</exception>

public static int[] Shift(int[] source, int[] iterations)

{

if (source == null)

{

throw new ArgumentNullException(nameof(source));

}

if (iterations == null)

{

throw new ArgumentNullException(nameof(iterations));

}

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

{

if (i % 2 == 0)

{

for (int k = 0; k < iterations[i]; k++)

{

int firstElem = source[0];

for (int j = 0; j < source.Length; j++)

{

if (j != source.Length - 1)

{

source[j] = source[j + 1];

}

else

{

source[j] = firstElem;

}

}

}

}

else if (i % 2 == 1)

{

for (int k = 0; k < iterations[i]; k++)

{

int lastElem = source[source.Length - 1];

for (int j = source.Length - 1; j >= 0; j--)

{

if (j != 0)

{

source[j] = source[j - 1];

}

else

{

source[j] = lastElem;

}

}

}

}

}

return source;

}

}

}

ENumShifter.cs:

﻿using System;

namespace ShiftArrayElements

{

public static class EnumShifter

{

/// <summary>

/// Shifts elements in a <see cref="source"/> array using directions from <see cref="directions"/> array, one element shift per each direction array element.

/// </summary>

/// <param name="source">A source array.</param>

/// <param name="directions">An array with directions.</param>

/// <returns>An array with shifted elements.</returns>

/// <exception cref="ArgumentNullException">source array is null.</exception>

/// <exception cref="ArgumentNullException">directions array is null.</exception>

/// <exception cref="InvalidOperationException">direction array contains an element that is not <see cref="Direction.Left"/> or <see cref="Direction.Right"/>.</exception>

public static int[] Shift(int[] source, Direction[] directions)

{

if (source == null)

{

throw new ArgumentNullException(nameof(source));

}

if (directions == null)

{

throw new ArgumentNullException(nameof(directions));

}

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

{

if (directions[i] != Direction.Left && directions[i] != Direction.Right)

{

throw new InvalidOperationException();

}

Direction currentDirection = directions[i];

switch (currentDirection)

{

case Direction.Left:

{

int firstElem = source[0];

for (int j = 0; j < source.Length; j++)

{

if (j != source.Length - 1)

{

source[j] = source[j + 1];

}

else

{

source[j] = firstElem;

}

}

break;

}

case Direction.Right:

{

int lastElem = source[source.Length - 1];

for (int j = source.Length - 1; j >= 0; j--)

{

if (j != 0)

{

source[j] = source[j - 1];

}

else

{

source[j] = lastElem;

}

}

break;

}

default:

throw new InvalidOperationException($"Incorrect {currentDirection} enum value.");

}

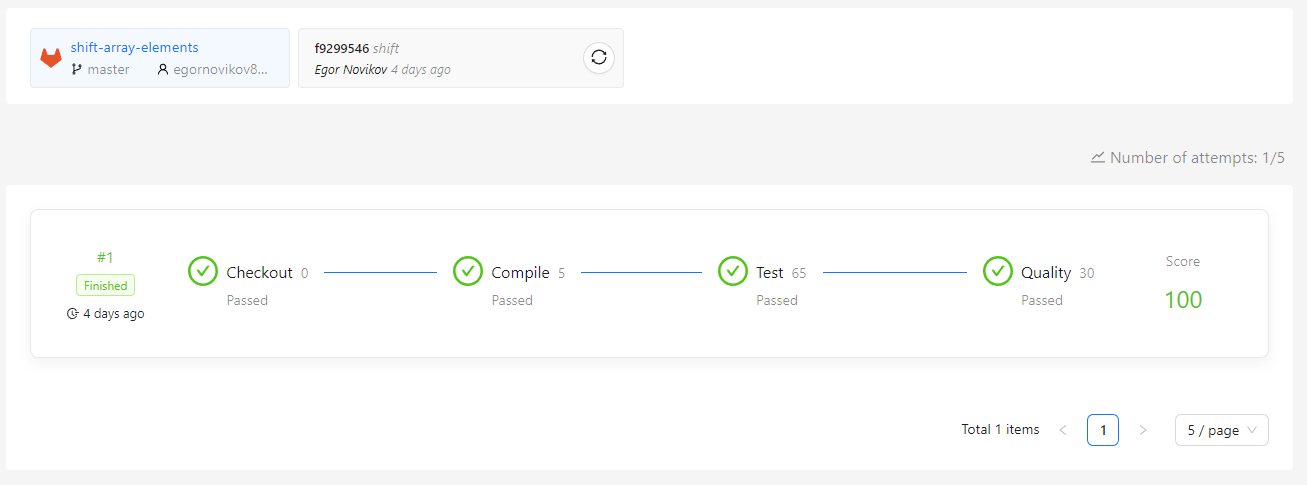
}

return source;

}

}

}

﻿ Результаты теста:

Задача №23

Название: Prime factors

Условие: Решить задачу на поиск простых множителей числа

Код:

﻿using System;

using System.Collections.Generic;

using System.Numerics;

namespace PrimeFactorsTask

{

public static class PrimeFactors

{

/// <summary>

/// Compute the prime factors of a given natural number.

/// A prime number is only evenly divisible by itself and 1.

/// Note that 1 is not a prime number.

/// </summary>

/// <param name="number">Source number.</param>

/// <returns>Prime factors of a given natural number.</returns>

/// <enumberception cref="ArgumentEnumberception">Thrown when number less or equal 0.</enumberception>

/// <enumberample>

/// 60 => {2, 2, 3, 5}

/// 8 => {2, 2, 2}

/// 12 => {2, 2, 3}

/// 901255 => {5, 17, 23, 461}

/// 93819012551 => {11, 9539, 894119}.

/// </enumberample>

public static int[] GetFactors(int number)

{

if (number <= 0)

{

throw new ArgumentException("lol");

}

List<int> factors = new List<int>();

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

{

while (number % i == 0)

{

factors.Add(i);

number /= i;

}

}

int[] result = new int[factors.Count];

for (int i = 0; i < factors.Count; i++)

{

result[i] = factors[i];

}

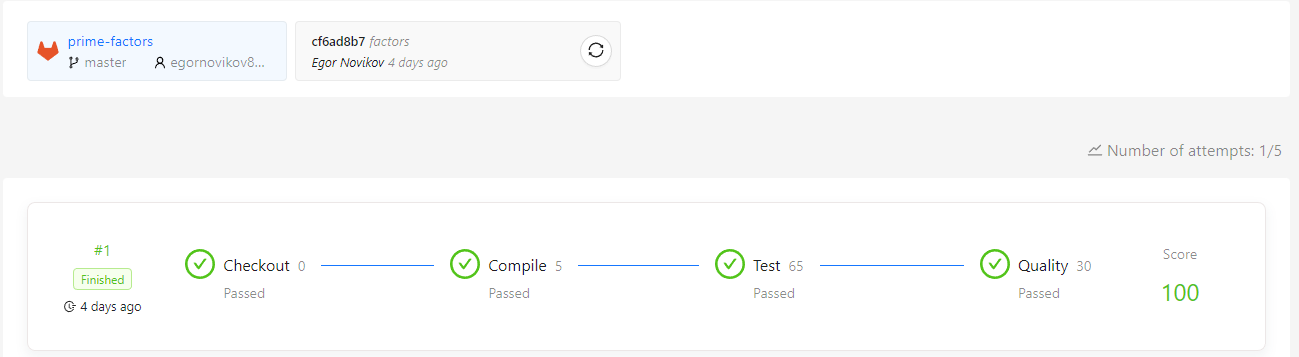
Array.Sort(result);

return result;

}

}

}

 Результаты теста:

Задача №24

Название: Get neighbors

Условие: Определить, является ли точка «соседом», т.е. попадает ли в ограниченную область и вернуть массив таких точек

Код:

﻿ ﻿using System;

using System.Collections.Generic;

namespace GetNeighbors

{

public static class CartesianCoordinates

{

/// <summary>

/// Gets from a set of points only points that are h-neighbors for a point with integer coordinates x and y.

/// </summary>

/// <param name="point">Given point with integer coordinates x and y.</param>

/// <param name="h">Distance around a given point.</param>

/// <param name="points">A given set of points.</param>

/// <returns>Only points that are h-neighbors for a point with integer coordinates x and y.</returns>

/// <exception cref="ArgumentNullException">Throw when array points is null.</exception>

/// <exception cref="ArgumentException">Throw when h-distance is less or equals zero.</exception>

public static Point[] GetNeighbors(Point point, int h, params Point[] points)

{

if (points == null)

{

throw new ArgumentNullException(nameof(points));

}

if (h <= 0)

{

throw new ArgumentException("lol");

}

List<Point> neighbours = new List<Point>();

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

{

if (points[i].X <= point.X + h && points[i].X >= point.X - h && points[i].Y <= point.Y + h && points[i].Y >= point.Y - h)

{

neighbours.Add(points[i]);

}

}

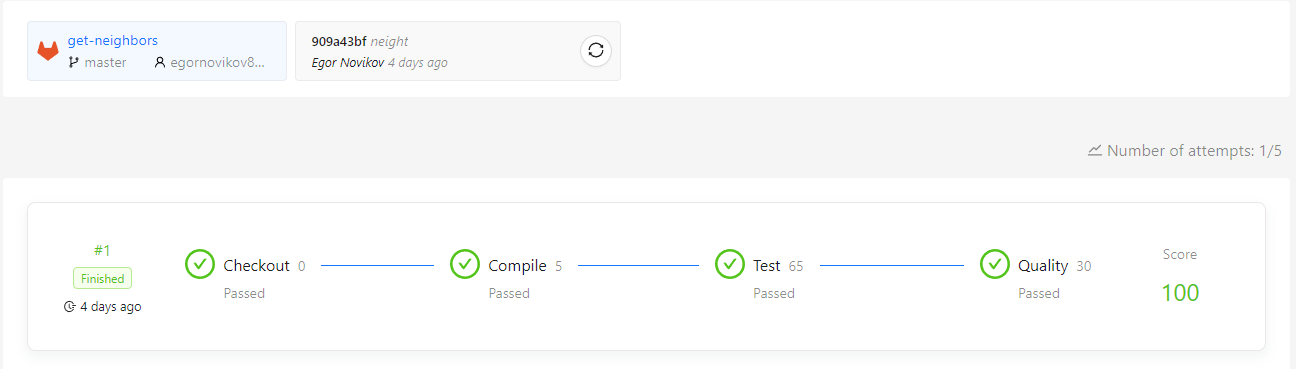
Point[] neigh = neighbours.ToArray();

return neigh;

}

}

}

﻿ Результаты теста:

Задача №25

Название: Working with strings

Условие: Научиться базовым операциям со строками

Код:

ConcatenatingString.cs:

﻿using System;

using System.Collections.Generic;

using Microsoft.VisualBasic;

namespace WorkingWithStrings

{

public static class ConcatenatingStrings

{

/// <summary>

/// Concatenates two specified instances of <see cref="string"/>.

/// </summary>

public static string ConcatenateStrings(string str1, string str2)

{

return string.Concat(str1, str2);

}

/// <summary>

/// Concatenates three specified instances of <see cref="string"/>.

/// </summary>

public static string ConcatenateStrings(string str1, string str2, string str3)

{

return string.Concat(str1, str2, str3);

}

/// <summary>

/// Concatenates four specified instances of <see cref="string"/>.

/// </summary>

public static string ConcatenateStrings(string str1, string str2, string str3, string str4)

{

return string.Concat(str1, str2, str3, str4);

}

/// <summary>

/// Concatenates the members of an <see cref="IEnumerable{T}"/> implementation.

/// </summary>

public static string ConcatenateStrings(IEnumerable<string> strings)

{

return string.Concat(strings);

}

/// <summary>

/// Concatenates the string representations of three specified objects.

/// </summary>

public static string ConcatenateValues(string str, int intValue, long longValue)

{

return string.Concat(str, intValue, longValue);

}

/// <summary>

/// Concatenates the string representations of four specified objects.

/// </summary>

public static string ConcatenateValues(short shortValue, float floatValue, bool boolValue, double doubleValue)

{

return string.Concat(shortValue, floatValue, boolValue, doubleValue);

}

/// <summary>

/// Concatenates the members of an <see cref="IEnumerable{T}"/> implementation.

/// </summary>

public static string ConcatenateValues(IEnumerable<object> values)

{

return string.Concat(values);

}

}

}

CopyingString.cs:

using System;

namespace WorkingWithStrings

{

public static class CopyingStrings

{

/// <summary>

/// Copies one character from the <paramref name="source"/> to the <paramref name="destination"/>.

/// </summary>

public static string CopyOneChar(string source, string destination)

{

char[] destinationArray = destination.ToCharArray();

source.CopyTo(0, destinationArray, 4, source.Length);

return new string(destinationArray);

}

/// <summary>

/// Copies three characters from the <paramref name="source"/> to the <paramref name="destination"/>.

/// </summary>

public static string CopyThreeChars(string source, string destination)

{

char[] destinationArray = destination.ToCharArray();

source.CopyTo(0, destinationArray, 0, 3);

return new string(destinationArray);

}

/// <summary>

/// Copies five characters from the <paramref name="source"/> to the <paramref name="destination"/>.

/// </summary>

public static string CopyFiveChars(string source, string destination)

{

char[] destinationArray = destination.ToCharArray();

source.CopyTo(0, destinationArray, 4, 5);

return new string(destinationArray);

}

/// <summary>

/// Copies six characters from the <paramref name="source"/> to the <paramref name="destination"/>.

/// </summary>

public static string CopySixChars(string source, string destination)

{

char[] destinationArray = destination.ToCharArray();

source.CopyTo(2, destinationArray, 5, 6);

return new string(destinationArray);

}

/// <summary>

/// Gets a production code by copying substrings of the <paramref name="regionCode"/>, <paramref name="locationCode"/>, <paramref name="dateCode"/> and <paramref name="factoryCode"/> parameters to the <paramref name="template"/>.

/// </summary>

public static string GetProductionCode(string template, string regionCode, string locationCode, string dateCode, string factoryCode)

{

char[] destinationArray = template.ToCharArray();

regionCode.CopyTo(1, destinationArray, 0, 1);

locationCode.CopyTo(4, destinationArray, 3, 2);

dateCode.CopyTo(3, destinationArray, 7, 3);

factoryCode.CopyTo(2, destinationArray, 12, 4);

return new string(destinationArray);

}

}

}

CreatingString.cs:

﻿using System;

using System.Drawing;

namespace WorkingWithStrings

{

public static class CreatingStrings

{

/// <summary>

/// Returns a new string that contains a specified string.

/// </summary>

public static string ReturnNewString(string str)

{

return new string(str);

}

/// <summary>

/// Returns a new string that contains a specified character repeated a specified number of times.

/// </summary>

public static string ReturnStringWithRepeatedChars(char c, int count)

{

return new string(c, count);

}

/// <summary>

/// Returns a new string that contains a specified character array.

/// </summary>

public static string ReturnStringFromCharArray(char[] c)

{

return new string(c);

}

/// <summary>

/// Returns a new string that contains a part of a specified character array.

/// </summary>

public static string ReturnStringFromCharArray(char[] c, int startIndex, int length)

{

return new string(c, startIndex, length);

}

}

}

JoiningStrings.cs:

﻿using System;

using System.Collections.Generic;

namespace WorkingWithStrings

{

public static class JoiningStrings

{

/// <summary>

/// Concatenates an array of strings, using the comma character between each member.

/// </summary>

public static string GetCommaSeparatedString(string[] values)

{

return string.Join(',', values);

}

/// <summary>

/// Concatenates an array of strings, using the colon character between each member.

/// </summary>

public static string GetColonSeparatedString(string[] values)

{

return string.Join(':', values);

}

/// <summary>

/// Concatenates an array of strings without the first array element, using the comma between each member.

/// </summary>

public static string GetCommaSeparatedStringWithoutFirstElement(string[] values)

{

return string.Join(',', values, 1, values.Length - 1);

}

/// <summary>

/// Concatenates an array of strings without the first and the last array elements, using the hyphen between each member.

/// </summary>

public static string GetHyphenSeparatedStringWithoutFirstAndLastElements(string[] values)

{

return string.Join('-', values, 1, values.Length - 2);

}

/// <summary>

/// Concatenates an array of strings, using the plus sign between each member.

/// </summary>

public static string GetPlusSeparatedString(IEnumerable<string> values)

{

return string.Join('+', values);

}

/// <summary>

/// Concatenates an array of strings, using the backslash sign between each member.

/// </summary>

public static string GetBackslashSeparatedString(IEnumerable<object> values)

{

return string.Join('\\', values);

}

/// <summary>

/// Concatenates an array of strings, using the separator string between each member.

/// </summary>

public static string GetStringSeparatedString(object[] values)

{

return string.Join("], [", values);

}

/// <summary>

/// Concatenates an array of strings, using the separator string between each member.

/// </summary>

public static string GetStringSeparatedStringForLastThreeElements(string separator, string[] values)

{

return string.Join(separator, values, values.Length - 3, 3);

}

}

}

SplittingStrings.cs:

using System;

namespace WorkingWithStrings

{

public static class SplittingStrings

{

/// <summary>

/// Splits a comma-separated string into substrings that are based on the comma character, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitCommaSeparatedString(string str)

{

return str.Split(',');

}

/// <summary>

/// Splits a colon-separated string into substrings that are based on the colon character, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitColonSeparatedString(string str)

{

return str.Split(':');

}

/// <summary>

/// Splits a comma-separated string into substrings that are based on the comma character, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitCommaSeparatedStringMaxTwoElements(string str)

{

return str.Split(',', 2);

}

/// <summary>

/// Splits a colon-separated string into substrings that are based on the colon character, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitColonSeparatedStringMaxThreeElements(string str)

{

return str.Split(':', 3);

}

/// <summary>

/// Splits a hyphen-separated string into substrings that are based on the hyphen character, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitHyphenSeparatedStringMaxThreeElementsRemoveEmptyStrings(string str)

{

return str.Split('-', 3, StringSplitOptions.RemoveEmptyEntries);

}

/// <summary>

/// Splits a separated string that is separated with colon and comma characters into substrings, and return an array whose elements contain the substrings.

/// </summary>

public static string[] SplitColonAndCommaSeparatedStringMaxFourElementsRemoveEmptyStrings(string str)

{

char[] separators = new char[] { ',', ':' };

return str.Split(separators, 4, StringSplitOptions.RemoveEmptyEntries);

}

/// <summary>

/// Splits a sentence into substrings, and return an array whose elements contain only words.

/// </summary>

public static string[] GetOnlyWords(string str)

{

char[] separators = new char[] { ' ', '.', ':', ',', '-', '\t', '!' };

string[] subs = str.Split(separators, StringSplitOptions.RemoveEmptyEntries);

return subs;

}

/// <summary>

/// Splits a CSV (comma-separated values) string into substrings that are based on the comma character, and return an array of the CSV line elements.

/// </summary>

public static string[] GetDataFromCsvLine(string str)

{

char[] separators = new char[] { ' ', ':', ',', '-', '\t', '!' };

string[] subs = str.Split(separators, StringSplitOptions.RemoveEmptyEntries);

return subs;

}

}

}

StringLiterals.cs:

using System;

// ReSharper disable StringLiteralTypo

// ReSharper disable CommentTypo

// ReSharper disable IdentifierTypo

namespace WorkingWithStrings

{

public static class StringLiterals

{

/// <summary>

/// Returns a "abc" string.

/// </summary>

public static string ReturnAbcStringLiteral()

{

return "abc";

}

/// <summary>

/// Returns a "ABCDEF" string.

/// </summary>

public static string ReturnAbcdefStringLiteral()

{

return "ABCDEF";

}

/// <summary>

/// Returns an empty string.

/// </summary>

public static string ReturnEmptyString()

{

return string.Empty;

}

/// <summary>

/// Returns a file path.

/// </summary>

public static string ReturnFilePathStringLiteral()

{

return "c:\\documents\\files\\myfile0234.txt";

}

/// <summary>

/// Returns a file path.

/// </summary>

public static string ReturnFilePathVerbatimStringLiteral()

{

return @"c:\documents\files\myfile0234.txt";

}

}

}

UsingIndexForAccessingStringChar.cs:

using System;

namespace WorkingWithStrings

{

public static class UsingIndexerForAccessingStringChar

{

/// <summary>

/// Gets the first character of the <paramref name="str"/> string.

/// </summary>

public static char GetFirstChar(string str)

{

return str[0];

}

/// <summary>

/// Gets the second character of the <paramref name="str"/> string.

/// </summary>

public static char GetSecondChar(string str)

{

return str[1];

}

/// <summary>

/// Gets the third character of the <paramref name="str"/> string.

/// </summary>

public static char GetThirdChar(string str)

{

return str[2];

}

/// <summary>

/// Gets the last character of the <paramref name="str"/> string.

/// </summary>

public static char GetLastChar(string str)

{

return str[^1];

}

/// <summary>

/// Gets the next to last character of the <paramref name="str"/> string.

/// </summary>

public static char GetNextToLastChar(string str)

{

return str[^2];

}

/// <summary>

/// Gets the third character from the end of the <paramref name="str"/> string.

/// </summary>

public static char GetThirdCharFromEnd(string str)

{

return str[^3];

}

/// <summary>

/// Gets the n-th character of the <paramref name="str"/> string.

/// </summary>

public static char GetNthChar(string str, int n)

{

return str[n - 1];

}

/// <summary>

/// Gets the n-th character from the end of the <paramref name="str"/> string.

/// </summary>

public static char GetNthCharFromEnd(string str, int n)

{

return str[^n];

}

/// <summary>

/// Gets the last character of the <paramref name="str"/> string.

/// </summary>

public static char GetLastCharUsingLength(string str)

{

return str[str.Length - 1];

}

/// <summary>

/// Gets the next to last character of the <paramref name="str"/> string.

/// </summary>

public static char GetNextToLastCharUsingLength(string str)

{

return str[str.Length - 2];

}

/// <summary>

/// Gets the fifth character from the end of the <paramref name="str"/> string.

/// </summary>

public static char GetFifthCharFromEndUsingLength(string str)

{

return str[str.Length - 5];

}

/// <summary>

/// Gets the n-th character of the <paramref name="str"/> string.

/// </summary>

public static char GetNthCharFromEndUsingLength(string str, int n)

{

return str[str.Length - n];

}

/// <summary>

/// Gets special code characters from <paramref name="serialNumber"/>.

/// </summary>

public static void GetSpecialCodes(string serialNumber, out char expectedCode1, out char expectedCode2, out char expectedCode3)

{

expectedCode1 = serialNumber[1];

expectedCode2 = serialNumber[^8];

expectedCode3 = serialNumber[^4];

}

}

}

UsingRanges.cs:

﻿using System;

namespace WorkingWithStrings

{

public static class UsingRanges

{

/// <summary>

/// Gets a string with all characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithAllChars(string str)

{

return str[..];

}

/// <summary>

/// Gets a string without first character of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutFirstChar(string str)

{

return str[1..];

}

/// <summary>

/// Gets a string without two first character of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutTwoFirstChars(string str)

{

return str[2..];

}

/// <summary>

/// Gets a string without three first character of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutThreeFirstChars(string str)

{

return str[3..];

}

/// <summary>

/// Gets a string without a last character of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutLastChar(string str)

{

return str[..^1];

}

/// <summary>

/// Gets a string without a two last characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutTwoLastChars(string str)

{

return str[..^2];

}

/// <summary>

/// Gets a string without a three last characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutThreeLastChars(string str)

{

return str[..^3];

}

/// <summary>

/// Gets a string without the first and the last characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutFirstAndLastChars(string str)

{

return str[1..^1];

}

/// <summary>

/// Gets a string without the first two and the last two characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutTwoFirstAndTwoLastChars(string str)

{

return str[2..^2];

}

/// <summary>

/// Gets a string without the first three and the last three characters of the <paramref name="str"/> string.

/// </summary>

public static string GetStringWithoutThreeFirstAndThreeLastChars(string str)

{

return str[3..^3];

}

/// <summary>

/// Gets details of the production code.

/// </summary>

public static void GetProductionCodeDetails(string productionCode, out string regionCode, out string locationCode, out string dateCode, out string factoryCode)

{

regionCode = productionCode[..1];

locationCode = productionCode[3..5];

dateCode = productionCode[^9..^6];

factoryCode = productionCode[^4..];

}

/// <summary>

/// Gets details of the production code.

/// </summary>

public static void GetSerialNumberDetails(string serialNumber, out string countryCode, out string manufacturerCode, out string factoryCode, out string stationCode)

{

countryCode = serialNumber[^9..^8];

manufacturerCode = serialNumber[^8..^6];

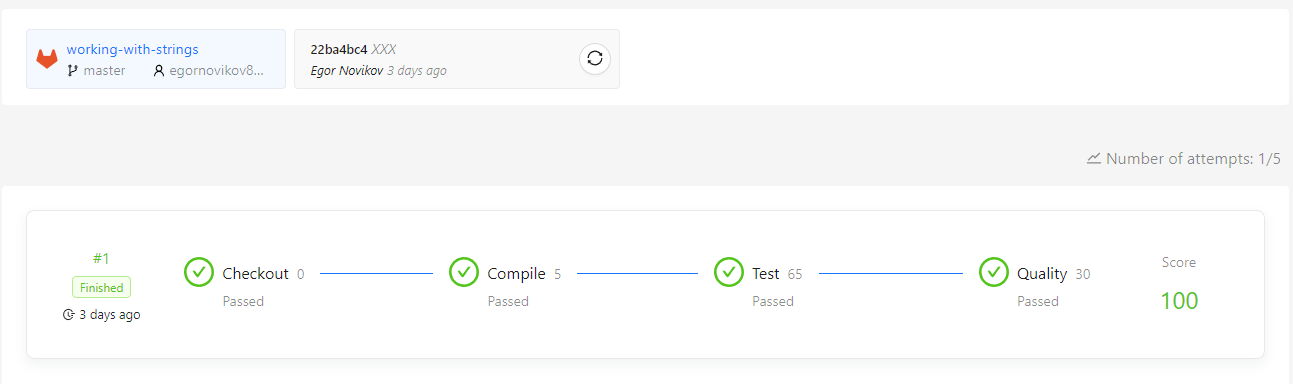
factoryCode = serialNumber[^5..^1];

stationCode = serialNumber[^1..];

}

}

}

﻿ Результаты теста:

Задача №26

Название: Getting a Character Index

Условие: Решить задачи на получение индекса символа в строке

Код:

DoWhileMethods.cs:

﻿using System;

using System.Drawing;

namespace GettingCharIndex

{

public static class DoWhileMethods

{

public static int GetIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return -1;

}

int i = 0;

do

{

if (str[i] == value)

{

return i;

}

i++;

}

while (i < str.Length);

return -1;

}

public static int GetIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

if (str.Length == 0)

{

return -1;

}

int i = startIndex;

do

{

if (str[i] == value)

{

return i;

}

i++;

}

while (i < startIndex + count);

return -1;

}

public static int GetLastIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

int currentCharIndex = str.Length - 1;

if (currentCharIndex < 0)

{

return -1;

}

do

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

currentCharIndex--;

}

while (currentCharIndex >= 0);

return -1;

}

public static int GetLastIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

int currentCharIndex = startIndex + count - 1;

if (currentCharIndex < 0)

{

return -1;

}

do

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

currentCharIndex--;

}

while (currentCharIndex >= startIndex);

return -1;

}

}

}

ForMethods.cs:

using System;

namespace GettingCharIndex

{

public static class ForMethods

{

public static int GetIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return -1;

}

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

{

if (str[i] == value)

{

return i;

}

}

return -1;

}

public static int GetIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

if (str.Length == 0)

{

return -1;

}

for (int i = startIndex; i < startIndex + count; i++)

{

if (str[i] == value)

{

return i;

}

}

return -1;

}

public static int GetLastIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

for (int currentCharIndex = str.Length - 1; currentCharIndex >= 0; currentCharIndex--)

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

}

return -1;

}

public static int GetLastIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

for (int currentCharIndex = startIndex + count - 1; currentCharIndex >= startIndex; currentCharIndex--)

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

}

return -1;

}

}

}

WhileMethods.cs:

﻿using System;

namespace GettingCharIndex

{

public static class WhileMethods

{

public static int GetIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return -1;

}

int i = 0;

while (i < str.Length)

{

if (str[i] == value)

{

return i;

}

i++;

}

return -1;

}

public static int GetIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

if (str.Length == 0)

{

return -1;

}

int i = startIndex;

while (i < startIndex + count)

{

if (str[i] == value)

{

return i;

}

i++;

}

return -1;

}

public static int GetLastIndexOfChar(string str, char value)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

int currentCharIndex = str.Length - 1;

while (currentCharIndex >= 0)

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

currentCharIndex--;

}

return -1;

}

public static int GetLastIndexOfChar(string str, char value, int startIndex, int count)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is less than zero");

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex), "startIndex is greater than str.Length");

}

if (count < 0)

{

throw new ArgumentOutOfRangeException(nameof(count), "count is less than zero");

}

if (startIndex + count > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(count), "startIndex + count > str.Length");

}

int currentCharIndex = startIndex + count - 1;

while (currentCharIndex >= startIndex)

{

char currentChar = str[currentCharIndex];

if (currentChar == value)

{

return currentCharIndex;

}

currentCharIndex--;

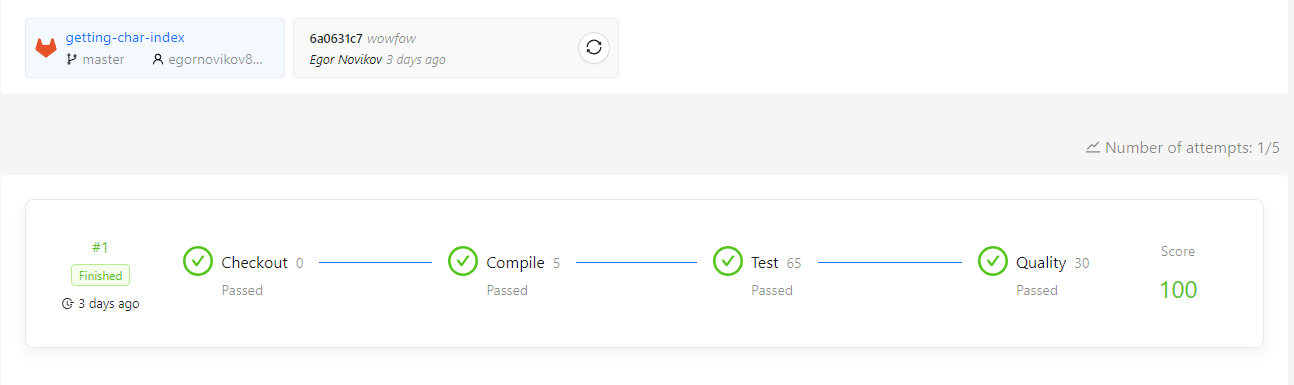
}

return -1;

}

}

}

﻿ Результаты теста:

Задача №27

Название: Counting String Chars

Условие: Решить задачи на получение количества определённых символов в строке

Код:

DoWhileMethods.cs:

﻿using System;

namespace CountingStringChars

{

public static class DoWhileMethods

{

/// <summary>

/// Returns a number of digits in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of digits in a string.</returns>

public static int GetDigitCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int i = 0;

int result = 0;

do

{

if (char.IsDigit(str[i]))

{

result++;

}

i++;

}

while (i < str.Length);

return result;

}

/// <summary>

/// Returns a number of letters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of letters in a string.</returns>

public static int GetLetterCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int i = 0;

int result = 0;

do

{

if (char.IsLetter(str[i]))

{

result++;

}

i++;

}

while (i < str.Length);

return result;

}

/// <summary>

/// Returns a number of digits in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of digits in a string.</returns>

public static int GetDigitCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

return GetDigitCountRecursive(str, str.Length, 0);

}

/// <summary>

/// Returns a number of letters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of letters in a string.</returns>

public static int GetLetterCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

return GetLetterCountRecursive(str, str.Length, 0);

}

private static int GetDigitCountRecursive(string str, int charsLeft, int counter)

{

if (charsLeft > 0)

{

return GetDigitCountRecursive(str, charsLeft - 1, char.IsDigit(str[^charsLeft]) ? counter + 1 : counter);

}

return counter;

}

private static int GetLetterCountRecursive(string str, int charsLeft, int counter)

{

if (charsLeft > 0)

{

return GetLetterCountRecursive(str, charsLeft - 1, char.IsLetter(str[^charsLeft]) ? counter + 1 : counter);

}

return counter;

}

}

}

ForMethods.cs:

﻿using System;

namespace CountingStringChars

{

public static class ForMethods

{

/// <summary>

/// Returns a number of characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of characters in a string.</returns>

public static int GetCharCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int result = 0;

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

{

if (char.IsLetter(str[i]))

{

result++;

}

}

return result;

}

/// <summary>

/// Returns a number of upper characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of upper characters in a string.</returns>

public static int GetUpperCharCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int result = 0;

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

{

if (char.IsUpper(str[i]))

{

result++;

}

}

return result;

}

/// <summary>

/// Returns a number of characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of characters in a string.</returns>

public static int GetCharCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

return GetCharCountRecursive(str, 0);

}

/// <summary>

/// Returns a number of upper characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of upper characters in a string.</returns>

public static int GetUpperCharCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

return GetUpperCharCountRecursive(str, 0);

}

private static int GetCharCountRecursive(string str, int index)

{

if (index >= str.Length)

{

return 0;

}

return GetCharCountRecursive(str, index + 1) + 1;

}

private static int GetUpperCharCountRecursive(string str, int index)

{

if (index >= str.Length)

{

return 0;

}

bool isUpper = char.IsUpper(str[index]);

int currentIncrement = isUpper ? 1 : 0;

return GetUpperCharCountRecursive(str, index + 1) + currentIncrement;

}

}

}

WhileMethods.cs:

﻿using System;

namespace CountingStringChars

{

public static class WhileMethods

{

/// <summary>

/// Returns a number of white space characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of white space characters in a string.</returns>

public static int GetSpaceCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int result = 0;

int i = 0;

while (i < str.Length)

{

if (str[i] == ' ')

{

result++;

}

i++;

}

return result;

}

/// <summary>

/// Returns a number of punctuation marks in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of punctuation marks in a string.</returns>

public static int GetPunctuationCount(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int i = 0;

int result = 0;

while (i < str.Length)

{

if (char.IsPunctuation(str[i]))

{

result++;

}

i++;

}

return result;

}

/// <summary>

/// Returns a number of white space characters in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of white space characters in a string.</returns>

public static int GetSpaceCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

int result = GetSpaceCountRecursive(str[1..]) + (char.IsWhiteSpace(str[0]) ? 1 : 0);

return result;

}

/// <summary>

/// Returns a number of punctuation marks in a string.

/// </summary>

/// <param name="str">A <see cref="string"/> to search.</param>

/// <returns>A number of punctuation marks in a string.</returns>

public static int GetPunctuationCountRecursive(string str)

{

if (str is null)

{

throw new ArgumentNullException(nameof(str));

}

if (string.IsNullOrEmpty(str))

{

return 0;

}

bool isPunctuation = char.IsPunctuation(str[0]);

int currentIncrement = isPunctuation ? 1 : 0;

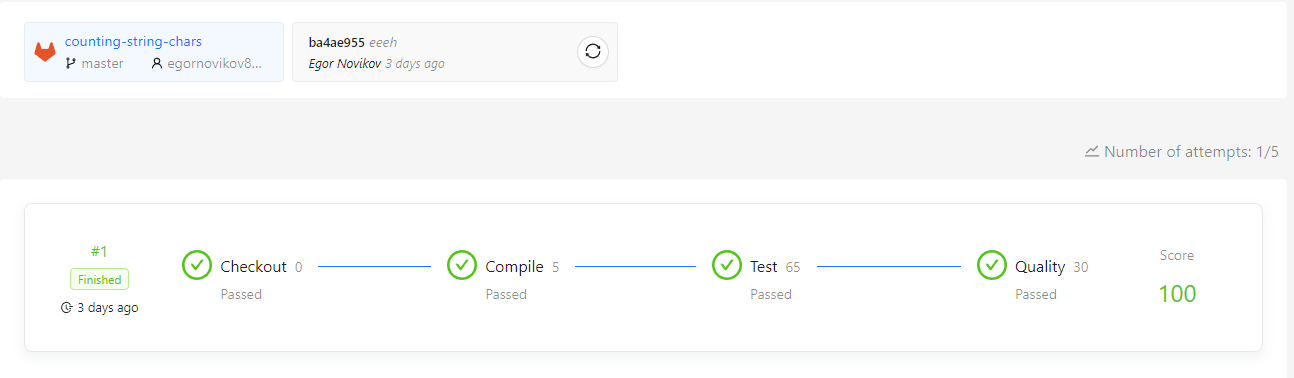
int result = GetPunctuationCountRecursive(str[1..]) + currentIncrement;

return result;

}

}

}

﻿ Результаты теста:

Задача №28

Название: Count of vowels

Условие: Решить задачу на получение количества гласных букв в строке

Код:

﻿using System;

namespace VowelCountTask

{

public static class StringHelper

{

/// <summary>

/// Calculates the count of vowels in the source string.

/// 'a', 'e', 'i', 'o', and 'u' are vowels.

/// </summary>

/// <param name="source">Source string.</param>

/// <returns>Count of vowels in the given string.</returns>

/// <exception cref="ArgumentException">Thrown when source string is null or empty.</exception>

public static int GetCountOfVowel(string source)

{

if (string.IsNullOrEmpty(source))

{

throw new ArgumentException("lol");

}

if (source.Length == 0)

{

return 0;

}

int result = 0;

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

{

if (source[i] == 'u' || source[i] == 'a' || source[i] == 'o' || source[i] == 'i' || source[i] == 'e')

{

result++;

}

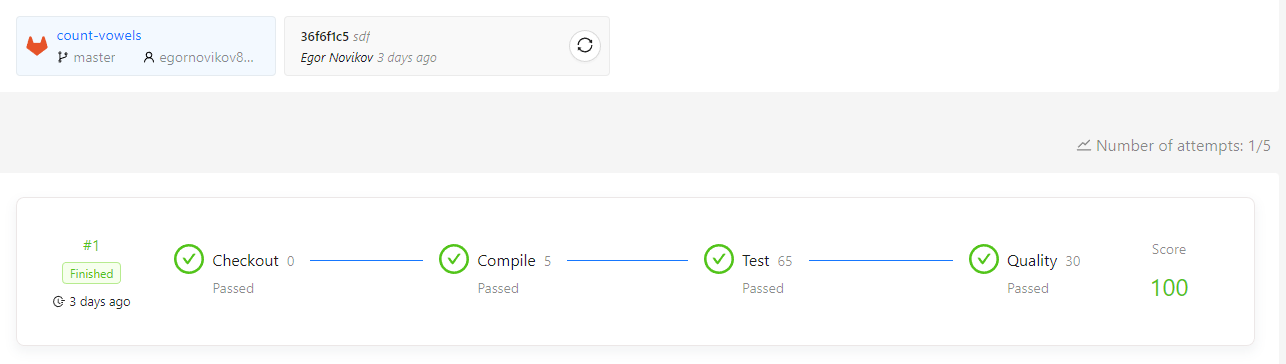
}

return result;

}

}

}

﻿ Результаты теста:

Задача №29

Название: Looking for Chars

Условие: Решить задачи на получение количества совпадающих символов в строках

Код:

﻿using System;

namespace LookingForChars

{

public static class CharsCounter

{

/// <summary>

/// Searches a string for all characters that are in <see cref="Array" />, and returns the number of occurrences of all characters.

/// </summary>

/// <param name="str">String to search.</param>

/// <param name="chars">One-dimensional, zero-based <see cref="Array"/> that contains characters to search for.</param>

/// <returns>The number of occurrences of all characters.</returns>

public static int GetCharsCount(string str, char[] chars)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (chars == null)

{

throw new ArgumentNullException(nameof(chars));

}

int result = 0;

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

{

for (int j = 0; j < chars.Length; j++)

{

if (str[i] == chars[j])

{

result++;

}

}

}

return result;

}

/// <summary>

/// Searches a string for all characters that are in <see cref="Array" />, and returns the number of occurrences of all characters within the range of elements in the <see cref="string"/> that starts at the specified index and ends at the specified index.

/// </summary>

/// <param name="str">String to search.</param>

/// <param name="chars">One-dimensional, zero-based <see cref="Array"/> that contains characters to search for.</param>

/// <param name="startIndex">A zero-based starting index of the search.</param>

/// <param name="endIndex">A zero-based ending index of the search.</param>

/// <returns>The number of occurrences of all characters within the specified range of elements in the <see cref="string"/>.</returns>

public static int GetCharsCount(string str, char[] chars, int startIndex, int endIndex)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (chars == null)

{

throw new ArgumentNullException(nameof(chars));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (endIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(endIndex));

}

if (startIndex > endIndex)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (endIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(endIndex));

}

int result = 0;

for (int i = startIndex; i <= endIndex; i++)

{

for (int j = 0; j < chars.Length; j++)

{

if (str[i] == chars[j])

{

result++;

}

}

}

return result;

}

/// <summary>

/// Searches a string for a limited number of characters that are in <see cref="Array" />, and returns the number of occurrences of all characters within the range of elements in the <see cref="string"/> that starts at the specified index and ends at the specified index.

/// </summary>

/// <param name="str">String to search.</param>

/// <param name="chars">One-dimensional, zero-based <see cref="Array"/> that contains characters to search for.</param>

/// <param name="startIndex">A zero-based starting index of the search.</param>

/// <param name="endIndex">A zero-based ending index of the search.</param>

/// <param name="limit">A maximum number of characters to search.</param>

/// <returns>The limited number of occurrences of characters to search for within the specified range of elements in the <see cref="string"/>.</returns>

public static int GetCharsCount(string str, char[] chars, int startIndex, int endIndex, int limit)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (chars == null)

{

throw new ArgumentNullException(nameof(chars));

}

if (startIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (endIndex < 0)

{

throw new ArgumentOutOfRangeException(nameof(endIndex));

}

if (limit < 0)

{

throw new ArgumentOutOfRangeException(nameof(limit));

}

if (startIndex > endIndex)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (startIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(startIndex));

}

if (endIndex > str.Length)

{

throw new ArgumentOutOfRangeException(nameof(endIndex));

}

int result = 0;

for (int i = startIndex; i <= endIndex; i++)

{

for (int j = 0; j < chars.Length; j++)

{

if (str[i] == chars[j] && result < limit)

{

result++;

}

}

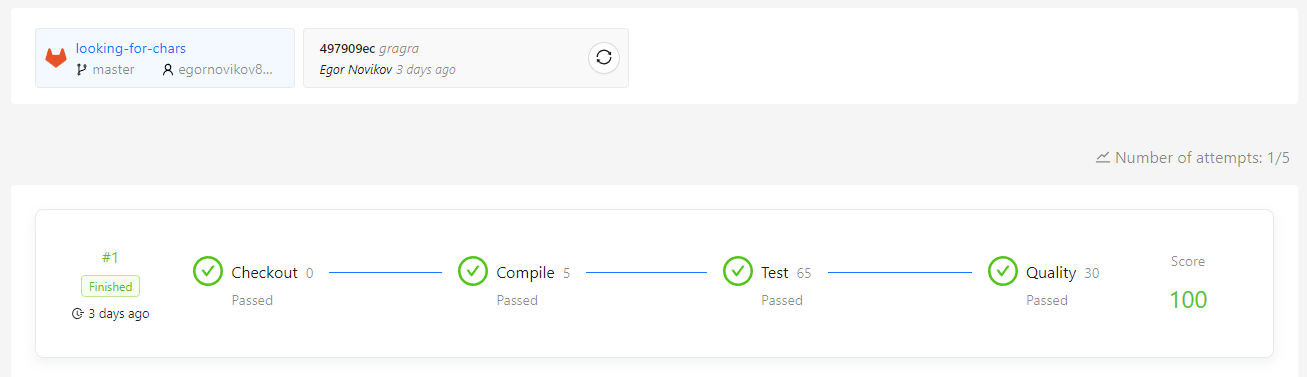
}

return result;

}

}

}

 Результаты теста:

Задача №30

Название: Shuffle Characters

Условие: Решить задачу на перестановки символов в строке

Код:

﻿ ﻿using System;

using System.Linq;

namespace ShuffleCharacters

{

public static class StringExtension

{

/// <summary>

/// Shuffles characters in source string according some rule.

/// </summary>

/// <param name="source">The source string.</param>

/// <param name="count">The count of iterations.</param>

/// <returns>Result string.</returns>

/// <exception cref="ArgumentException">Source string is null or empty or white spaces.</exception>

/// <exception cref="ArgumentException">Count of iterations is less than 0.</exception>

public static string ShuffleChars(string source, int count)

{

if (source is null)

{

throw new ArgumentException("sourc");

}

if (source == " ")

{

throw new ArgumentException("sourc");

}

#pragma warning disable S2583 // Conditionally executed code should be reachable

if (source == " \t\n \t \r")

{

throw new ArgumentException("sourc");

}

#pragma warning restore S2583 // Conditionally executed code should be reachable

if (source.Length == 0)

{

throw new ArgumentException("sourc");

}

if (count < 0)

{

throw new ArgumentException("sourc");

}

char[] sourceChar = source.ToCharArray();

char[] array = new char[source.Length];

char[] start = sourceChar.ToArray();

var t = 0;

int mid = (source.Length / 2) + (source.Length % 2);

for (int k = 0; k < count; k++)

{

(array, sourceChar) = (sourceChar, array);

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

{

sourceChar[i] = array[i \* 2];

}

for (int i = mid, j = 1; i < source.Length; i++, j += 2)

{

sourceChar[i] = array[j];

}

t++;

if (sourceChar.SequenceEqual(start))

{

while (count > k + t)

{

count -= t;

}

}

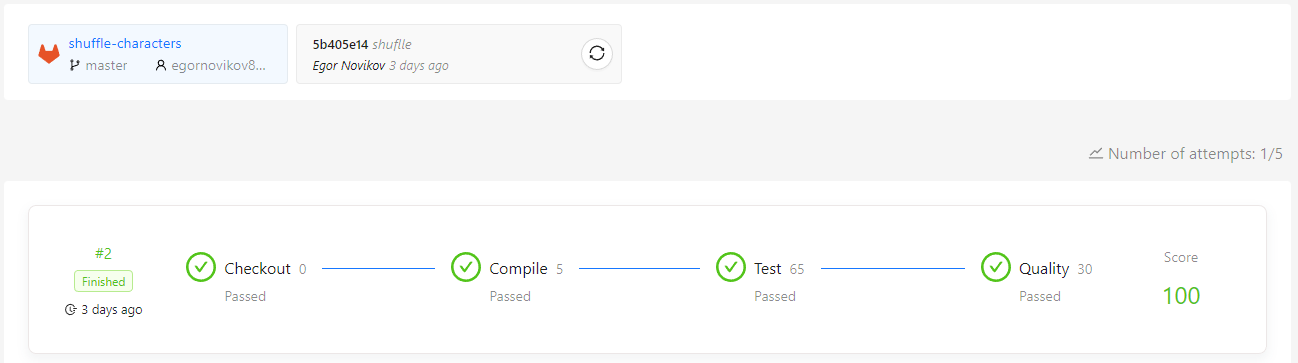
}

return new string(sourceChar);

}

}

}

 Результаты теста:

Задача №31

Название: Parsing strings

Условие: Научиться работать с операциями приведения строк к другим типам данных

Код:

﻿BooleanParser.cs:

using System;

namespace ParsingStrings

{

public static class BooleanParser

{

/// <summary>

/// Tries to convert the specified string representation of a logical value to its Boolean equivalent.

/// </summary>

/// <param name="str">A string containing the value to convert.</param>

/// <param name="result">When this method returns, if the conversion succeeded, contains true if value is equal to <see cref="bool.TrueString"/> or false if value is equal to <see cref="bool.FalseString"/>. If the conversion failed, contains false.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseBoolean(string str, out bool result)

{

return bool.TryParse(str, out result);

}

/// <summary>

/// Converts the specified string representation of a logical value to its Boolean equivalent.

/// </summary>

/// <param name="str">A string containing the value to convert.</param>

/// <returns>true if value is equivalent to <see cref="bool.TrueString"/>; false if value is equivalent to <see cref="bool.FalseString"/>.</returns>

public static bool ParseBoolean(string str)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return false;

}

return bool.Parse(str);

}

}

}

CharParser.cs:

﻿using System;

namespace ParsingStrings

{

public static class CharParser

{

/// <summary>

/// Converts the value of the specified string to its equivalent Unicode character.

/// </summary>

/// <param name="str">A string that contains a single character, or null.</param>

/// <param name="result">When this method returns, contains a Unicode character equivalent to the sole character in <see cref="str"/>, if the conversion succeeded, or an undefined value if the conversion failed.</param>

/// <returns>true if the <see cref="str"/> parameter was converted successfully; otherwise, false.</returns>

public static bool TryParseChar(string str, out char result)

{

return char.TryParse(str, out result);

}

/// <summary>

/// Converts the value of the specified string to its equivalent Unicode character.

/// </summary>

/// <param name="str">A string that contains a single character, or null.</param>

/// <returns>A Unicode character equivalent to the sole character in <see cref="str"/>. If a formatting error occurs returns space character.</returns>

public static char ParseChar(string str)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length != 1)

{

return ' ';

}

return char.Parse(str);

}

}

}

FloatingPointParser.cs:

﻿using System;

namespace ParsingStrings

{

public static class FloatingPointParser

{

/// <summary>

/// Converts the string representation of a number to its single-precision floating-point number equivalent.

/// </summary>

/// <param name="str">A string representing a number to convert.</param>

/// <param name="result">When this method returns, contains single-precision floating-point number equivalent to the numeric value or symbol contained in <paramref name="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <paramref name="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseFloat(string str, out float result)

{

return float.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its single-precision floating-point number equivalent.

/// </summary>

/// <param name="str">A string that contains a number to convert.</param>

/// <returns>A single-precision floating-point number equivalent to the numeric str or symbol specified in <paramref name="str"/>. If a formatting error occurs returns NaN. </returns>

public static float ParseFloat(string str)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return float.NaN;

}

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

{

if (!char.IsDigit(str[i]) && str[i] != '+' && str[i] != '-' && str[i] != 'E' && str[i] != '.' && str[i] != ' ')

{

return float.NaN;

}

}

return float.Parse(str);

}

/// <summary>

/// Converts the string representation of a number to its double-precision floating-point number equivalent.

/// </summary>

/// <param name="str">A string representing a number to convert.</param>

/// <param name="result">When this method returns, contains double-precision floating-point number equivalent to the numeric value or symbol contained in <paramref name="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <paramref name="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseDouble(string str, out double result)

{

return double.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its double-precision floating-point number equivalent.

/// </summary>

/// <param name="str">A string that contains a number to convert.</param>

/// <returns>A double-precision floating-point number equivalent to the numeric str or symbol specified in <paramref name="str"/>. If a formatting error occurs returns Epsilon.</returns>

public static double ParseDouble(string str)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return 4.9406564584124654E-324d;

}

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

{

if (!char.IsDigit(str[i]) && str[i] != '+' && str[i] != '-' && str[i] != 'E' && str[i] != '.' && str[i] != ' ')

{

return 4.9406564584124654E-324d;

}

}

return double.Parse(str);

}

/// <summary>

/// Converts the string representation of a number to its Decimal equivalent.

/// </summary>

/// <param name="str">The string representation of the number to convert.</param>

/// <param name="result">When this method returns, contains the Decimal number that is equivalent to the numeric value contained in <paramref name="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <paramref name="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseDecimal(string str, out decimal result)

{

return decimal.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its Decimal equivalent.

/// </summary>

/// <param name="str">The string representation of the number to convert.</param>

/// <returns>The equivalent to the number contained in <paramref name="str"/>.</returns>

public static decimal ParseDecimal(string str)

{

if (str == null)

{

throw new ArgumentNullException(nameof(str));

}

if (str.Length == 0)

{

return decimal.MinValue;

}

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

{

if (!char.IsDigit(str[i]) && str[i] != '+' && str[i] != '-' && str[i] != 'E' && str[i] != '.' && str[i] != ' ')

{

return decimal.MinValue;

}

}

return decimal.Parse(str);

}

}

}

NumberParser.cs:

using System;

namespace ParsingStrings

{

public static class NumberParser

{

/// <summary>

/// Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the operation succeeded.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <param name="result">When this method returns, contains the 32-bit signed integer value equivalent of the number contained in <see cref="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseInteger(string str, out int result)

{

return int.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 32-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <returns>A 32-bit signed integer equivalent to the number contained in <see cref="str"/>. If a formatting error occurs returns zero. If an overflow error occurs returns minus one.</returns>

public static int ParseInteger(string str)

{

try

{

return int.Parse(str);

}

catch (FormatException)

{

return 0;

}

catch (OverflowException)

{

return -1;

}

}

/// <summary>

/// Tries to convert the string representation of a number to its 32-bit unsigned integer equivalent. A return value indicates whether the conversion succeeded or failed.

/// </summary>

/// <param name="str">A string that represents the number to convert.</param>

/// <param name="result">When this method returns, contains the 32-bit unsigned integer value that is equivalent to the number contained in <see cref="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseUnsignedInteger(string str, out uint result)

{

return uint.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 32-bit unsigned integer equivalent.

/// </summary>

/// <param name="str">A string representing the number to convert.</param>

/// <returns>A 32-bit unsigned integer equivalent to the number contained in <see cref="str"/>. If a formatting error occurs returns minimum value of unsigned int. If an overflow error occurs returns maximum value of unsigned int.</returns>

public static uint ParseUnsignedInteger(string str)

{

try

{

return uint.Parse(str);

}

catch (FormatException)

{

return 0;

}

catch (OverflowException)

{

return 4294967295;

}

}

/// <summary>

/// Tries to convert the string representation of a number to its Byte equivalent, and returns a value that indicates whether the conversion succeeded.

/// </summary>

/// <param name="str">A string that contains a number to convert.</param>

/// <param name="result">When this method returns, contains the Byte value equivalent to the number contained in <see cref="str"/> if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseByte(string str, out byte result)

{

return byte.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its Byte equivalent.

/// </summary>

/// <param name="str">A string that contains a number to convert.</param>

/// <returns>A byte value that is equivalent to the number contained in <see cref="str"/>. If a formatting error occurs returns maximum value of byte. If an overflow error occurs returns minimum value of byte.</returns>

public static byte ParseByte(string str)

{

try

{

return byte.Parse(str);

}

catch (FormatException)

{

return 255;

}

catch (OverflowException)

{

return 0;

}

}

/// <summary>

/// Tries to convert the string representation of a number to its SByte equivalent, and returns a value that indicates whether the conversion succeeded.

/// </summary>

/// <param name="str">A string that contains a number to convert.</param>

/// <param name="result">When this method returns, contains the 8-bit signed integer value that is equivalent to the number contained in <see cref="str"/> if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TrySignedByte(string str, out sbyte result)

{

return sbyte.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 8-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string that represents a number to convert.</param>

/// <returns>An 8-bit signed integer that is equivalent to the number contained in the <see cref="str"/> parameter. If a formatting error occurs returns maximum value of signed byte.</returns>

public static sbyte ParseSignedByte(string str)

{

try

{

return sbyte.Parse(str);

}

catch (FormatException)

{

return sbyte.MaxValue;

}

}

/// <summary>

/// Converts the string representation of a number to its 16-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <param name="result">When this method returns, contains the 16-bit signed integer value equivalent to the number contained in <see cref="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseShort(string str, out short result)

{

return short.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 16-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <returns>A 16-bit signed integer equivalent to the number contained in <see cref="str"/>. If an overflow error occurs returns maximum value of short.</returns>

public static short ParseShort(string str)

{

try

{

return short.Parse(str);

}

catch (OverflowException)

{

return -1;

}

}

/// <summary>

/// Converts the string representation of a number to its 16-bit unsigned integer equivalent.

/// </summary>

/// <param name="str">A string that represents the number to convert.</param>

/// <param name="result">When this method returns, contains the 16-bit unsigned integer value that is equivalent to the number contained in <see cref="str"/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseUnsignedShort(string str, out ushort result)

{

return ushort.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 16-bit unsigned integer equivalent.

/// </summary>

/// <param name="str">A string that represents the number to convert.</param>

/// <returns>A 16-bit unsigned integer equivalent to the number contained in <see cref="str"/>. If a formatting error occurs returns zero. If an overflow error occurs returns maximum value of unsigned short.</returns>

public static ushort ParseUnsignedShort(string str)

{

try

{

return ushort.Parse(str);

}

catch (FormatException)

{

return 0;

}

catch (OverflowException)

{

return ushort.MaxValue;

}

}

/// <summary>

/// Converts the string representation of a number to its 64-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <param name="result">When this method returns, contains the 64-bit signed integer value equivalent of the number contained in <see cref="str"/>, if the conversion succeeded, or zero if the conversion failed. </param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseLong(string str, out long result)

{

return long.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 64-bit signed integer equivalent.

/// </summary>

/// <param name="str">A string containing a number to convert.</param>

/// <returns>A 64-bit signed integer equivalent to the number contained in <see cref="str"/>. If a formatting error occurs returns minimum value of long. If an overflow error occurs returns minus one.</returns>

public static long ParseLong(string str)

{

try

{

return long.Parse(str);

}

catch (FormatException)

{

return long.MinValue;

}

catch (OverflowException)

{

return -1;

}

}

/// <summary>

/// Tries to convert the string representation of a number to its 64-bit unsigned integer equivalent.

/// </summary>

/// <param name="str">A string that represents the number to convert.</param>

/// <param name="result">When this method returns, contains the 64-bit unsigned integer value that is equivalent to the number contained in <see cref=""/>, if the conversion succeeded, or zero if the conversion failed.</param>

/// <returns>true if <see cref="str"/> was converted successfully; otherwise, false.</returns>

public static bool TryParseUnsignedLong(string str, out ulong result)

{

return ulong.TryParse(str, out result);

}

/// <summary>

/// Converts the string representation of a number to its 64-bit unsigned integer equivalent.

/// </summary>

/// <param name="str">A string that represents the number to convert.</param>

/// <returns>A 64-bit unsigned integer equivalent to the number contained in <see cref="str"/>.</returns>

public static ulong ParseUnsignedLong(string str)

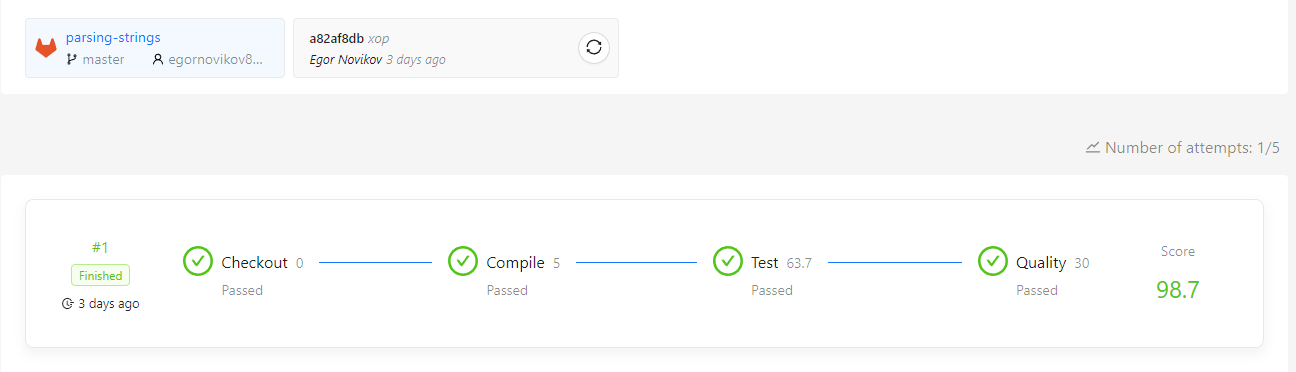
{

return ulong.Parse(str);

}

}

}

 Результаты теста:

Задача №32

Название: RGB converter

Условие: Решить задачу на получение кода цвета по его RGB параметрам.

Код:

﻿ using System;

namespace RgbConverter

{

public static class Rgb

{

/// <summary>

/// Gets hexadecimal representation source RGB decimal values.

/// </summary>

/// <param name="red">The valid decimal value for RGB is in the range 0-255.</param>

/// <param name="green">The valid decimal value for RGB is in the range 0-255.</param>

/// <param name="blue">The valid decimal value for RGB is in the range 0-255.</param>

/// <returns>Returns hexadecimal representation source RGB decimal values.</returns>

public static string GetHexRepresentation(int red, int green, int blue)

{

string rgb = string.Empty;

string tmp = string.Empty;

if (red > 255)

{

tmp = "FF";

}

else if (red < 0)

{

tmp = "00";

}

else

{

do

{

int r = red % 16;

if (r > 9)

{

r += (int)'A' - 10;

}

else

{

r += (int)'0';

}

tmp = (char)r + tmp;

red /= 16;

}

while (red != 0);

}

if (tmp.Length == 1)

{

tmp = tmp.Insert(0, "0");

}

rgb += tmp;

tmp = string.Empty;

if (green > 255)

{

tmp = "FF";

}

else if (green < 0)

{

tmp = "00";

}

else

{

do

{

int r = green % 16;

if (r > 9)

{

r += (int)'A' - 10;

}

else

{

r += (int)'0';

}

tmp = (char)r + tmp;

green /= 16;

}

while (green != 0);

}

if (tmp.Length == 1)

{

tmp = tmp.Insert(0, "0");

}

rgb += tmp;

tmp = string.Empty;

if (blue > 255)

{

tmp = "FF";

}

else if (blue < 0)

{

tmp = "00";

}

else

{

do

{

int r = blue % 16;

if (r > 9)

{

r += (int)'A' - 10;

}

else

{

r += (int)'0';

}

tmp = (char)r + tmp;

blue /= 16;

}

while (blue != 0);

}

if (tmp.Length == 1)

{

tmp = tmp.Insert(0, "0");

}

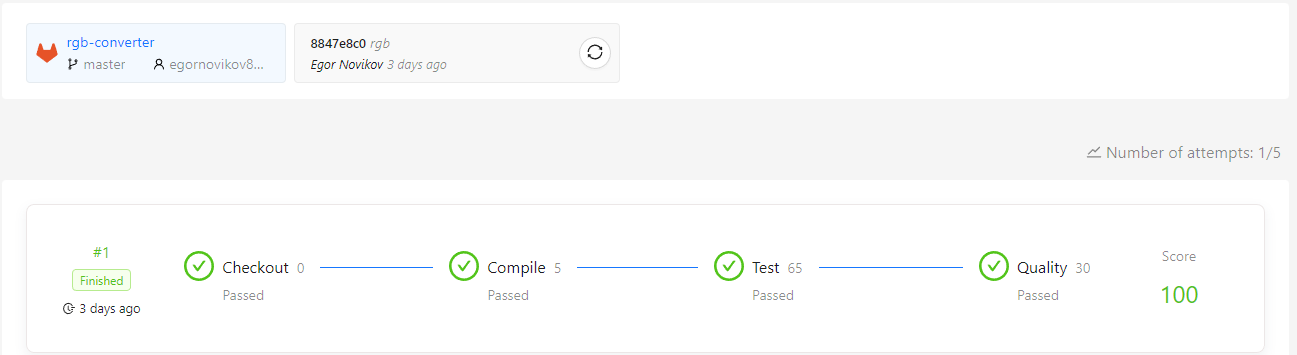
rgb += tmp;

return rgb;

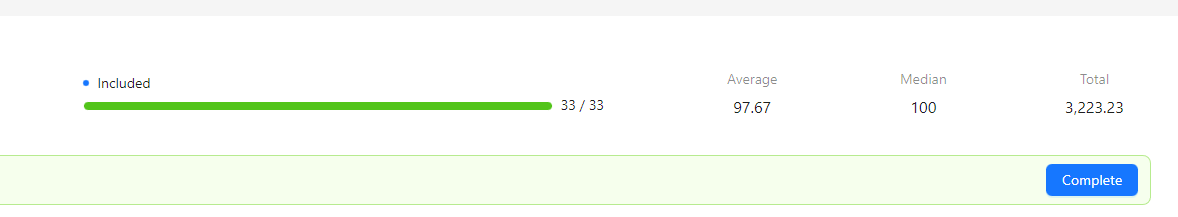
}

}

}

 Результаты теста:

1. **Итоги**

В результате был полностью был пройден курс .Net Development (2021/Q4/1)

Были изучены такие основы языка .Net C# как работа с различными типами данных, работа с вводом-выводом, ветвление (if-else statements), работа с исключениями, работа с различными конструкциями циклов (for, while, do-while), работа с массивами, работа со строками, форматирование и приведение строк.

После завершения учебной практики планирую продолжить изучение языка самостоятельно, а также переделать некоторые задание на более высокий балл и повторить изученный материал.