# Решения на задачите от глава 14 на книгата "Въведение в програмирането със C#"

Предлагаме ви решения на задачите от [книгата "Въведение в програ­мирането със C#"](http://www.introprogramming.info), заедно с анализ на задачата, описание на използва­ните идеи, алгоритми, подходи за решаване и тестове.

## Авторски колектив

**Момчил Рогелов**

Контакти: [momchilrogelov@gmail.com](mailto:momchilrogelov@gmail.com)

**Станислава Костадинова Богданова**

Контакти: [stanislava.bogdanova@yahoo.com](mailto:stanislava.bogdanova@yahoo.com)

**Милко Карамачев**

Контакти: [milkodk@gmail.com](mailto:milkodk@gmail.com)

# Шаблон за описание на задачите

|  |  |
| --- | --- |
| Задача 1-7. Студенти | |
| **Условие**  Дефинирайте клас **Student**, който съдържа следната информация за студентите: трите имена, курс, специалност, университет, електронна поща и телефонен номер. Декларирайте три конструктора за класа **Student**, които имат различни списъци с параметри (за цялостната информация за даден студент и за част от нея:за лично име и фамилия, и за лично име фамилия и университет ). Данните, за които няма входна информация да се инициализират съответно с **null** или **0**. Добавете статично поле в класа **Student**, в което се съхранява броя на създадените обекти от този клас. Добавете метод в класа **Student**, който извежда пълна информация за студента. Напишете клас **StudentTest**, който да тества функционалността на класа **Student**, Програмата трябва да има следната функционалност:   * **AddStudent(firstName, lastName)-** да добавя данните за студент. Трябва да принтира „**Student added.“.** * **AddStudent(firstName, lastName, university)** – да добавя данните за студент. Трябва да принтира **„Student added.“.** * **AddStudent(firstName, middleName, lastName, course, specialty, university, eMail, phoneNumber)** – да добавя данните за студент. Трябва да принтира **„Student added.“.** * **PrintNumberOfStudents** – да принтира броя на студентите. * **PrintStudentsInfo** – да принтира пълна информация за всеки един от студентите.   Край на въвежданите данни ще е въвеждането на **“End.”.** Изходния файл трябва да се отпечата на конзолата | |
| **Описание на входа**  Входните данни могат да бъдат не повече от 20 реда. Всеки ред ще започва с команда (без разделителни интервали преди или след командите) **AddStudent, PrintNumberOfStudents,** или **PrintStudentsInfo** изписани точно по този начин. След командата без разделящ интервал ще има кръгли скоби и в тях ще бъдат параметрите необходими за изпълнението на командата, отделени един от друг със запетая и интервал ( параметрите винаги ще бъдат зададени в правилния формат). | |
| **Описание на изхода**  Изходните данни трябва да съдържат резултата на всяка една входна операция (изписан по начина, който е даден в условието на задачата ) отпечатан на нов ред като започват от началото на реда без интервал. При извеждане на информация за повече от един студент, то студентите трябва да се подредят точно в реда в който са записани. | |
| **Анализ на задачата**  В тази задача няма нищо за измисляне. Тя не е алгоритмична и в нея няма какво толкова да мислим. Трябва за обекта студент от описанието в условието на задачата да дефинираме един клас и след това в този клас да дефинираме свойства, които го описват и да прибавим една статична променлива, която да брой добавените студенти. Това е всичко. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Text;  namespace TestStudents  {  class Student  {  static int numberOfStudents = 0;  private string firstName;  private string middleName;  private string lastName;  private int course;  private string specialty;  private string university;  private string eMail;  private string phoneNumber;  public Student(string firstName, string middleName, string lastName, int course, string specialty,  string university, string eMail, string phoneNumber)  {  this.firstName = firstName;  this.middleName = middleName;  this.lastName = lastName;  this.course = course;  this.specialty = specialty;  this.university = university;  this.eMail = eMail;  this.phoneNumber = phoneNumber;  numberOfStudents++;  }  public Student(string firstName, string lastName)  : this(firstName, null, lastName, 0, null, null, null, null)  {  }  public Student(string firstName, string lastName, string university)  : this(firstName, null, lastName, 0, null, university, null, null)  {  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append("First Name: " + firstName);  sb.Append(System.Environment.NewLine);  sb.Append("Middle Name: " + middleName);  sb.Append(System.Environment.NewLine);  sb.Append("Last Name: " + lastName);  sb.Append(System.Environment.NewLine);  sb.Append("Course: " + course);  sb.Append(System.Environment.NewLine);  sb.Append("Specialty: " + specialty);  sb.Append(System.Environment.NewLine);  sb.Append("University: " + university);  sb.Append(System.Environment.NewLine);  sb.Append("eMail: " + eMail);  sb.Append(System.Environment.NewLine);  sb.Append("Phone Number: " + phoneNumber);  return sb.ToString();  }  public static int NumberOfStudents  {  get { return Student.numberOfStudents; }  set { Student.numberOfStudents = value; }  }  }  class Program  {  static List<Student> students = new List<Student>();  static StringBuilder result = new StringBuilder();  static void Main(string[] args)  {  StartTesting();  PrintOutput();  }  static void StartTesting()  {  string line = Console.ReadLine();  while (line != "End.")  {  string[] separators = { ",", "(", ")" };  string[] expectedComand = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < expectedComand.Length; i++)  {  expectedComand[i] = expectedComand[i].Trim();  }  ComandExecutes(expectedComand);  line = Console.ReadLine();  }  }  private static void ComandExecutes(string[] expectedComand)  {  switch (expectedComand[0])  {  case "AddStudent":  {  if (expectedComand.Length == 3)  {  students.Add(new Student(expectedComand[1], expectedComand[2]));  result.Append("Student added." + System.Environment.NewLine);  }  if (expectedComand.Length == 4)  {  students.Add(new Student(expectedComand[1], expectedComand[2], expectedComand[3]));  result.Append("Student added." + System.Environment.NewLine);  }  if (expectedComand.Length == 10)  {  students.Add(new Student(expectedComand[1], expectedComand[2], expectedComand[3],  int.Parse(expectedComand[4]), expectedComand[5],  expectedComand[6], expectedComand[7], expectedComand[8]));  result.Append("Student added." + System.Environment.NewLine);  }  break;  }  case "PrintNumberOfStudents":  {  result.Append(Student.NumberOfStudents + System.Environment.NewLine);  break;  }  case "PrintStudentsInfo":  {  foreach (var item in students)  {  result.Append(item.ToString() + System.Environment.NewLine);  }  break;  }  default:  {  result.Append("Invalid Comand.");  break;  }  }  }  static void ReadInput()  {  string line = Console.ReadLine();  while (line != "End.")  {  string[] separators = { ",", "(", ")" };  string[] expectedComand = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);  line = Console.ReadLine();  }  }  static void PrintOutput()  {  Console.WriteLine(result);  }  }  } | |
| **Тестове**  Случаите за тестване са следните:   * Добавяне на един ученик. * Добавяне на няколко ученика. * Принтиране броя на учениците. * Принтиране на данните на учениците | |
| **Вход** | **Изход** |
| AddStudent(Aristotel, Aristotelov)  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Chuk, Norise, GodUniversity)  End. | Student added.  Student added.  Student added.  3  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  Student added. |
| **Вход** | **Изход** |
| AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  End. | Student added.  1  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris |
| **Вход** | **Изход** |
| AddStudent(Aristotel, Aristotelov)  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  AddStudent(Chuk, Norise, GodUniversity)  PrintNumberOfStudents  End. | Student added.  Student added.  Student added.  3  Student added.  4 |
| **Вход** | **Изход** |
| AddStudent(Aristotel, Aristotelov)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  End. | Student added.  1  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  Student added.  2  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  Student added.  3  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris |
| **Вход** | **Изход** |
| AddStudent(Cpiran, Planeta)  AddStudent(Kalinka, Malinka, FlowerUniversity)  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Chuk, Norise, GodUniversity)  End. | Student added.  Student added.  Student added.  3  First Name: Cpiran  Middle Name:  Last Name: Planeta  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Kalinka  Middle Name:  Last Name: Malinka  Course: 0  Specialty:  University: FlowerUniversity  eMail:  Phone Number:  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  Student added. |
| **Вход** | **Изход** |
| AddStudent(Aristotel, Aristotelov)  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Chuk, Norise, GodUniversity)  PrintNumberOfStudents  PrintNumberOfStudents  End. | Student added.  Student added.  Student added.  3  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  Student added.  4  4 |
| **Вход** | **Изход** |
| PrintNumberOfStudents  End. | 0 |
| **Вход** | **Изход** |
| End. | [празен стринг] |
| **Вход** | **Изход** |
| AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Aristotel, Aristotelov)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  PrintNumberOfStudents  PrintStudentsInfo  End. | Student added.  1  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  Student added.  2  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  Student added.  3  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  Student added.  4  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  First Name: Pesho  Middle Name: Peshov  Last Name: Peshov  Course: 3  Specialty: C#  University: Telerik  eMail: pesho@chuk  Phone Number: noris |
| **Вход** | **Изход** |
| AddStudent(Aristotel, Aristotelov)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Hipokrat, Hipokratov, MedicalUniversity)  PrintNumberOfStudents  PrintStudentsInfo  AddStudent(Pesho, Peshov,Peshov,3, C#, Telerik, pesho@chuk,noris,223234)  End. | Student added.  1  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  Student added.  2  First Name: Aristotel  Middle Name:  Last Name: Aristotelov  Course: 0  Specialty:  University:  eMail:  Phone Number:  First Name: Hipokrat  Middle Name:  Last Name: Hipokratov  Course: 0  Specialty:  University: MedicalUniversity  eMail:  Phone Number:  Student added. |

|  |  |
| --- | --- |
| Задача 8-13. GSM Class | |
| **Условие**  Дефинирайте клас, който съдържа информация за мобилен телефон: модел, производител, цена, собственик, характеристики на батерията (модел, idle time и часове разговор /hourstalk/) и характеристики на екрана (големина и цветове).  Декларирайте няколко конструктора за всеки от създадените класове от предходната задача, които имат различни списъци с параметри (за цялостната информация за даден студент или част от нея). Данните за полетата, които не са известни трябва да се инициализират съответно със стойности с **null** или **0**.  Към класа за мобилен телефон добавете статично поле **NokiaN95**, което да съхранява информация за мобилен телефон моделNokia 95. Добавете метод, в същия клас, който извежда информация за това статично поле.  Добавете изброим тип **BatteryType**, който съдържа стойности за тип на батерията (Li-Ion, NiMH, NiCd, …) и го използвайте като ново поле за класа **Battery**.  Добавете метод в класа **GSM**, който да връща информация за обекта под формата на **string**.  Дефинирайте свойства, за да капсулирате данните в класовете **GSM**, **Battery** и **Display**.  Nokia95: model: nokia95, manufacturer: nokia, price: 199, battery: hoursIdle: 12  , hoursTalk: 123, batteryModel: coolBatModel, , display: displaySize: 7, display  Colors: 100 | |
| **Описание на входа**  Първия ред от входа е число N – броят на моделите, които ще запаметим.  На следващите N реда получаваме информация за информация за моделите. Входът може да бъде в 3 формата.  Model;Producer;Price;Owner  или  Model;Producer;Price;Owner;BatteryModel;BatteryIdleTime;BatteryHoursTalk;ScreenColors;ScreenSize  или  Nokia95  Пример:  Galaxy;Samsung;200;Lucho;HugeBat;200;20;256;7 | |
| **Описание на изхода**  За всеки един от телефоните изкарайте информацията във следния формат:  model: Galaxy, manufacturer: Samsung, price: 200, owner: Lucho, battery: hoursIdle: 200, hoursTalk: 20, batteryModel: HugeBat, , display: displaySize: 256, displayColors: 7 | |
| **Анализ на задачата**  Задачата е техническа и няма нужда от анализ. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace GSMProblem  {  class Program  {  static void Main(string[] args)  {  int n = int.Parse(Console.ReadLine());  List<GSM> phones = new List<GSM>();  for (int i = 0; i < n; i++)  {  string line = Console.ReadLine();  if (line=="Nokia95")  {  phones.Add(GSM.NokiaN95);  }  else  {  string[] arguments = line.Split(';');  if (arguments.Count()==4)  {  GSM newPhone = new GSM(arguments[0], arguments[1], double.Parse(arguments[2]), arguments[3]);  phones.Add(newPhone);  }  else  {  GSM newPhone = new GSM(arguments[0], arguments[1], double.Parse(arguments[2]), arguments[3],arguments[4],int.Parse(arguments[5]),int.Parse(arguments[6]),double.Parse(arguments[7]),int.Parse(arguments[8]));  phones.Add(newPhone);  }  }  }  foreach (GSM phone in phones)  {  Console.WriteLine(phone);  }  }  }  public class GSM  {  private string model;  private string manufacturer;  private double? price;  private string owner;  private Battery battery;  private Display display;    public static GSM NokiaN95  {  get  {  return new GSM("nokia95","nokia",199.0,"","coolBatModel",12,123,7.0,100);  }  }  public Display Display  {  get  {  return this.display;  }  set  {  this.display = value;  }  }  public Battery Battery  {  get  {  return this.battery;  }  set  {  this.battery = value;  }  }  public string Owner  {  get  {  return this.owner;  }  set  {  this.owner = value;  }  }  public double? Price  {  get  {  return this.price;  }  set  {  this.price = value;  }  }  public string Manufacturer  {  get  {  return this.manufacturer;  }  set  {  this.manufacturer = value;  }  }  public string Model  {  get  {  return this.model;  }  set  {  this.model = value;  }  }  //constructors  public GSM(string model, string manufacturer, double price, string owner)  {  this.model = model;  this.manufacturer = manufacturer;  this.price = price;  this.owner = owner;  this.battery = new Battery();  this.display = new Display();  }  public GSM(string model, string manufacturer, double price, string owner, string batteryModel, int hoursIdle, int hoursTalk, double displaySize, int displayColors)  {  this.model = model;  this.manufacturer = manufacturer;  this.price = price;  this.owner = owner;  this.battery = new Battery(batteryModel, hoursIdle, hoursTalk);  this.display = new Display(displaySize, displayColors);  }  public override string ToString()  {  return string.Format("model: {0}, manufacturer: {1}, price: {2}, owner: {3}, battery: {4}, , display: {5}", model, manufacturer, price, owner, battery, display);  }  }  public class Battery  {  private int? hoursIdle;  private int? hoursTalk;  private string batteryModel;  private BatteryType battery;  enum BatteryType { LiIon, NiMH, NiCD };  public Battery()  {  // TODO: Complete member initialization  this.batteryModel = null;  this.hoursIdle = null;  this.hoursTalk = null;  this.battery = BatteryType.LiIon;  }  public Battery(string batteryModel, int hoursIdle, int hoursTalk)  {  // TODO: Complete member initialization  this.batteryModel = batteryModel;  this.hoursIdle = hoursIdle;  this.hoursTalk = hoursTalk;  }  public override string ToString()  {  return string.Format("hoursIdle: {0}, hoursTalk: {1}, batteryModel: {2}", hoursIdle, hoursTalk, batteryModel);  }  }  public class Display  {  private double? displaySize;  private int? displayColors;  public Display()  {  this.displaySize = null;  this.displayColors = null;  }  public Display(double displaySize, int displayColors)  {  // TODO: Complete member initialization  this.displaySize = displaySize;  this.displayColors = displayColors;  }  public override string ToString()  {  return string.Format("displaySize: {0}, displayColors: {1}", displaySize, displayColors);  }  }  } | |
| **Тестове**   * Nokia * Кратка дефиниция * Пълна дефиниция * Няколко дефиниции | |
| **Вход** | **Изход** |
| 1  Nokia95 | model: nokia95, manufacturer: nokia, price: 199, owner: , battery: hoursIdle: 12, hoursTalk: 123, batteryModel: coolBatModel, , display: displaySize: 7, displayColors: 100 |
| 1  Galaxy II;Samsung;250;Lucho;HugeBat;200;20;256;7 | 1  model: Galaxy II, manufacturer: Samsung, price: 250, owner: Lucho, battery: hoursIdle: 200, hoursTalk: 20, batteryModel: HugeBat, , display: displaySize: 256, displayColors: 7 |
| 1  Galaxy II;Samsung;250;Lucho | model: Galaxy II, manufacturer: Samsung, price: 250, owner: Lucho, battery: hoursIdle: , hoursTalk: , batteryModel: , , display: displaySize: , displayColors: |
| 3  Nokia95  Galaxy II;Samsung;250;Lucho;HugeBat;200;20;256;7  Galaxy II;Samsung;250;Lucho | model: nokia95, manufacturer: nokia, price: 199, owner: , battery: hoursIdle: 12, hoursTalk: 123, batteryModel: coolBatModel, , display: displaySize: 7, displayColors: 100  model: Galaxy II, manufacturer: Samsung, price: 250, owner: Lucho, battery: hoursIdle: 200, hoursTalk: 20, batteryModel: HugeBat, , display: displaySize: 256, displayColors: 7  model: Galaxy II, manufacturer: Samsung, price: 250, owner: Lucho, battery: hoursIdle: , hoursTalk: , batteryModel: , , display: displaySize: , displayColors: |

|  |  |
| --- | --- |
| Задача 14-19. GSM Call History | |
| **Условие**  Създайте клас **Call**, който съдържа информация за разговор, осъщес­твен през мобилен телефон. Той трябва да съдържа информация за датата, времето на започване и продължителността на разговора.  Добавете свойство архив с обажданията – **callHistory**, което да пази списък от осъщест­вените разговори.  В класа**GSM** добавете методи за добавяне и изтриване на обаждания (**Call**) в архива с обаждания на мобилния телефон. Добавете метод, който изтрива всички обаждания от архива.  В класа **GSM** добавете метод, който пресмята общата сума на обажда­нията (**Call**) от архива с обаждания на телефона (**callHistory**) като нека цената за едно обаждане се подава като параметър на метода. Запишете няколко обаждания, изчислете сумата за обажданията, премахнете най-скъпото обаждане и изчислете отново сумата.  Програмата трябва да изпълнява следните операции:   * **AddCall(date, time, duration)** - да добавя нов запис към листа със **callHistory**. Като резултат принтира **„Call added.“** * **DeleteCall** – изтрива всички разговори от **callHistory.** Като резултат принтира **„Call history deleted.“** * **CalculateSum(price)** – пресмята цената на всички разговори по зададена цена на минута. * **DeleteMostExpensiveCall** – изтрива най – скъпия разговор от листа **callHistory.** Принтира „**Most expensive call deleted.“.** * **PrintCallHistory** – принтира всички разговори в реда на тяхното въвеждане. В случай, че няма записани разговори принтира **„No calls.“.**   Край на въвежданите данни ще е въвеждането на **“End.”** ред. Изходния файл трябва да се отпечата на конзолата. | |
| **Описание на входа**  Входните данни могат да бъдат не повече от 20 реда. Всеки ред ще започва с команда (без разделителни интервали преди или след командите) **AddCall**(date, time, duration), **DeleteCall**, **CalculateSum**(**price**), **DeleteMostExpensiveCall** или **PrintCallHistory** изписани точно по този начин. След командата без разделящ интервал ще има кръгли скоби и в тях ще бъдат параметрите необходими за изпълнението на командата, отделени един от друг със запетая и интервал(параметрите винаги ще бъдат зададени в правилния формат.  Параметрите на метода **AddCall** ще са във формата 12/03/2009 13:39 5  Където последната цифра е времетраенето на разговора. | |
| **Описание на изхода**  Изходните данни трябва да съдържат резултата на всяка една входна операция (изписан по начина, който е даден в условието на задачата)отпечатан на нов ред като започват от началото на реда без интервал. В случаи, че някоя от командите трябва да принтира повече от един отговор те трябва да се подредят в реда в които са въведени от входа на програмата. | |
| **Анализ на задачата**  Тя не е алгоритмична и в нея няма какво толкова да мислим. Трябва да направим лист от тип **Call** и в него да записваме разговорите. Трябва да напишем методи за добавяне в листа, изтриване на всички елементи в листа, изтриване на най- скъпия разговор, пресмятане на стойността на всички разговори и принтиране на всички разговори в реда на записването им. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Pr.\_14\_19.GSMCallStory  {  class Program  {  static StringBuilder result = new StringBuilder();  static GSM ourPhone = GSM.NokiaN95;  static void Main(string[] args)  {  ReadInput();  PrintOutput();  }  static void ReadInput()  {  string line = Console.ReadLine();  while (line != "End.")  {  string[] separators = { ",", "(", ")", " " };  string[] arguments = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < arguments.Length; i++)  {  arguments[i] = arguments[i].Trim();  }  ComandExecutes(arguments);  line = Console.ReadLine();  }  }  private static void ComandExecutes(string[] arguments)  {  switch (arguments[0])  {  case "AddCall":  {  ourPhone.AddCall(arguments[1], arguments[2], int.Parse(arguments[3]));  result.Append("Call added." + System.Environment.NewLine);    break;  }  case "DeleteCall":  {  ourPhone.ClearCallHistory();  result.Append("Call history deleted." + System.Environment.NewLine);  break;  }  case "CalculateSum":  {  result.Append(ourPhone.CalculateBill(double.Parse(arguments[1])).ToString() + System.Environment.NewLine);  break;  }  case "DeleteMostExpensiveCall":  {  List<Call> calls = ourPhone.CallHistory;  Call maxCall = calls.Max();  calls.Remove(maxCall);  result.Append("Most expensive call deleted." + System.Environment.NewLine);  break;  }  case "PrintCallHistory":  {  if(ourPhone.CallHistory.Count>0)  {  foreach (var item in ourPhone.CallHistory)  {  result.Append(item.ToString() + System.Environment.NewLine);  }  }  else  {  result.Append("No calls." + System.Environment.NewLine);  }  break;  }  default:  {  result.Append("Invalid Comand." + System.Environment.NewLine);  break;  }  }  }  static void PrintOutput()  {  Console.WriteLine(result);  }  }  public class GSM  {  private string model;  private string manufacturer;  private double? price;  private string owner;  private Battery battery;  private Display display;  private List<Call> callHistory = new List<Call>();  public List<Call> CallHistory  {  get { return callHistory; }  set { callHistory = value; }  }  public static GSM NokiaN95  {  get  {  return new GSM("nokia95", "nokia", 199.0, "", "coolBatModel", 12, 123, 7.0, 100);  }  }  public Display Display  {  get  {  return this.display;  }  set  {  this.display = value;  }  }  public Battery Battery  {  get  {  return this.battery;  }  set  {  this.battery = value;  }  }  public string Owner  {  get  {  return this.owner;  }  set  {  this.owner = value;  }  }  public double? Price  {  get  {  return this.price;  }  set  {  this.price = value;  }  }  public string Manufacturer  {  get  {  return this.manufacturer;  }  set  {  this.manufacturer = value;  }  }  public string Model  {  get  {  return this.model;  }  set  {  this.model = value;  }  }  //constructors  public GSM(string model, string manufacturer, double price, string owner)  {  this.model = model;  this.manufacturer = manufacturer;  this.price = price;  this.owner = owner;  this.battery = new Battery();  this.display = new Display();  }  public GSM(string model, string manufacturer, double price, string owner, string batteryModel, int hoursIdle, int hoursTalk, double displaySize, int displayColors)  {  this.model = model;  this.manufacturer = manufacturer;  this.price = price;  this.owner = owner;  this.battery = new Battery(batteryModel, hoursIdle, hoursTalk);  this.display = new Display(displaySize, displayColors);  }  public override string ToString()  {  return string.Format("model: {0}, manufacturer: {1}, price: {2}, owner: {3}, battery: {4}, , display: {5}", model, manufacturer, price, owner, battery, display);  }  public void AddCall(string date, string time, int duration)  {  //Call newCall = new Call(date, time, duration);  callHistory.Add(new Call(date, time, duration));  }  public void ClearCallHistory()  {  callHistory.Clear();  }    public void DeleteCall(Call callToRemove)  {  callHistory.Remove(callToRemove);  }  public string CalculateBill(double pricePerMinute)  {  double bill = 0;  foreach (Call item in this.callHistory)  {  bill = bill + item.Duration \* pricePerMinute;  }  return String.Format("{0:0.00}", bill);  }  }  public class Display  {  private double? displaySize;  private int? displayColors;  public Display()  {  this.displaySize = null;  this.displayColors = null;  }  public Display(double displaySize, int displayColors)  {  // TODO: Complete member initialization  this.displaySize = displaySize;  this.displayColors = displayColors;  }  public override string ToString()  {  return string.Format("displaySize: {0}, displayColors: {1}", displaySize, displayColors);  }  }  public class Call : IComparable  {  private string date;  private string time;  private int duration;  public Call(string date, string time, int duration)  {  this.date = date;  this.time = time;  this.duration = duration;  }  public int Duration  {  get  {  return this.duration;  }  set  {  this.duration = value;  }  }  public int CompareTo(object obj)  {  Call toBeComapredTo = obj as Call;  if (this.Duration > toBeComapredTo.Duration)  {  return 1;  }  else  {  return -1;  }  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append(date + " ");  sb.Append(time.ToString() + " ");  sb.Append(duration );  return sb.ToString();  }  }  public class Battery  {  private int? hoursIdle;  private int? hoursTalk;  private string batteryModel;  private BatteryType battery;  enum BatteryType { LiIon, NiMH, NiCD };  public Battery()  {  // TODO: Complete member initialization  this.batteryModel = null;  this.hoursIdle = null;  this.hoursTalk = null;  this.battery = BatteryType.LiIon;  }  public Battery(string batteryModel, int hoursIdle, int hoursTalk)  {  // TODO: Complete member initialization  this.batteryModel = batteryModel;  this.hoursIdle = hoursIdle;  this.hoursTalk = hoursTalk;  }  public override string ToString()  {  return string.Format("hoursIdle: {0}, hoursTalk: {1}, batteryModel: {2}", hoursIdle, hoursTalk, batteryModel);  }  }  } | |
| **Тестове**   * Записване на разговори. * Принтиране на листа със записани разговори. * Изтриване на всички разговори. * Пресмятане на стойността на всички разговори. * Изтриване на най – скъпия разговор. | |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 12:05 20)  AddCall(12/03/2012 12:05 40)  End. | Call added.  Call added.  Call added. |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 12:05 20)  AddCall(12/03/2012 12:05 40)  PrintCallHistory  DeleteCall  PrintCallHistory  End. | AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 12:05 20)  AddCall(12/03/2012 12:05 40)  PrintCallHistory  DeleteCall  PrintCallHistory  End. |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 14:05 40)  CalculateSum(0.10)  End. | Call added.  Call added.  Call added.  7.00 |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.10)  DeleteMostExpensiveCall  CalculateSum(0.10)  End. | Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00 |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.10)  DeleteMostExpensiveCall  CalculateSum(0.10)  DeleteCall  End. | Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00  Call history deleted. |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.10)  DeleteMostExpensiveCall  CalculateSum(0.10)  PrintCallHistory  DeleteCall  PrintCallHistory  End. | Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00  12/03/2012 12:05 10  12/03/2012 13:05 20  Call history deleted.  No calls. |
| **Вход** | **Изход** |
| PrintCallHistory  End. | No calls. |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.10)  DeleteMostExpensiveCall  CalculateSum(0.10)  PrintCallHistory  DeleteCall  PrintCallHistory  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  End. | Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00  12/03/2012 12:05 10  12/03/2012 13:05 20  Call history deleted.  No calls.Call added.  Call added. |
| **Вход** | **Изход** |
| AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.20)  CalculateSum(0.10)  End. | Call added.  Call added.  12.00  6.00 |
| **Вход** | **Изход** |
| AddCall(12/03/2012 12:05 10)  AddCall(12/03/2012 13:05 20)  AddCall(12/03/2012 15:05 40)  CalculateSum(0.10)  DeleteMostExpensiveCall  CalculateSum(0.10)  PrintCallHistory  DeleteCall  PrintCallHistory  CalculateSum(0.10)  End. | Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00  12/03/2012 12:05 10  12/03/2012 13:05 20  Call history deleted.  No calls.  0.00  Call added.  Call added.  Call added.  7.00  Most expensive call deleted.  3.00  12/03/2012 12:05 10  12/03/2012 13:05 20  Call history deleted.  No calls.  0.00 |

|  |
| --- |
| Задача 20-21. Библиотека |
| **Условие**  Нека е дадена библиотека с книги. Дефинирайте класове съответно за библиотека и книга. Библиотеката трябва да съдържа име и списък от книги. Книгите трябва да съдържат информация за заглавие, автор, издателство, година на издаване и **ISBN**-номер. В класа, който описва библиотека, добавете методи за добавяне на книга към библиотеката, търсене на книга по предварително зададен автор, извеждане на информация за дадена книга и изтриване на книга от библиотеката по дадено заглавие.  Програмата трябва да изпълнява следните операции:   * **AddBook(title, author, publisher, yearOfPublishing, ISBN)** – да добавя книга по зададени заглавие, автор, издателство, година на издаване и **ISBN** – номер. Като резултат трябва да се принтира **„Book added.“**. * **SearchAuthor(author)** – намира книга по предварително зададен автор. Като резултат отпечатва информацията за намерената книга в случай, че намерените книги са повече от 1 се разпечатват в реда на въвеждането им, или **"Not found book of this author."** в случай, че не е намерена книга на дадения автор. * **SearchTitle(title)** – намира книга по дадено заглавие. Като резултат трябва да отпечатва информацията за намерената книга или **"Not found."** в случай, че не е намерена книга с търсеното заглавие. * **DeleteBook(title)** – изтрива от библиотеката книга по дадено заглавие. Като резултат отпечатва **„Book deleted.“** или **„Not found.“**  в случай, че не е намерена книга с търсеното заглавие. * **DeleteAllBooks(author)** – изтрива от библиотеката всички книги по даден автор. Като резултат отпечатва **„n Book deleted.“** (n- броя на изтритите книги)или **„Not found.“**  в случай, че не е намерена книга с търсеното заглавие.   Край на въвежданите данни ще е въвеждането на празен ред. Изходния файл трябва да се отпечата на конзолата. |
| **Описание на входа**  Входните данни могат да бъдат не повече от 10 реда. Всеки ред ще започва с команда (без разделителни интервали преди или след командите) **AddBook, SearchAuthor, SearchTitle** или **DeleteBook** изписани точно по този начин. След командата без разделящ интервал ще има кръгли скоби и в тях ще бъдат параметрите необходими за изпълнението на командата, отделени един от друг със запетая и интервал(параметрите винаги ще бъдат зададени в правилния формат). Заглавията и авторите на книгите, които ще се добавят няма да се дублират т.е. всяка книга ще бъде с уникално заглавие и от автор, който не фигурира в данните на библиотеката. |
| **Описание на изхода**  Изходните данни трябва да съдържат резултата на всяка една входна операция (изписан по начина, който е даден в условието на задачата)отпечатан на нов ред като започват от началото на реда без интервал. Информацията за книга трябва да се изведе на пет последователни реда. На първия ред **“Title: „**и името на книгата изписано по начина по които е въведено от входния файл. На следващия ред: **„Author: “** и след това автора, изписан по начина по които въведен във входния файл. На третия пореден ред **„Publisher: “**, на следващия ред: „Year of **publishing: “**, и на последния ред **„ISBN: “**.  При въвеждане на некоректна команда да се върне резултат **„Unknown operation.“** |
| **Анализ на задачата**  В тази задача няма нищо за измисляне. Тя не е алгоритмична и в нея няма какво толкова да мислим. Трябва за всеки обект от описаните в условието на задачата (библиотека и книга) да дефинираме по един клас и след това в този клас да дефинираме свойства, които го описват и действия, които той може да направи. Това е всичко.  Имплементацията на всеки един от класовете можем да разглеждаме като подзадача на дадената:  -    Клас за книгите – **Book**  -    Клас за библиотеката – **Library**  Удачно е да започнем реализацията с класа **Book**, тъй като от усло­вието на задачата лесно се вижда, че той не зависи от класа **Library**.  Първо дефинираме класа **Book.** В дефиницията имаме полета, представляващи информация заглавие на книгата, автор, издателство, година на издаване и ISBN-номер. Имаме и свойства за всяко от полетата които имат само set тъй, като не ни е нужна функционалност с която да променяме стойностите на полетата.  Следващият клас, който дефинираме е **Library**. В дефиницията му се налага да използваме класа **Book**. Полетата, които ще дефинираме представляват име на библиотеката и списък с книги, които библиотеката съдържа. За реализацията на списъка с книги ще използваме класа **List<Book>**. Класът ще има свойствата **Name** и **Books**, които извличат стойностите на двете полета. Добавяме методите, които ни трябват – **АddBook(…)**, **SearchAuthor(…), SearchTitle(…),** **DeleteBook(…).** Методът **АddBook(…)** добавя обект от тип **Book** към списъка **books**, методът **SearchAuthor(…) и SearchTitle(…)** търсят книга (по даден автор или дадено заглавие) и отпечатват информация за намерената книга в случай, че не намерят книга на този автор или с това заглавие отпечатват съобщение, **DeleteBook(…)**изтрива даден книга по предварително зададено заглавие.  Остава ни да направим в Program няколко метода: за четене от конзолата, за обработване на входните данни и за принтиране на изходните данни.  Първо правим метода за четене на входните данни. Той има един **whie** цикъл, който чете един ред входни данни и след това сплитва входния стринг.  След това правим метода, който изпълнява въведената команда **ExecuteCommand()**. Той получава като параметри името на библиотеката и масива от входни данни. По условие знаем, че в началото на въведения ред е командата, която трябва да се изпълни за това правим **swith - case** блок с четирите възможни метода. При попадане във всеки един от случаите изпълняваме исканата команда като викаме съответния метод от класа **Library**. Всеки метод връща string резултат, който записваме в желания формат в променлива от тип **StringBuilder**.  Програмата завършва като разпечатаме резултатния string. |
| **Решение (сорс код)** |
| using System;  using System.Collections.Generic;  using System.Text;  namespace Ex.\_20And21  {  class Book  {  private string title;  private string author;  private string publisher;  private int yearOfPublishing;  private string ISBN;  public string Title  {  get { return title; }  }  public string Author  {  get { return author; }  }  public string Publisher  {  get { return publisher; }  }  public int YearOfPublishing  {  get { return yearOfPublishing; }  }  public string ISBNNumber  {  get { return ISBN; }  }  public Book(string title, string author, string publisher, int yearOfPublishing, string ISBN)  {  this.title = title;  this.author = author;  this.publisher = publisher;  this.yearOfPublishing = yearOfPublishing;  this.ISBN = ISBN;  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append("Title: " + title);  sb.Append(System.Environment.NewLine);  sb.Append("Author: " + author);  sb.Append(System.Environment.NewLine);  sb.Append("Publisher: " + publisher);  sb.Append(System.Environment.NewLine);  sb.Append("Year of publishing: " + yearOfPublishing);  sb.Append(System.Environment.NewLine);  sb.AppendLine("ISBN: " + ISBN);  return sb.ToString();  }  }  class Library  {  private string name;  private List<Book> books;  public string Name  {  get { return name; }  set { name = value; }  }  internal List<Book> Books  {  get { return books; }  set { books = value; }  }  public Library(string name)  {  this.name = name;  this.books = new List<Book>();  }  public string AddBook(Book book)  {  books.Add(book);  return "Book added.";  }  public string SearchAuthor(string author)  {  StringBuilder listWithSearchedBooks = new StringBuilder();  bool isFoundBook = false;  foreach (Book book in books)  {  if (book.Author.Equals(author))  {  listWithSearchedBooks.Append(book.ToString());  isFoundBook = true;  }  }  if (isFoundBook)  {  return listWithSearchedBooks.ToString();  }  else  {  return "Not found book of this author.";  }  }  public string SearchTitle(string title)  {  foreach (Book book in books)  {  if (book.Title.Equals(title))  {  return book.ToString();  }  }  return "Not found.";  }  public string DeleteBook(Book book)  {  if (books.Contains(book))  {  books.Remove(book);  return "Book deleted.";  }  else  {  return "Book not found.";  }  }  public string DeleteAllBooks(string author)  {  int numberOfBooksBeforeDel = books.Count;  books.RemoveAll(x => x.Author == author);  int numberOfBooksAfterDel = books.Count;  int deletedBooks = numberOfBooksBeforeDel - numberOfBooksAfterDel;  return deletedBooks.ToString()+" Book deleted.";  }  }  class Program  {  static StringBuilder outputString = new StringBuilder();  static void StartTesting()  {  Library helikon = new Library("Helikon");  string row = Console.ReadLine();  while (row != String.Empty)  {  string[] separators = { ",", "(", ")" };  string[] inputLine = row.Split(separators, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < inputLine.Length; i++)  {  inputLine[i] = inputLine[i].Trim();  }  ComandExecutes(inputLine, helikon);  row = Console.ReadLine();  }  }  static void ComandExecutes(string[] inputLine, Library helikon)  {  switch (inputLine[0])  {  case "AddBook":  {  outputString.Append(helikon.AddBook  (new Book(inputLine[1], inputLine[2], inputLine[3], int.Parse(inputLine[4]), inputLine[5])) + System.Environment.NewLine);  break;  }  case "SearchAuthor":  {  outputString.Append(helikon.SearchAuthor(inputLine[1]));  break;  }  case "SearchTitle":  {  outputString.Append(helikon.SearchTitle(inputLine[1]) + System.Environment.NewLine);  break;  }  case "DeleteBook":  {  outputString.Append(helikon.DeleteBook(  helikon.Books.Find(x => x.Title.Equals(inputLine[1]))) + System.Environment.NewLine);  break;  }  case "DeleteAllBooks":  {  outputString.Append(helikon.DeleteAllBooks(inputLine[1]) + System.Environment.NewLine);  break;  }  default:  {  outputString.Append("Unknown operation." + System.Environment.NewLine);  break;  }  }  }  static void PrintOutput()  {  Console.WriteLine(outputString.ToString());  }  static void Main(string[] args)  {  StartTesting();  PrintOutput();  }  }  } |

|  |  |
| --- | --- |
| **Тестове**  Случаите за тестване са следните:   * Добавяне на книга * Търсене на съществуваща книга по зададен автор или по заглавие и отпечатване на информация за нея * Търсене на не съществуваща книга по даден автор или по заглавие * Изтриване на книга която е в библиотеката и изтриване на не съществуваща книга * Въвеждане на не валидна операция. | |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(GreenMile, Stephen King, BulPres, 2004, BG14752534)  AddBook(It2, Stephen King, BulPres, 2003, BG14752534)  AddBook(Carrie, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchAuthor(Stephen King)  DeleteAllBooks(Stephen King)  SearchAuthor(Stephen King) | Book added.  Book added.  Book added.  Book added.  Book added.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  Title: GreenMile  Author: Stephen King  Publisher: BulPres  Year of publishing: 2004  ISBN: BG14752534  Title: It2  Author: Stephen King  Publisher: BulPres  Year of publishing: 2003  ISBN: BG14752534  Title: Carrie  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  4 Book deleted.  Not found book on this author. |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchAuthor(Nakov) | Book added.  Book added.  Book added.  Title: C# For Dummies  Author: Nakov  Publisher: TelerikPres  Year of publishing: 1  ISBN: BG43296745 |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchAuthor(Pesho) | Book added.  Book added.  Book added.  Not found book on this author. |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchTitle(It) | Book added.  Book added.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534 |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchTitle(Itt) | Book added.  Book added.  Book added.  Not found. |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchTitle(It)  DeleteBook(It)  SearchTitle(It) | Book added.  Book added.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  Book deleted  Not found. |
| **Вход** | **Изход** |
| AddBooks(It, Stephen King, BulPres, 2001, BG14752534) | Unknown operation. |
| **Вход** | **Изход** |
| DeleteBook(It) | Book not found. |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchAuthor(Stephen King)  DeleteBook(It)  SearchAuthor(Stephen King)  SearchTitle(It)  DeleteBook(It)  ADDBOOK(It, Stephen King, BulPres, 2001, BG14752534)  SearchTitle(Ittttt)  SearchAuthor(STEPHEN KING) | Book added.  Book added.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  Book deleted.  Not found book on this author.  Not found.  Book not found.  Unknown operation.  Not found.  Not found book on this author. |
| **Вход** | **Изход** |
| AddBook(It, Stephen King, BulPres, 2001, BG14752534)  AddBook(C# For Dummies, Nakov, TelerikPres, 0001, BG43296745)  AddBook(The Snow Queen, Andersen, FeryPres, 1800, BG86476944)  SearchAuthor(Stephen King)  SearchAuthor(Nakov)  SearchAuthor(Andersen)  DeleteBook(It)  SearchAuthor(Stephen King)  SearchTitle(It)  DeleteBook(It)  AddBook(It, Stephen King, BulPres, 2001, BG14752534)  SearchAuthor(Stephen King)  SearchTitle(It) | Book added.  Book added.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  Title: C# For Dummies  Author: Nakov  Publisher: TelerikPres  Year of publishing: 1  ISBN: BG43296745  Title: The Snow Queen  Author: Andersen  Publisher: FeryPres  Year of publishing: 1800  ISBN: BG86476944  Book deleted.  Not found book on this author.  Not found.  Book not found.  Book added.  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534  Title: It  Author: Stephen King  Publisher: BulPres  Year of publishing: 2001  ISBN: BG14752534 |

|  |  |
| --- | --- |
| Задача 22. Училище | |
| **Условие**  Дадено ни е училище. В училището имаме класове и ученици. Всеки преподавател има мно­жест­во от дисциплини, по които преподава. Учениците имат име и уникален номер на класа. Класовете имат уникален текстов иден­тификатор. Дисциплините имат име, брой уроци и брой упражне­ния.  Задачата е да се моделира училище с C# класове. Трябва да декларирате класове заедно с техните полета, свойства, методи и конструктори. Дефинирайте и тестов клас, който демонстрира, че останалите класове работят коректно.  Програмата трябва да изпълнява следните операции:   * **AddStudent(name, classId)-** добавя ученик по зададено име и клас. Като резултат принтира **„Student added.“.** * **AddTeacher(name)** –добавя учител по зададено име.Като резултат принтира **“Teacher added.”.** * **AddDiscipline(nameOfTeacher, nameOfDiscipline,NumberOfLessons, NumberOfExercise)** – добавя дисциплина към името на даден преподавател.Като резултат принтира **„Discipline added.“.** * **PrintSudents(classId)** – принтира всички ученици по зададен клас. В случай, че няма ученици в търсения клас принтира **„No students.“.** * **PrintTeachers(Discipline)** – принтира всички учители, които преподават дадена дисциплина.В случай, че не е въведен учител принтира: **„No teacher.“.**   Край на въвежданите данни ще е въвеждането на **“End.”** ред. Изходния файл трябва да се отпечата на конзолата. | |
| **Описание на входа**  Входните данни могат да бъдат не повече от 20 реда. Всеки ред ще започва с команда (без разделителни интервали преди или след командите) **AddStudent, AddTeacher, AddDiscipline PrintSudents, PrintSudents** или **PrintTeachers** изписани точно по този начин. След командата без разделящ интервал ще има кръгли скоби и в тях ще бъдат параметрите необходими за изпълнението на командата, отделени един от друг със запетая и интервал(параметрите винаги ще бъдат зададени в правилния формат. | |
| **Описание на изхода**  Изходните данни трябва да съдържат резултата на всяка една входна операция (изписан по начина, който е даден в условието на задачата)отпечатан на нов ред като започват от началото на реда без интервал. В случаи, че някоя от командите трябва да принтира повече от един отговор те трябва да се подредят в реда в които са въведени от входа на програмата. | |
| **Анализ на задачата**  В тази задача няма нищо за измисляне. Тя не е алгоритмична и в нея няма какво толкова да мислим. Трябва за всеки обект от описаните в условието на задачата (учител,ученик и дисциплина) да дефинираме по един клас и след това в този клас да дефинираме свойства, които го описват и действия, които той може да направи. Това е всичко.  Имплементацията на всеки един от класовете можем да разглеждаме като подзадача на дадената:  -    Клас за дисциплините – **Discipline**  -    Клас за учителите – **Teacher**  -    Клас за студентите – **Student**  Удачно е да започнем реализацията с класа **Discipline**, тъй като от усло­вието на задачата лесно се вижда, че той не зависи от друг клас. В него имаме три полета: име, брой уроци и брой упражнения. Имаме и едно свойство на полето име,конструктор и сме пренаписали метода **ToString()** за да принтира дисциплините в желания от нас ред.  След това създаваме класа **Teacher**, той има две полета: име и списък от тип **Discipline.** Имаме конструктор и метод, който добавя дисциплини към даден учител.  Следва да реализираме класа **Student**. Той има две полета: **name** и **classId**. Има и конструктор и свойства.  След това реализираме класа School,който обединява трите класа. Той има лист в който пазим учениците и речник в който пазим учителите. В класа **School** имаме два метода единия добавя студенти към листа със студенти, а другия добавя учители към колекцията с учители.  Остава само са оформим входа на програмата. Правим метод **ReadInput**, който чете ред по ред входните докато срещне стринга „**End**.“ Команди докато срещне стринга **„End.“** И сплитва входящия стринг. След това го подава на друг метод, които изпълнява командата и записва резултата в една променлива от тип **stringbilder**. Програмата приключва когато разпечатаме резултатния **stringbilder**. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Text;  namespace School  {  class Discipline  {  string name;  int numberOfLessons;  int numberOfEx;  public Discipline(string name, int numberOfLessons, int numberOfEx)  {  this.name = name;  this.numberOfLessons = numberOfLessons;  this.numberOfEx = numberOfEx;  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append(name + ", ");  sb.Append(numberOfLessons + ", ");  sb.Append(numberOfEx + ", ");  return sb.ToString();  }  public string Name  {  get { return name; }  }  }  class Teacher  {  string name;  List<Discipline> listOfDiscipline = new List<Discipline>();  public void AddDiscipline(string name, int numberOfLessons, int numberOfEx)  {  this.listOfDiscipline.Add(new Discipline(name, numberOfLessons, numberOfEx));  }  public Teacher(string name)  {  this.name = name;  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append("Teacher name:" + name);  foreach (var item in listOfDiscipline)  {  sb.Append(", " + item.ToString());  }  return sb.ToString();  }  public string Name  {  get { return name; }  set { name = value; }  }  internal List<Discipline> ListOfDiscipline  {  get { return listOfDiscipline; }  set { listOfDiscipline = value; }  }  }  class School  {  private static List<Student> listOfStudents = new List<Student>();  private static Dictionary<string, Teacher> listOfTeachers = new Dictionary<string, Teacher>();  public void AddTeacher(Teacher teacher)  {  listOfTeachers.Add(teacher.Name, teacher);  }  public void AddStudent(Student student)  {  listOfStudents.Add(student);  }  internal List<Student> ListOfStudents  {  get { return listOfStudents; }  set { listOfStudents = value; }  }  public static Dictionary<string, Teacher> ListOfTeachers  {  get { return School.listOfTeachers; }  set { School.listOfTeachers = value; }  }  public static List<Student> ListOfStudents1  {  get { return School.listOfStudents; }  set { School.listOfStudents = value; }  }  }  class Student  {  string name;  string classId;  public Student(string name, string classId)  {  this.name = name;  this.classId = classId;  }  public string Name  {  get { return name; }  set { name = value; }  }  public string ClassId  {  get { return classId; }  set { classId = value; }  }  }  class Program  {  static StringBuilder result = new StringBuilder();  static void Main(string[] args)  {  ReadInput();  Console.WriteLine(result.ToString());  }  public static void ReadInput()  {  string line;  line = Console.ReadLine();  while (line != "End.")  {  string[] separators = { ",", "(", ")" };  string[] expectedComand = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < expectedComand.Length; i++)  {  expectedComand[i] = expectedComand[i].Trim();  }  ComandExecutes(expectedComand);  line = Console.ReadLine();  }  }  private static void ComandExecutes(string[] expectedComand)  {  switch (expectedComand[0])  {  case "AddStudent":  {  School.ListOfStudents1.Add(new Student(expectedComand[1], expectedComand[2]));  result.Append("Student added." + System.Environment.NewLine);  return;  }  case "AddTeacher":  {  School.ListOfTeachers.Add(expectedComand[1], new Teacher(expectedComand[1]));  result.Append("Teacher added." + System.Environment.NewLine);  return;  }  case "AddDiscipline":  {  if (School.ListOfTeachers.ContainsKey(expectedComand[1]))  {  School.ListOfTeachers[expectedComand[1]].AddDiscipline(expectedComand[2],  int.Parse(expectedComand[3]), int.Parse(expectedComand[4]));  }  else  {  School.ListOfTeachers.Add(expectedComand[1], new Teacher(expectedComand[1]));  School.ListOfTeachers[expectedComand[1]].AddDiscipline(expectedComand[2],  int.Parse(expectedComand[3]), int.Parse(expectedComand[4]));  }  result.Append("Discipline added." + System.Environment.NewLine);  return;  }  case "PrintStudents":  {  bool isStudentInThisClass = false;  foreach (var item in School.ListOfStudents1)  {  if (item.ClassId.Equals(expectedComand[1]))  {  isStudentInThisClass = true;  result.Append(item.Name + System.Environment.NewLine);  }  }  if (!isStudentInThisClass)  {  result.Append("No students." + System.Environment.NewLine);  }  return;  }  case "PrintTeacher":  {  bool isFoundTeacher = false;  foreach (var teacher in School.ListOfTeachers)  {  for (int i = 0; i < teacher.Value.ListOfDiscipline.Count; i++)  {  if (teacher.Value.ListOfDiscipline[i].Name.Equals(expectedComand[1]))  {  isFoundTeacher = true;  result.Append(teacher.Value + System.Environment.NewLine);  }  }  }  if (!isFoundTeacher)  {  result.Append("No teacher.");  }  break;  }  default:  {  result.Append("Invalid comand");  return;  }  }  }  }  } | |
| **Тестове**  Случаите за тестване са следните:   * Добавяне на студент. * Добавяне на учител. * Добавяне на дисциплина. * Принтиране на студентите от даден клас. * Принтиране на студентите от не съществуващ клас. * Принтиране на учителите, които преподават дадена дисциплина. * Принтиране на учителите, които преподават дисциплина която не е въведена. | |
| **Вход** | **Изход** |
| AddStudent(Dqdo Mraz, 1A)  AddStudent(Snejanka, 1A)  AddStudent(Baba Matra, 3A)  PrintStudents(1A)  End. | Student added.  Student added.  Student added.  Dqdo Mraz  Snejanka |
| **Вход** | **Изход** |
| AddTeacher(Spqshtata Krasavica)  AddTeacher(Hristo Stoichkov)  AddTeacher(Lili Ivaniva)  AddDiscipline(Spqshtata Krasavica,History,12,12)  AddDiscipline(Hristo Stoichkov, sport, 11,11)  AddDiscipline(Lili Ivanova, History,12,12)  End. | Teacher added.  Teacher added.  Teacher added.  Discipline added.  Discipline added.  Discipline added. |
| **Вход** | **Изход** |
| AddTeacher(Spqshtata Krasavica)  AddTeacher(Hristo Stoichkov)  AddTeacher(Lili Ivaniva)  AddDiscipline(Spqshtata Krasavica,History,12,12)  AddDiscipline(Hristo Stoichkov, sport, 11,11)  AddDiscipline(Lili Ivanova, History,12,12)  PrintTeacher(History)  End. | Teacher added.  Teacher added.  Teacher added.  Discipline added.  Discipline added.  Discipline added.  Teacher name:Spqshtata Krasavica, History, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, |
| **Вход** | **Изход** |
| AddTeacher(Kumcho Valcho)  AddDiscipline(Kumcho Valcho, Digostaciq, 1, 1)  PrintTeacher(Anatomiq)  End. | Teacher added.  Discipline added.  No teacher. |
| **Вход** | **Изход** |
| AddStudent(Pepelqshka,1A)  AddStudent(Princa, 1A)  AddStudent(Prosqka,1A)  AddStudent(Ariel, 2B)  AddStudent(Pokahontas, 2B)  PrintStudents(1A)  PrintStudents(2B)  PrintStudents(2A)  End. | Student added.  Student added.  Student added.  Student added.  Student added.  Pepelqshka  Princa  Prosqka  Ariel  Pokahontas  No students. |
| **Вход** | **Изход** |
| AddStudent(Pepelqshka,1A)  AddStudent(Princa, 1A)  AddStudent(Prosqka,1A)  AddStudent(Ariel, 2B)  AddStudent(Pokahontas, 2B)  PrintStudents(1A)  PrintStudents(2B)  PrintStudents(2A)  AddTeacher(Mashtehata)  AddDiscipline(Mashtehata, gotvarstvo, 12123,12343)  PrintTeacher(gotvarstvo)  End. | Student added.  Student added.  Student added.  Student added.  Student added.  Pepelqshka  Princa  Prosqka  Ariel  Pokahontas  No students.  Teacher added.  Discipline added.  Teacher name:Mashtehata, gotvarstvo, 12123, 12343, |
| **Вход** | **Изход** |
| AddTeacher(Spqshtata Krasavica)  AddTeacher(Hristo Stoichkov)  AddTeacher(Lili Ivaniva)  AddDiscipline(Spqshtata Krasavica,History,12,12)  AddDiscipline(Hristo Stoichkov, sport, 11,11)  AddDiscipline(Lili Ivanova, History,12,12)  AddDiscipline(Lili Ivanova, Muzika,12,12)  AddDiscipline(Sofka, Muzika,12,12)  AddDiscipline(Boticheli, Muzika,12,12)  PrintTeacher(History)  PrintTeacher(Muzika)  End. | Teacher added.  Teacher added.  Teacher added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Teacher name:Spqshtata Krasavica, History, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, , Muzika, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, , Muzika, 12, 12,  Teacher name:Sofka, Muzika, 12, 12,  Teacher name:Boticheli, Muzika, 12, 12, |
| **Вход** | **Изход** |
| AddTeacher(Spqshtata Krasavica)  AddTeacher(Hristo Stoichkov)  AddTeacher(Lili Ivaniva)  AddDiscipline(Spqshtata Krasavica,History,12,12)  AddDiscipline(Hristo Stoichkov, sport, 11,11)  AddDiscipline(Lili Ivanova, History,12,12)  AddDiscipline(Lili Ivanova, Muzika,12,12)  AddDiscipline(Sofka, Muzika,12,12)  AddDiscipline(Boticheli, Muzika,12,12)  PrintTeacher(History)  PrintTeacher(Muzika)  End.  PrintStudents(1A)  AddStudent(Dqdo Mraz, 1A)  AddStudent(Snejanka, 1A)  AddStudent(Baba Matra, 3A)  PrintStudents(1A) | Teacher added.  Teacher added.  Teacher added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Discipline added.  Teacher name:Spqshtata Krasavica, History, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, , Muzika, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, , Muzika, 12, 12,  Teacher name:Sofka, Muzika, 12, 12,  Teacher name:Boticheli, Muzika, 12, 12, |
| **Вход** | **Изход** |
| AddDiscipline(Spqshtata Krasavica,History,12,12)  AddDiscipline(Hristo Stoichkov, sport, 11,11)  AddDiscipline(Lili Ivanova, History,12,12)  PrintTeacher(History)  End. | Discipline added.  Discipline added.  Discipline added.  Teacher name:Spqshtata Krasavica, History, 12, 12,  Teacher name:Lili Ivanova, History, 12, 12, |
|  |  |

|  |  |
| --- | --- |
| Задача 23 - 24 Реализация на списък с масив | |
| **Условие**  Напишете типизиран клас **GenericList<T>**, който пази списък от елементи от тип **T**. Пазете елементите от списъка в масив с фиксиран капацитет, който е зададен като параметър на конструктора на класа. При създаване на списъка с конструктор без параметри, началният капацитет трябва да е 4. Добавете методи за добавяне на елемент, достъпване на елемент по индекс, премахване на елемент по индекс, вмъкване на елемент на зададена позиция, изчистване на списъка, търсене на елемент по стойност и предефинирайте метода **ToString()**. Той трябва да извежда елементите във формат **{e0;e1;e2;e3;e4;…..eN-1}** където еP e елемента на позиция P в списъка без никакви интервали между тях. | |
| **Описание на входа**  Подаваните елементи за тестовете ще са от тип **double**.  На първият ред ще бъде подаден броя **N** елементи, който ще има в списъка.**[0 -100]**  На следващите **N** реда ще бъдат въведени елементите в списъка. Елементите ще са от посоченият тип.  На **N+2**-я ред ще бъдат зададени елементи от същият тип, които трябва да бъдат добавени последователно в края на списъка. Елементите ще бъдат разделени помежду си със ‘,’ (запетая), а разделителят между цяла и дробна част на всяко число ще бъде ‘.’ (точка).Бройката на тези елементи ще е **[0-10]**.  На **N+3** –я ред ще бъде зададен списък от индекси на елементи (цели числа в интервала [0,N-1]), които да бъдат премахнати от списъка. Числата ще бъдат в намаляващ ред, така че след премахването на всеки елемент да не се променят индексите на следващите елементи, които ще бъдат премахнати. Бройката на тези елементи ще е **[0-10]**.  На **N+4** –я ред ще бъде подаден елемент и позиция, на която елементът да бъде вмъкнат във формат елемент**:**позиция (позицията ще бъде валидна позиция в текущият масив). Възможно е да не се подават елементи за вмъкване.  На **N+5** –я ред ще бъде зададен елемент за търсене в така полученият списък.    Пример:  3  -5.34  0.03  -3.25  1.5 , -3.5 , 4  5 , 1, 0  -5.23:1  0 | |
| **Описание на изхода**  На първият ред да се отпечата текущият брой елементи в списъка след добавянето и премахването на гореописаните елементи.  На вторият ред да се отпечата капацитета на списъка след добавянето и премахването на гореописаните елементи.  На третият ред да се отпечата текущото съдържание на списъка с предефинираният метод **ToString();**  На четвъртият ред се отпечатва позицията на първо срещане в списъка на търсеният елемент или **“Not found!”** ако не е открит.  Пример:  4  8  {-3.25;-5.23;1.5;-3.5,}  “Not found!” | |
| **Анализ на задачата**  Показана е статичната реализация с масив на List<T>. Дефинирани са всички функции, описани в условието на задачата.При добавяне в края на списъка или при вмъкване на елемент, когато текущият капацитет не го позволява, капацитетът се увеличава двойно. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace List\_T  {  public class ArrayList<T>  {  private T[] arr;  private int count;  /// <summary>  /// Returns the actual list length  /// </summary>  public int Count  {  get  {  return count;  }  }  /// <summary>  /// Returns current list capacity  /// </summary>  public int Capacity  {  get  {  return this.arr.Length;  }  }  private static readonly int INITIAL\_CAPACITY = 4;  /// <summary>  /// Initializes the array-based list – allocate memory  /// </summary>  public ArrayList()  {  arr = new T[INITIAL\_CAPACITY];  count = 0;  }  /// <summary>  /// Initializes the array-based list – allocate memory  /// </summary>  public ArrayList(int capacity)  {  if (capacity < 4)  {  arr = new T[INITIAL\_CAPACITY];  }  else  {  arr = new T[capacity];  }  count = 0;  }  /// <summary>  /// Adds element to the list  /// </summary>  /// <param name="item">The element you want to add</param>  public void Add(T item)  {  Insert(count, item);  }  /// <summary>  /// Inserts the specified element at а given  /// position in this list  /// </summary>  /// <param name="index">  /// Index, at which the specified element is to be inserted  /// </param>  /// <param name="item">Element to be inserted</param>  /// <exception cref="System.IndexOutOfRangeException">Index is invalid</exception>  public void Insert(int index, T item)  {  if (index > count || index < 0)  {  throw new IndexOutOfRangeException(  "Invalid index: " + index);  }  T[] extendedArr = arr;  if (count + 1 == arr.Length)  {  extendedArr = new T[arr.Length \* 2];  }  Array.Copy(arr, extendedArr, index);  count++;  Array.Copy(arr, index, extendedArr, index + 1, count - index - 1);  extendedArr[index] = item;  arr = extendedArr;  }  /// Returns the index of the first occurrence  /// of the specified element in this list.  /// </summary>  /// <param name="item">The element you are searching</param>  /// <returns>  /// The index of given element or -1 is not found  /// </returns>  public int IndexOf(T item)  {    for (int i = 0; i < this.Count; i++)  {  if (item.Equals(arr[i]))  {  return i;  }  }  return -1;  }  /// <summary>  /// Clears the list  /// </summary>  public void Clear()  {  arr = new T[INITIAL\_CAPACITY];  count = 0;  }  /// <summary>  /// Checks if an element exists  /// </summary>  /// <param name="item">The item to be checked</param>  /// <returns>If the item exists</returns>  public bool Contains(T item)  {  int index = IndexOf(item);  bool found = (index != -1);  return found;  }  /// <summary>  /// Retrieves the element on the set index  /// </summary>  /// <param name="index">Index of the element</param>  /// <returns>The element on the current position</returns>  public T this[int index]  {  get  {  if (index >= count || index < 0)  {  throw new ArgumentOutOfRangeException(  "Invalid index: " + index);  }  return arr[index];  }  set  {  if (index >= count || index < 0)  {  throw new ArgumentOutOfRangeException(  "Invalid index: " + index);  }  arr[index] = value;  }  }  /// <summary>  /// Removes the element at the specified index  /// </summary>  /// <param name="index">  /// The index, whose element you want to remove  /// </param>  /// <returns>The removed element</returns>  public T Remove(int index)  {  if (index >= count || index < 0)  {  throw new ArgumentOutOfRangeException(  "Invalid index: " + index);  }  T item = arr[index];  Array.Copy(arr, index + 1, arr, index, count - index );    arr[count - 1] = default(T);  count--;  return item;  }  /// <summary>  /// Removes the specified item  /// </summary>  /// <param name="item">The item you want to remove</param>  /// <returns>Item index or -1 if item does not exists</returns>  public int Remove(T item)  {  int index = IndexOf(item);  if (index == -1)  {  return index;  }  Array.Copy(arr, index + 1, arr, index, count - index + 1);  count--;  return index;  }  public override string ToString()  {  StringBuilder sb = new StringBuilder();  sb.Append("{");    for (int i = 0; i < this.Count; i++)  {  sb.Append(arr[i]+";");  }  if (sb.Length > 1)  {  sb.Remove(sb.Length - 1, 1);  }  sb.Append("}");  Console.WriteLine(sb);  return sb.ToString();  }  }    class Program  {  static void Main(string[] args)  {    int numberOfElementsInList = int.Parse(Console.ReadLine());    ArrayList<double> MyList= new ArrayList<double>(numberOfElementsInList);    for (int i = 0; i < numberOfElementsInList; i++)  {  double currentNumber= double.Parse(Console.ReadLine());  MyList.Add(currentNumber);  }  string newElements = Console.ReadLine();  string[] elementsToAdd = newElements.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);  foreach (string element in elementsToAdd)  {  double elementToDouble = double.Parse(element);  MyList.Add(elementToDouble);  }  string inputIndexes = Console.ReadLine();  string[] indexesOfElementsToRemove = inputIndexes.Split(new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);  foreach (string element in indexesOfElementsToRemove)  {  int indexToInt=int.Parse(element);  MyList.Remove(indexToInt);  }  string elementToInsert = Console.ReadLine();  if (elementToInsert != string.Empty)  {  string[] elementAndPosition = elementToInsert.Split(new char[] { ':' }, StringSplitOptions.RemoveEmptyEntries);  MyList.Insert(int.Parse(elementAndPosition[1]), double.Parse(elementAndPosition[0]));  }  double elementToSearchInList = double.Parse(Console.ReadLine());    Console.WriteLine(MyList.Count);  Console.WriteLine(MyList.Capacity);    MyList.ToString();  if(MyList.Contains(elementToSearchInList)&&MyList.Count!=0)  {  Console.WriteLine(MyList.IndexOf(elementToSearchInList));  }  else  {  Console.WriteLine("Not found!");  }  }  }  } | |
| Тестове  1-3.Входни тестове  4-10 Проверяват създаването на списък със зададен капацитет и с капацитет по подразбиране. Проверяват търсенето на елемент, когато елемента присъства и не в списъка. Проверяват изтриването на всички елементи в списъка един по един. Проверяват дали предефинираният метод **ToString()**, работи вярно за различни списъци, включително за празен списък. | |
| **Вход** | **Изход** |
| 3  -5.34  0.03  -3.25  1.5 , -3.5 , 4  5 , 1, 0  -5,23:1  0 | 4  8  {-3.25;-5.23;1.5;-3.5}  “Not found!” |
| 4  -1.35  -123  0.0002  1.0002  1.5 , -3.5 , 4  5 , 1, 0  -5,23:4  0 | 5  8  {0.0002;1.0002;1.5;4;-5.23}  “Not found!” |
| 4  -1.35  -123  0.0002  1.0002  1.5 , -3.5 , 4  5 , 1, 0  ‘\n’  4.0000000000001 | 4  8  {0.0002;1.0002;1.5;4}  “Not found!” |
| 0  1,-2.34,123333  1  1:0  1 | 3  4  {1;1;123333}  0 |
| 0  1,-2.34,123333,-1245.34535,5  2  5:4  5 | 5  8  {1;-2.34;-1245.34535;5;5}  3 |
| 0  ‘\n’ (не се добавят елементи)  ‘\n’ (не се премахват елементи)  1:0  0 | 1  4  {1}  “Not found!” |
| 1  3  ‘\n’ (не се добавят елементи)  0  -5.23:0  -5.23 | 1  4  {-5.23}  0 |
| 9  0  1  2  3  4  5  6  7  8  0  9,8,7,6,5,4,3,2,1,0  0.000:0  0 | 1  18  {0}  0 |
| 1  -1  -2,-3,-4,-5,-6,-7,-8,-9,-10,-11  8,6,4,2,0  -11:5  -11 | 7  16  {-2;-4;-6;-8;-10;-11;-11}  5 |
| 0  ‘\n’ (не се добавят елементи)  ‘\n’ (не се премахват елементи)  ‘\n’ (не се вмъкват елементи)  0 | 0  4  {}  “Not found!” |

|  |  |
| --- | --- |
| Задача 25, Задача 26, Задача 27 Fraction | |
| **Условие**  25. Дефинирайте клас Fraction, който съдържа информация за рационална дроб (например ¼, ½). Дефинирайте статичен метод Parse(), който да опитва да създаде дроб от символен низ (например -3/4). Дефинирайте подходящи свойства и конструктори на класа. Предефинирайте оператори “+” и “-“ за събиране и изваждане на дроби. Напишете и свойство от тип Decimal, което връща десетичната стойност на дробта (например 0.25).  26. Напишете клас FractionTest, който тества функционалността на класа от предната задача Fraction. Отделете специално внимание на тестването на функцията Parse с различни входни данни.  27. Напишете функция, която съкращава дробта (Например ако числителя и знаменателя са съответно 10 и 15, дробта да се съкращава до 2/3). | |
| **Описание на входа**  Входът ще бъде на два реда като на всеки ред се подава **string** , който може да представлява рационална дроб.    Пример:  -2/4  2/3 | |
| **Описание на изхода**  На **първият ред** се отпечатва информация за първата дроб :   * **“The fraction is valid!”** ако дробта е валидна; * **“Division by zero!”** ако знаменателят е нула;   **- "Given string is not a valid fraction!" –** ако парсването е невъзможно  На **вторият ред** се отпечатва информация за втората дроб по същите правила, както за първият ред;  **В случай, че и двете дроби са валидни** програмата изпечатва **още**  **6 реда :**  **трети ред:**  съкратената първа дроб (ако не е възможно се извежда самата дроб) във формат **числител/знаменател** . (За примера -1/2)  **четвърти ред:**  съкратената втора дроб (ако не е възможно се извежда самата дроб) във формат **числител/знаменател** . (За примера 2/3)  **пети ред:** сумата от първата и втората дроб във формат **числител/знаменател**, следвани от десетичната стойност на сумата оградена в скоби и с точност до 2-я знак след десетичната запетая.Дробният резултат трябва да е опростен и задължително да съдържа числител и знаменател.За примера 1/6(0.17).  **шести ред:** разликата от първата и втората дроб (от първата се вади втората) във формат **числител/знаменател**, следвани от десетичната стойност на сумата оградена в скоби и с точност до 2-я знак след десетичната запетая. Дробният резултат трябва да е опростен и задължително да съдържа числител и знаменател. За примера -7/6(-1.17).  Пример:  “The fraction is valid!”  “The fraction is valid!”  -1/2  2/3  1/6(0,17)  -7/6(-1.17) | |
| **Анализ на задачата**  Реализиран е клас **Fraction** (рационална дроб) с две полета от тип **int**. Направена е функция Parse, която взима като параметър символен низ и проверява дали е валидна дроб.Ако е така извиква конструкторът за рационални дроби. Функция **Simplification** опростява дробта (ако е възможно). Чрез **GreatestCommonDivisor** (Най-голям общ делител) и **TheLeastCommonMultiple**(най-малко общо кратно), намерени по алгоритъма на Евклид, се опростява последващото предефиниране на операторите за събиране и изваждане на рационални дроби. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Text.RegularExpressions;  namespace Fraction  {  class Fraction  {  private int fractionNumerator;  private int fractionDenominator;  private Fraction(int fractionNumerator, int fractionDenominator)  {  try  {  if(fractionDenominator==0)  {  throw new DivideByZeroException();  }    else  {  this.fractionNumerator = fractionNumerator;  this.fractionDenominator = fractionDenominator;  }  }    catch (DivideByZeroException)  {  Console.WriteLine("Division by zero!");  }  catch (Exception)  {  Console.WriteLine("Unexpected error occured!");  }  }  /// <summary>  /// Returns the decimal value from this fraction.  /// </summary>  public string DecimalValue  {  get  {    decimal decimalValue=Math.Round(this.fractionNumerator / (decimal)this.fractionDenominator,2);  return "("+decimalValue.ToString("0.00")+")";  }  }  public int FractionNumerator  { get  {  return this.fractionNumerator;  }  private set  {  this.fractionNumerator=value;  }  }  public int FractionDenominator  { get  {  return this.fractionDenominator;  }  private set  {  if (value == 0)  {  throw new DivideByZeroException("Division by zero!");  }  else  {  this.fractionDenominator = value;  }  }  }  /// <summary>  /// Greatest common divisor by Euclidian algorithm (with recursion)  /// </summary>  /// <param name="firstNumber"></param>  /// <param name="secondNumber"></param>  /// <returns></returns>  private static int GreatestCommonDivisor(int firstNumber, int secondNumber)  {  if (firstNumber != 0)  {  int num1 = Math.Abs(firstNumber);  int num2 = Math.Abs(secondNumber);  if (num1 != num2)  {  if (num1 > num2)  {  return GreatestCommonDivisor(num1 - num2, num2);  }  else  {  return GreatestCommonDivisor(num1, num2 - num1);  }  }  return num1;  }  else  {  return secondNumber;  }  }  /// <summary>  /// The least common multiple  /// </summary>  /// <param name="firstNumber"></param>  /// <param name="secondNumber"></param>  /// <returns></returns>  private static int TheLeastCommonMultiple(int firstNumber, int secondNumber)  {    return firstNumber\*secondNumber/(GreatestCommonDivisor(firstNumber,secondNumber));  }  /// <summary>  /// Alows to find the Sum of two fractions  /// </summary>  /// <param name="a"> first fraction</param>  /// <param name="b">second fraction</param>  /// <returns></returns>  public static Fraction operator +(Fraction a, Fraction b)  {  int denominator = TheLeastCommonMultiple(a.fractionDenominator, b.fractionDenominator);  int numerator = a.fractionNumerator\*denominator/a.fractionDenominator + b.fractionNumerator\*denominator/b.fractionDenominator;  Fraction newFraction = new Fraction(numerator, denominator);  // newFraction.Simplification();  Simplification(newFraction);  return newFraction;  }  /// <summary>  /// Alows to substract second fraction from first  /// </summary>  /// <param name="a"> first fraction</param>  /// <param name="b">second fraction</param>  /// <returns></returns>  public static Fraction operator -(Fraction a, Fraction b)  {  int denominator = TheLeastCommonMultiple(a.fractionDenominator, b.fractionDenominator);  int numerator = a.fractionNumerator \* denominator / a.fractionDenominator - b.fractionNumerator \* denominator / b.fractionDenominator;  Fraction newFraction = new Fraction(numerator, denominator);  Simplification(newFraction);  return newFraction;  }  /// <summary>  /// Allows simplifikation of fraction  /// </summary>  private static void Simplification(Fraction fract)  {    int commonDivisor= GreatestCommonDivisor(fract.FractionNumerator, fract.FractionDenominator);  fract.fractionNumerator = fract.fractionNumerator / commonDivisor;  fract.FractionDenominator = fract.FractionDenominator / commonDivisor;  }    /// <summary>  /// Creates fraction from string /if possible/  /// </summary>  public static Fraction Parse( string fractionAsString )  {  try  {  int fractionNumerator;  int fractionDenominator;  string[] fractionParts = fractionAsString.Split(new char[] { '/' }, StringSplitOptions.RemoveEmptyEntries);  bool isNumeratorCorrect = int.TryParse(fractionParts[0], out fractionNumerator);  if (isNumeratorCorrect == false)  {  throw new FormatException();  }  if (fractionParts.Length > 1)  {  bool isDenominatorCorrect = int.TryParse(fractionParts[1], out fractionDenominator);  if (isDenominatorCorrect == false)  {  throw new FormatException();  }  if (fractionDenominator == 0)  {  throw new DivideByZeroException();  }  }  else  {  fractionDenominator = 1;  }  if (fractionDenominator < 0)  {  fractionDenominator = (-1) \* fractionDenominator;  fractionNumerator = (-1) \* fractionNumerator;  }  Fraction result= new Fraction(fractionNumerator, fractionDenominator);  Simplification(result);  Console.WriteLine("The fraction is valid!");  return result;  }  catch (FormatException)  {  Console.WriteLine("Given string is not a valid fraction!");  }  catch (DivideByZeroException)  {  Console.WriteLine("Division by zero!");  }  catch (Exception)  {  Console.WriteLine("Unexpected error occured!");  }    return default(Fraction);  }  public override string ToString()  {      string result = string.Empty;  try  {  if ((this == default(Fraction)))  {  throw new NullReferenceException();  }  else if (this.fractionDenominator == 0)  {  throw new DivideByZeroException();  }  else  {  result = this.fractionNumerator + "/" + this.fractionDenominator;  Console.Write(result);  return result;  }  }  catch (DivideByZeroException)  {  return "Division by zero!";  }  catch (NullReferenceException)  {  return "This is not a valid fraction!";  }    }      }  class FractionTest  {  static void Main(string[] args)  {  string firstInputLine = Console.ReadLine();  string secondInputLine = Console.ReadLine();  Fraction a = Fraction.Parse(firstInputLine);  Fraction b = Fraction.Parse(secondInputLine);  if ((a != null) && (b != null))  {  a.ToString();  Console.WriteLine();  b.ToString();  Console.WriteLine();  Fraction c = a + b;  c.ToString();  Console.Write(c.DecimalValue);  Fraction k = a - b;  Console.WriteLine();  k.ToString();  Console.WriteLine(k.DecimalValue);  }  }  }  } | |
| Тестове  1-2.Входни тестове  3-10 Тества се дали се вадят верни резултати при отрицателен знаменател, при знаменател 0, при числител 0, при стринг,който не може да се парсне.Тества се опростяването при числител 0 и т.н. | |
| **Вход** | **Изход** |
| -2/4  2/3 | “The fraction is valid!”  “The fraction is valid!”  -1/2  2/3  1/6(0,17)  -7/6(-1.17) |
| -10000/50  -1/2 | “The fraction is valid!”  “The fraction is valid!”  -200  -1/2  -401/2(-200.50)  -399/2(-199.50) |
| -8/8  11/22 | “The fraction is valid!”  “The fraction is valid!”  -1  1/2  -1/2(-0.50)  -3/2(-1.50) |
| 0  2 | “The fraction is valid!”  “The fraction is valid!”  0/1  2/1  2/1(2.00)  -2/1(-2.00) |
| 1/0  0/0 | “Division by zero!”  “Division by zero!” |
| 1 / 0.34  blabla | "Given string is not a valid fraction!"  "Given string is not a valid fraction!" |
| 123 / -246  -1 / -2 | “The fraction is valid!”  “The fraction is valid!”  -1/2  1/2  0/1(0.00)  -1/1(-1.00) |
| 1/ 2 4  1 1/4 | "Given string is not a valid fraction!"  "Given string is not a valid fraction!" |
| 0/2  0/2 | “The fraction is valid!”  “The fraction is valid!”  0/1  0/1  0/1(0.00)  0/1(0.00) |
| 0/-1  -24/-24 | “The fraction is valid!”  “The fraction is valid!”  0/1  1/1  1/1(1.00)  -1/1(-1.00) |