**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

ФАИС

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

по дисциплине **«Языки программирования высокого уровня»**

на тему: **«Коллекции структур данных. Классы прототипы»**

Выполнил: студент гр. ИП-22

Коваленко А. И.

Принял: ст. преподаватель Романькова Т. Л.

Дата сдачи отчета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**Коллекции структур данных. Классы прототипы**

**Цель**: *изучить возможности использования коллекций пространства имен System.Collection. Научиться создавать и использовать классы-прототипы для хранения и обработки данных.*

**Вариант 7.**

Подготовить текстовый файл, содержащий информацию о пациентах терапевтического отделения больницы: фамилию, год рождения, дату поступления, срок лечения в днях, код диагноза.

Описать структуру Patient, которая должна содержать следующие элементы: поля для хранения фамилии, даты рождения, даты поступления, диагноза, свойство для определения даты выписки, метод для определения возраста. Для описания последнего поля необходимо создать перечисление (например, пусть Diagnoz это перечисление с константами Грипп=12, Ангина=32, Пневмония=145, тогда поле с диагнозом должно быть описано так: Diagnoz diagnoz;). Структура должна реализовывать IComparable<T>.

Создать класс-прототип Collection<T> для хранения и обработки экземпляров структур, наложить ограничение на параметр типа данных: элементы коллекции должны быть значимого типа, тип–аргумент должен реализовывать интерфейс IComparable<T>. Объекты структур хранить в коллекции типа Stack<T>. Создать в этом классе метод Search для поиска информации по заданному критерию (критерий передавать через параметр-делегат: стандартный или созданный, результат – коллекция List<T>). Реализовать интерфейс IEnumerable<T> с использованием итератора.

Написать консольное приложение, выполняющее следующее функции:

* Чтение данных из файла и создание объекта Collection<Patient>.
* Вывод информации в виде таблицы.
* Добавление данных о новом пациенте.
* Сортировку по убыванию продолжительности пребывания в больнице или по возрасту пациента (по выбору пользователя).
* Поиск пациентов не старше 20 лет с заданным диагнозом заболевания (использовать метод Search).
* Определить общее количество пациентов старше 60 лет (использовать LINQ).
* Определить средний срок пребывания в больнице по каждому диагнозу (использовать LINQ).

Листинг программы:

**Program.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;

namespace lab2

{

class Program

{

static void Main(string[] args)

{

Collection<Patient> collectionPatients = new Collection<Patient>();

ReadingInformationFromFiles(collectionPatients);

OutputInformation(collectionPatients);

InputDataNewPatient(collectionPatients);

OutputInformation(collectionPatients);

SelectingSort(collectionPatients);

Console.WriteLine("List of patients with pneumonia: ");

OutputInformationInList(collectionPatients.Search(CriterionUpTo20YearsWithCodeDiagnosis));

var numberPatientsAfterSixty = collectionPatients.Where(x => x.Age() > 60).Count();

Console.WriteLine($"Amount of patients older than 60: {numberPatientsAfterSixty}");

var averageDeadlineOnDiagnosis = from patient in collectionPatients // источник данных с переменной диапазона

group patient by patient.Diagnosis into g //получение последовательности групп по ключу

select new//преобразования исходных данных в последовательности новых типов

{

Diag = g.Select(p => p.Diagnosis).Distinct(), //Distinct: удаляет дублирующиеся элементы из коллекции

AverageDeadline = g.Select(p => p.LengthOfTreatment).Average()

};

foreach (var group in averageDeadlineOnDiagnosis)

{

foreach (var item in group.Diag)

{

Console.WriteLine($"Average length of hospital stay with a diagnosis {item}: {group.AverageDeadline} days.");

}

}

Console.ReadKey();

}

private static void ReadingInformationFromFiles(Collection<Patient> collectionPatients)

{

try

{

StreamReader reader = new StreamReader("patients.txt");

string line;

while (!reader.EndOfStream)

{

line = reader.ReadLine();

string[] words = line.Split(new char[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

if (words.Length == 5 && Convert.ToInt32(words[1]) > 1900 && Convert.ToInt32(words[1]) < 2021 && Regex.IsMatch(words[2], @"(0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20)\d\d"))

{

collectionPatients.Patients.Push(new Patient { LastName = words[0], YearOfBirth = Convert.ToInt32(words[1]), DateOfAdmission = Convert.ToDateTime(words[2]), LengthOfTreatment = Convert.ToInt32(words[3]), Diagnosis = (Diseases)Convert.ToInt32(words[4]) });

}

else

{

throw new Exception("The data in the file is invalid.");

}

}

reader.Close();

}

catch (Exception e)

{

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

return;

}

}

private static void InputDataNewPatient(Collection<Patient> collectionPatients)

{

Console.WriteLine("Enter new patient details");

Console.Write("Enter the patient's last name: ");

string inputLastName = Console.ReadLine();

Console.Write("Enter patient's year of birth: ");

string input = Console.ReadLine();

int inputYearOfBirth;

while (!Int32.TryParse(input, out inputYearOfBirth) || inputYearOfBirth < 1900 || inputYearOfBirth > 2021)

{

Console.Write("Wrong input! Enter patient's year of birth:");

input = Console.ReadLine();

}

Console.Write("Enter the patient's admission date(For example: 30/12/2020): ");

input = Console.ReadLine();

while (!Regex.IsMatch(input, @"(0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.](19|20)\d\d"))

{

Console.Write("Wrong input! Enter the patient's admission date:");

input = Console.ReadLine();

}

DateTime inputDateOfAdmission = Convert.ToDateTime(input);

Console.Write("Enter the duration of treatment in days:");

input = Console.ReadLine();

int inputLengthOfTreatment;

while (!Int32.TryParse(input, out inputLengthOfTreatment) || inputLengthOfTreatment <= 0)

{

Console.Write("Wrong input! Enter the duration of treatment in days: ");

}

Console.Write("Enter disease code: ");

int numberDiagnosis;

input = Console.ReadLine();

while (!Int32.TryParse(input, out numberDiagnosis) || !System.Enum.IsDefined(typeof(Diseases), numberDiagnosis))

{

Console.Write("Wrong input! Enter disease code: ");

input = Console.ReadLine();

}

Diseases inputDiagnosis = (Diseases)numberDiagnosis;

collectionPatients.Patients.Push(new Patient { LastName = inputLastName, DateOfAdmission = inputDateOfAdmission, YearOfBirth = inputYearOfBirth, Diagnosis = inputDiagnosis, LengthOfTreatment = inputLengthOfTreatment });

}

private static void SelectingSort(Collection<Patient> collection)

{

Console.WriteLine("Select sort type:\n1 - sort by age\n2 - sort by staying");

string input = Console.ReadLine();

int select;

while (!Int32.TryParse(input, out select) || select < 1 || select > 2)

{

Console.Write("Wrong input! Select sort type:\n1 - sort by age\n2 - sort by staying");

input = Console.ReadLine();

}

if (select == 1)

{

OutputInformationInList(collection.OrderByDescending(x => x.Age()).ToList());

}

if (select == 2)

{

List<Patient> SortedPatients = collection.ToList();

SortedPatients.Sort();

OutputInformationInList(SortedPatients);

}

}

private static void OutputInformationInList<T>(List<T> list) where T : struct

{

Table table = new Table();

table.HeaderTable();

table.TableLinesList(list);

table.BottomTable();

}

private static void OutputInformation<T>(Collection<T> collection) where T : struct

{

Table table = new Table();

table.HeaderTable();

table.TableLines(collection);

table.BottomTable();

}

private static bool CriterionUpTo20YearsWithCodeDiagnosis(Patient item) => (item.Age() <= 20 && (int)item.Diagnosis == 145);

}

}

**CodeFile1.cs**

using System;

using System.Globalization;

namespace lab2

{

public enum Diseases

{

Flu = 12,

Angina = 32,

Pneumonia = 145,

Gastritis = 98,

Bronchitis = 67,

Hypertension = 34,

Covid = 777

}

struct Patient : IComparable<Patient>

{

private string lastName;

public string LastName

{

get => lastName;

set => lastName = value;

}

private int yearOfBirth;

public int YearOfBirth

{

get => yearOfBirth;

set => yearOfBirth = value;

}

private DateTime dateOfAdmission;

public DateTime DateOfAdmission

{

get => dateOfAdmission;

set => dateOfAdmission = value;

}

private int lengthOfTreatment;

public int LengthOfTreatment

{

get => lengthOfTreatment;

set => lengthOfTreatment = value;

}

private Diseases diagnosis;

public Diseases Diagnosis

{

get => diagnosis;

set => diagnosis = value;

}

private DateTime DurationOfHospitalStay => dateOfAdmission.AddDays(lengthOfTreatment);

public int Age() => DateTime.Now.Year - DateTime.ParseExact(Convert.ToString(yearOfBirth), "yyyy", CultureInfo.InvariantCulture).Year;

public override string ToString() => $"{lastName,-18} │ {DurationOfHospitalStay,-31} │ {Age(),-14}";

public int CompareTo(Patient obj)

{

if (this.LengthOfTreatment > obj.LengthOfTreatment)

return -1;

if (this.LengthOfTreatment < obj.LengthOfTreatment)

return 1;

return 0;

}

}

}

**Collection.cs**

using System.Collections;

using System.Collections.Generic;

namespace lab2

{

class Collection<T> : IEnumerable<T> where T : struct // ограничение на параметр типа значимый

{

public Stack<T> Patients { get; set; } = new Stack<T>();

public delegate bool SearchParameter(T item);

public List<T> Search(SearchParameter searchParameter)

{

List<T> listSearched = new List<T>();

foreach (var item in Patients)

{

if (searchParameter(item))

{

listSearched.Add(item);

}

}

return listSearched;

}

public IEnumerator<T> GetEnumerator()

{

return Patients.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return this.GetEnumerator();

}

}

}

**Table.cs**

using System;

using System.Collections.Generic;

namespace lab2

{

class Table

{

public void HeaderTable()

{

Console.WriteLine("┌─────────┬────────────────────┬─────────────────────────────────┬────────────────┐");

Console.WriteLine("│ │ Lastname │ date of │ Age │");

Console.WriteLine("│ │ │ discharge │ │ ");

}

public void TableLines<T>(Collection<T> collection) where T : struct

{

int i = 1;

foreach (var item in collection.Patients)

{

{

Console.WriteLine("├─────────┼────────────────────┼─────────────────────────────────┼────────────────┤");

Console.WriteLine($"│ {i,-7} │ {item} │");

i++;

}

}

}

public void TableLinesList<T>(List<T> list) where T : struct

{

int i = 1;

foreach (var item in list)

{

{

Console.WriteLine("├─────────┼────────────────────┼─────────────────────────────────┼────────────────┤");

Console.WriteLine($"│ {i,-7} │ {item} │");

i++;

}

}

}

public void BottomTable()

{

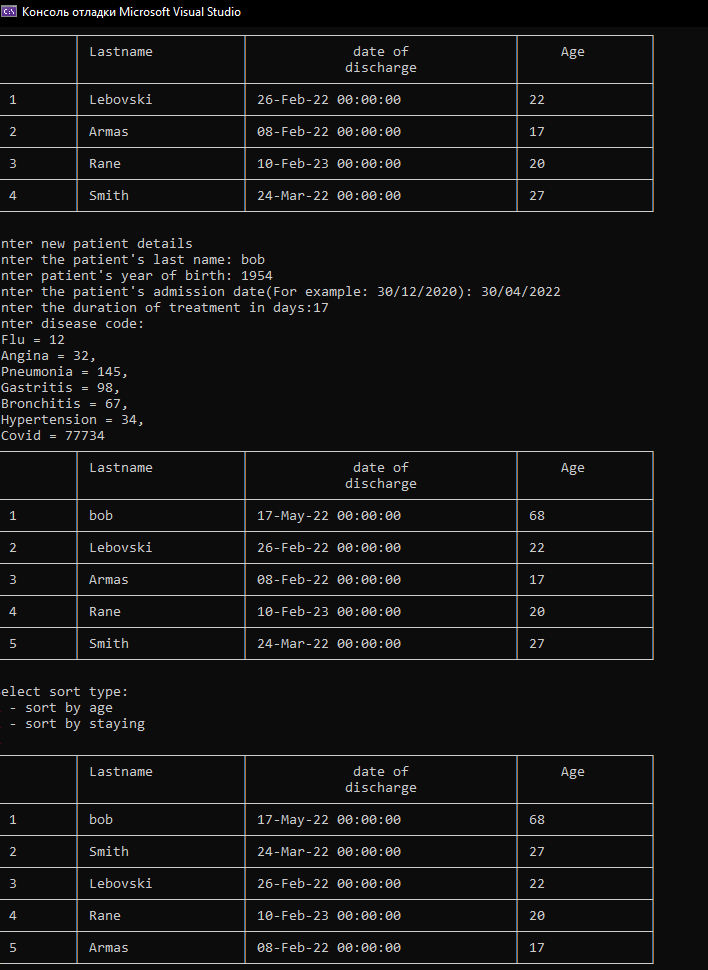
Console.WriteLine("└─────────┴────────────────────┴─────────────────────────────────┴────────────────┘\n");

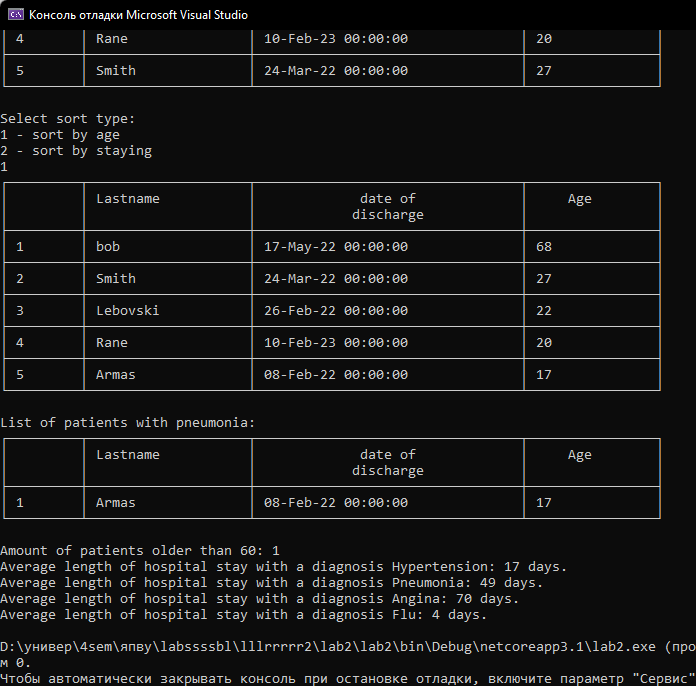
}

}

}

**Результат выполнения программы:**

****



**Вывод:** В ходе лабораторной работы были изучены особенности применения коллекций структур данных и классов прототипов.