**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Программирование |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по практической работе №5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  |  | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | Разработка программных модулем |
|  | | |  | (наименование дисциплины) |

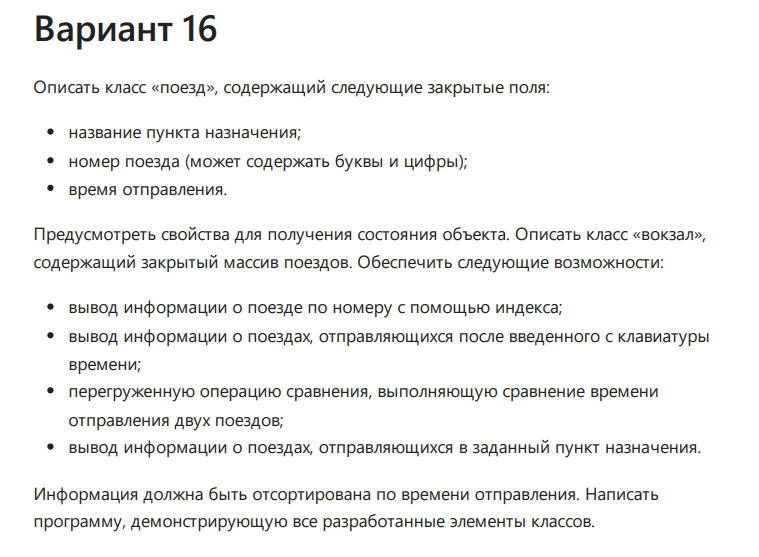
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Стрельцов Егор Михайлович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | VДКИП-111прог |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев И. В. |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2024 г**

Лабораторная работа 8.Классы и операции.

Вариант 16.



**Листинг кода:**

using System;

using System.Collections.Generic;

using System.Linq;

public class Train : IComparable<Train>

{

private string destination;

private string number;

private TimeSpan departureTime;

public Train(string destination, string number, TimeSpan departureTime)

{

this.destination = destination;

this.number = number;

this.departureTime = departureTime;

}

public string Destination => destination;

public string Number => number;

public TimeSpan DepartureTime => departureTime;

public int CompareTo(Train other)

{

if (other == null) return 1;

return this.departureTime.CompareTo(other.departureTime);

}

public override string ToString()

{

return $"Поезд №{number} в {destination}, отправление в {departureTime:hh\\:mm}";

}

}

public class Station

{

private List<Train> trains = new List<Train>();

public void AddTrain(Train train)

{

trains.Add(train);

}

public Train this[string number]

{

get

{

var train = trains.FirstOrDefault(t => t.Number == number);

if (train == null)

throw new ArgumentException($"Поезд с номером {number} не найден");

return train;

}

}

public IEnumerable<Train> GetTrainsAfterTime(TimeSpan time)

{

return trains.Where(t => t.DepartureTime > time).OrderBy(t => t.DepartureTime);

}

public IEnumerable<Train> GetTrainsToDestination(string destination)

{

return trains.Where(t => t.Destination == destination).OrderBy(t => t.DepartureTime);

}

}

class Program

{

static void Main()

{

Station station = new Station();

// Добавляем тестовые поезда

station.AddTrain(new Train("Москва", "123А", new TimeSpan(12, 30, 0)));

station.AddTrain(new Train("Санкт-Петербург", "456Б", new TimeSpan(15, 45, 0)));

station.AddTrain(new Train("Москва", "789В", new TimeSpan(8, 15, 0)));

station.AddTrain(new Train("Казань", "321Г", new TimeSpan(18, 20, 0)));

station.AddTrain(new Train("Москва", "654Д", new TimeSpan(10, 0, 0)));

// Демонстрация работы

// 1. Получение информации о поезде по номеру

Console.WriteLine("1. Информация о поезде по номеру:");

try

{

Console.WriteLine(station["456Б"]);

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

// 2. Поезда, отправляющиеся после указанного времени

Console.WriteLine("\n2. Поезда, отправляющиеся после 12:00:");

foreach (var train in station.GetTrainsAfterTime(new TimeSpan(12, 0, 0)))

{

Console.WriteLine(train);

}

// 3. Сравнение времени отправления двух поездов

Console.WriteLine("\n3. Сравнение времени отправления:");

var train1 = station["123А"];

var train2 = station["456Б"];

Console.WriteLine($"{train1.Number} ({train1.DepartureTime}) < {train2.Number} ({train2.DepartureTime}): {train1.CompareTo(train2) < 0}");

// 4. Поезда в указанный пункт назначения (отсортированные)

Console.WriteLine("\n4. Поезда в Москву (отсортированные по времени):");

foreach (var train in station.GetTrainsToDestination("Москва"))

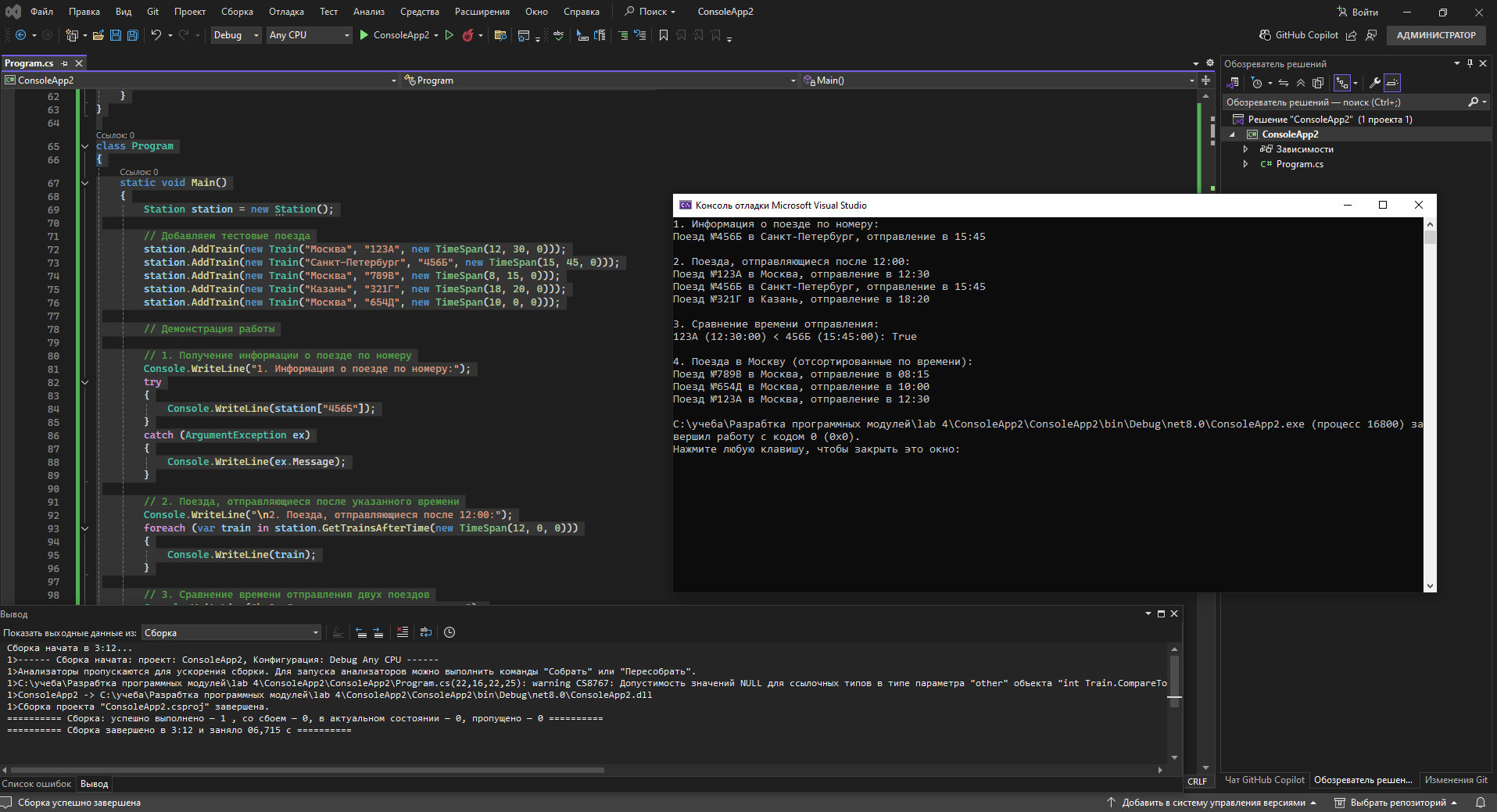
{

Console.WriteLine(train);

}

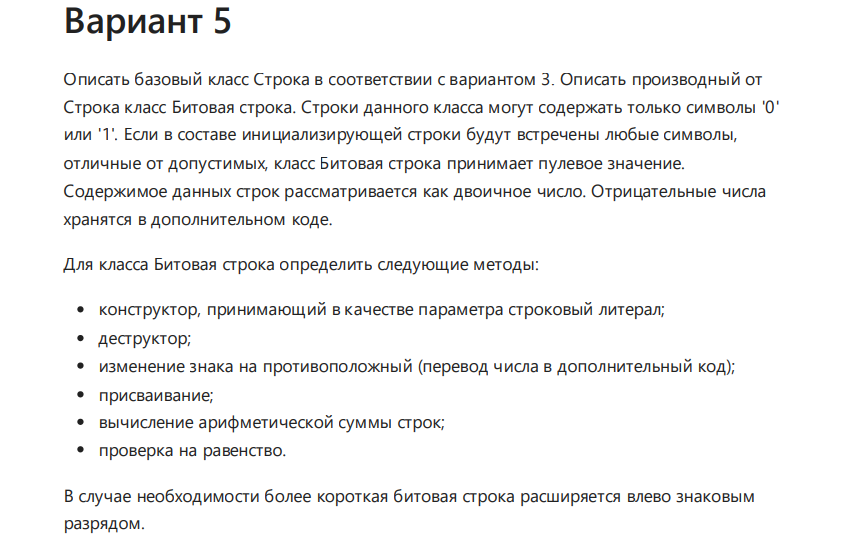
}

}



**Рис. 1. Успешное выполнение программы.**

Лабораторная работа 9. Наследование. Вариант 5.



**Листинг кода:**

using System;

using System.Linq;

// Базовый класс Строка

public class StringBase

{

protected string value;

public StringBase(string str)

{

value = str;

}

public virtual int Length => value.Length;

public virtual string Value => value;

public override string ToString() => value;

}

// Производный класс БитоваяСтрока

public class BitString : StringBase, IDisposable, IEquatable<BitString>

{

public BitString(string str) : base(ValidateBitString(str) ? str : "")

{

if (!ValidateBitString(str))

value = "";

}

private static bool ValidateBitString(string str)

{

return str.All(c => c == '0' || c == '1');

}

// Деструктор

~BitString()

{

Console.WriteLine($"Битовая строка '{value}' уничтожается");

}

// Реализация IDisposable

public void Dispose()

{

GC.SuppressFinalize(this);

}

// Изменение знака (перевод в дополнительный код)

public BitString Negate()

{

if (string.IsNullOrEmpty(value)) return new BitString("");

// Инвертируем биты

string inverted = new string(value.Select(c => c == '0' ? '1' : '0').ToArray());

// Добавляем 1

var result = AddOne(inverted);

return new BitString(result);

}

private static string AddOne(string binary)

{

char[] chars = binary.ToCharArray();

bool carry = true;

for (int i = chars.Length - 1; i >= 0 && carry; i--)

{

if (chars[i] == '0')

{

chars[i] = '1';

carry = false;

}

else

{

chars[i] = '0';

}

}

if (carry)

{

return '1' + new string(chars);

}

return new string(chars);

}

// Присваивание

public void Assign(BitString other)

{

value = other.value;

}

// Арифметическая сумма

public static BitString operator +(BitString a, BitString b)

{

if (string.IsNullOrEmpty(a.value)) return new BitString(b.value);

if (string.IsNullOrEmpty(b.value)) return new BitString(a.value);

// Выравнивание длин

int maxLength = Math.Max(a.value.Length, b.value.Length);

string aPadded = a.value.PadLeft(maxLength, a.value[0]);

string bPadded = b.value.PadLeft(maxLength, b.value[0]);

string result = BinaryAddition(aPadded, bPadded);

// Удаляем лишний бит, если он есть

if (result.Length > maxLength)

result = result.Substring(1);

return new BitString(result);

}

private static string BinaryAddition(string a, string b)

{

char[] result = new char[a.Length + 1];

int carry = 0;

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

{

int sum = (a[i] - '0') + (b[i] - '0') + carry;

result[i + 1] = (sum % 2 == 0) ? '0' : '1';

carry = sum / 2;

}

result[0] = (carry == 1) ? '1' : '0';

return new string(result);

}

// Проверка на равенство

public bool Equals(BitString other)

{

if (other == null) return false;

// Нормализация (удаление ведущих одинаковых битов)

string a = Normalize(value);

string b = Normalize(other.value);

return a == b;

}

private static string Normalize(string s)

{

if (string.IsNullOrEmpty(s)) return "";

int i = 0;

while (i < s.Length - 1 && s[i] == s[i + 1])

i++;

return s.Substring(i);

}

public override bool Equals(object obj) => Equals(obj as BitString);

public override int GetHashCode() => value.GetHashCode();

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

{

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

if (left is null || right is null) return false;

return left.Equals(right);

}

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

}

class Program

{

static void Main()

{

// Тестирование

BitString bs1 = new BitString("0101");

BitString bs2 = new BitString("1100");

BitString invalid = new BitString("0123"); // Некорректная строка

Console.WriteLine($"bs1: {bs1}");

Console.WriteLine($"bs2: {bs2}");

Console.WriteLine($"invalid: {invalid}");

// Изменение знака

BitString negBs1 = bs1.Negate();

Console.WriteLine($"Отрицание bs1: {negBs1}");

// Сумма

BitString sum = bs1 + bs2;

Console.WriteLine($"Сумма bs1 + bs2: {sum}");

// Проверка равенства

BitString bs3 = new BitString("0101");

Console.WriteLine($"bs1 == bs3: {bs1 == bs3}");

// Присваивание

bs1.Assign(bs2);

Console.WriteLine($"После присваивания bs1 = bs2: {bs1}");

// Использование using для вызова Dispose

using (BitString temp = new BitString("1010"))

{

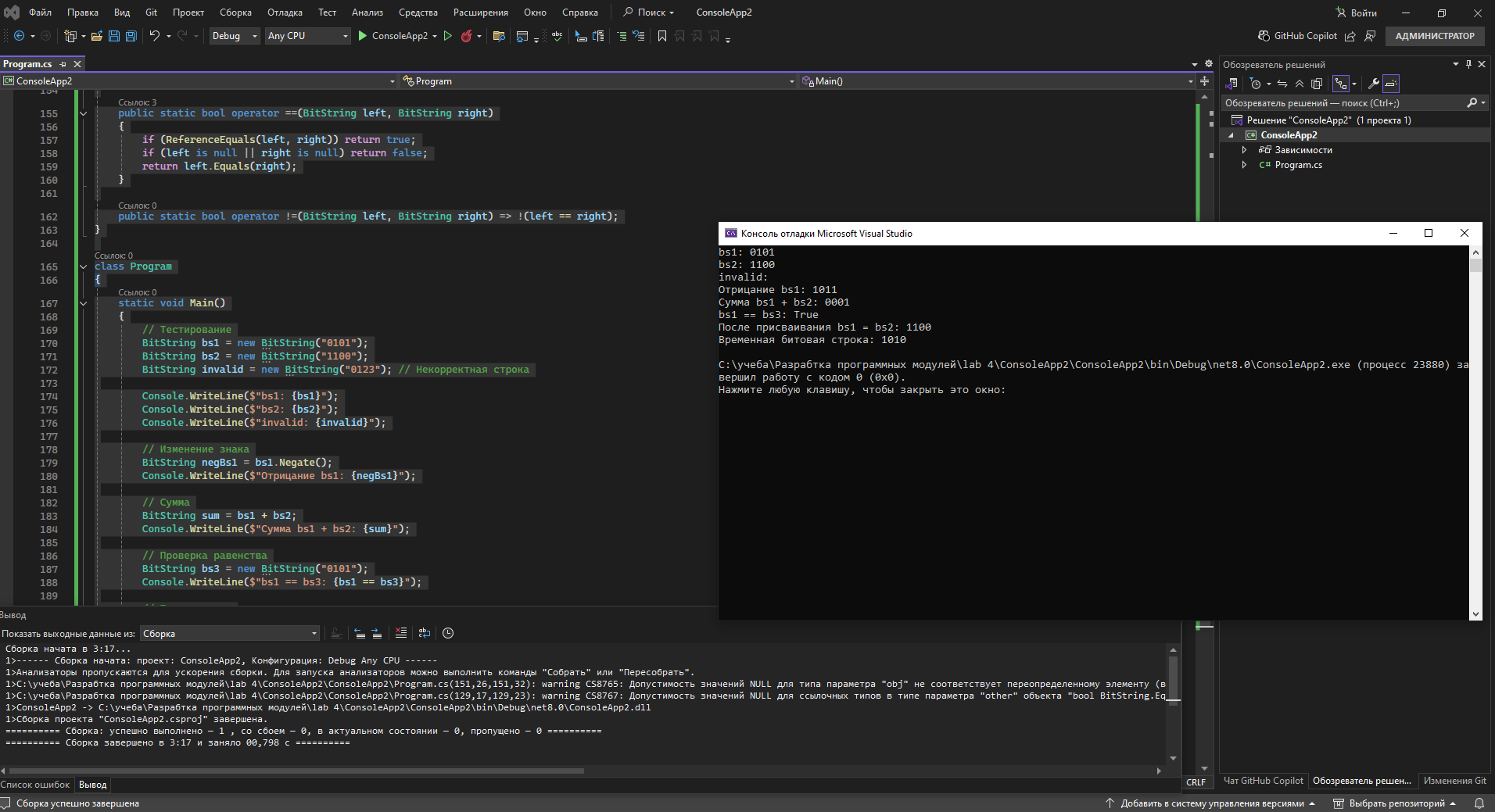
Console.WriteLine($"Временная битовая строка: {temp}");

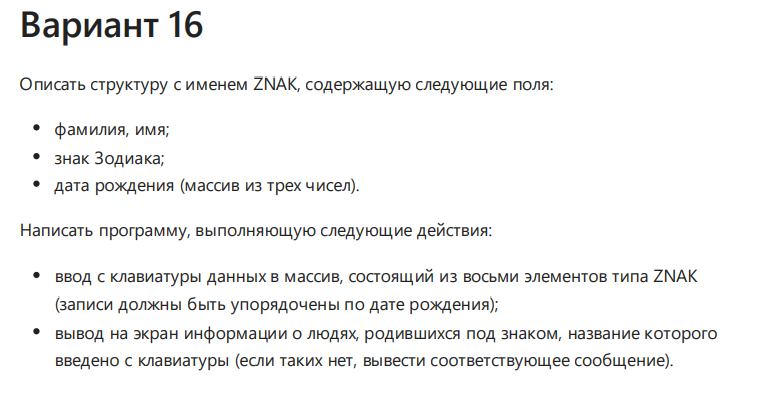
}

// Деструктор будет вызван автоматически для других объектов

}

}

 **Рис. 2. Успешное выполнение программы**

Лабораторная работа 10. Структуры. Вариант 16

**Листинг кода:**

using System;

using System.Linq;

// Структура ZNAK

struct ZNAK

{

public string LastName;

public string FirstName;

public string ZodiacSign;

public int[] BirthDate; // [день, месяц, год]

public ZNAK(string lastName, string firstName, string zodiacSign, int[] birthDate)

{

LastName = lastName;

FirstName = firstName;

ZodiacSign = zodiacSign;

BirthDate = birthDate;

}

public override string ToString()

{

return $"{LastName} {FirstName}, {ZodiacSign}, {BirthDate[0]}.{BirthDate[1]}.{BirthDate[2]}";

}

}

class Program

{

static void Main()

{

// Создаем массив из 8 элементов

ZNAK[] people = new ZNAK[8];

Console.WriteLine("Введите данные для 8 человек:");

// Ввод данных

for (int i = 0; i < 8; i++)

{

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

Console.Write("Фамилия: ");

string lastName = Console.ReadLine();

Console.Write("Имя: ");

string firstName = Console.ReadLine();

Console.Write("Знак Зодиака: ");

string zodiacSign = Console.ReadLine();

Console.WriteLine("Дата рождения:");

Console.Write("День: ");

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

Console.Write("Месяц: ");

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

Console.Write("Год: ");

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

people[i] = new ZNAK(lastName, firstName, zodiacSign, new int[] { day, month, year });

}

// Сортировка по дате рождения (год -> месяц -> день)

people = people.OrderBy(p => p.BirthDate[2])

.ThenBy(p => p.BirthDate[1])

.ThenBy(p => p.BirthDate[0])

.ToArray();

Console.WriteLine("\nОтсортированный список по дате рождения:");

foreach (var person in people)

{

Console.WriteLine(person);

}

// Поиск по знаку Зодиака

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

string searchSign = Console.ReadLine();

var foundPeople = people.Where(p => p.ZodiacSign.Equals(searchSign, StringComparison.OrdinalIgnoreCase)).ToArray();

if (foundPeople.Length > 0)

{

Console.WriteLine($"\nЛюди со знаком {searchSign}:");

foreach (var person in foundPeople)

{

Console.WriteLine(person);

}

}

else

{

Console.WriteLine($"\nЛюдей со знаком {searchSign} не найдено.");

}

}

}