**Задачи за първата домашна работа за първия срок**

**Указания:** Решете задачите и направете NUnit тестове за всички примерни входни данни както и няколко, които да покриват сценарии с прихващане на изключения.

**I част. Обектно-ориентирано програмиране.**

**Задача 1. Списък на служители**

Дефинирайте клас **Employee**, съдържащ информация за **име, заплата, длъжност, отдел, ел.поща** и **възраст.** Полетата **име, заплата, длъжност** и **отдел** са **задължителни**, останалите са **опционални**.

Вашата задача е да напишете програма, която прочита **N** реда с информация за служители от конзолата, намира кой е отдела с най-висока средна заплата и за всеки служител от този отдел отпечатва неговото **име, заплата, ел.поща и възраст**. Служителите трябва да са **сортирани според заплатите им, в намаляващ ред**. Ако някой служител **няма ел.поща**, на нейно място трябва да се отпечати “**n/a**”, а ако няма указана **възраст**, да се изведе “**-1**” вместо това. **Заплатата** трябва да бъде отпечатана с **две цифри** след десетичния знак.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  Pesho 120.00 Dev Development pesho@abv.bg 28  Toncho 333.33 Manager Marketing 33  Ivan 840.20 ProjectLeader Development ivan@ivan.com  Gosho 0.20 Freeloader Nowhere 18 | Highest Average Salary: Development  Ivan 840.20 ivan@ivan.com -1  Pesho 120.00 pesho@abv.bg 28 |
| 6  Stanimir 496.37 Temp Coding stancho@yahoo.com  Yovcho 610.13 Manager Sales  Toshko 609.99 Manager Sales toshko@abv.bg 44  Venci 0.02 Director BeerDrinking beer@beer.br 23  Andrei 700.00 Director Coding  Popeye 13.3333 Sailor SpinachGroup popeye@pop.ey | Highest Average Salary: Sales  Yovcho 610.13 n/a -1  Toshko 609.99 toshko@abv.bg 44 |

**Задача 2. Тестов Клиент**

Създайте клас **BankAccount**

Класът трябва да има private полета за:

* id: int
* balance: double

Класът трябва да има и следните свойства и методи:

* **ID:** **int**
* **Balance:** **double**
* **Deposit(Double amount):** **void**
* **Withdraw(Double amount):** **void**

Предефинирайте и метода **ToString()**.

Трябва да поддържате следните операции, подавани като входни данни от конзолата:

* **Create {Id}**
* **Deposit {Id} {Amount}**
* **Withdraw {Id} {Amount}**
* **Print {Id}**
* **End**

Създайте методи към Program.cs за всяка от командите. Имайте в предвид и следната допълнителна обработка на данните:

* Ако се опитате да създадете сметка със съществуващо Id, изведете **"Account already exists".**
* Ако се опитате да извършите операция върху несъществуваща сметка, изведете **"****Account does not exist"**.
* Ако се опитате да изтеглите сума, която е по-голяма от баланса, изведете **"****Insufficient balance"**.
* Print командата, трябва да изведе **"Account ID{id}, balance {balance}"**. Закръглете баланса до втория знак след запетаята.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Create 1  Create 2  Deposit 1 20  Withdraw 1 30  Withdraw 1 10  Print 1  End | Account already exists  Insufficient balance  Account ID1, balance 10.00 |
| Create 1  Deposit 2 20  Withdraw 2 30  Print 2  End | Account does not exist  Account does not exist  Account does not exist |

**Решение**

Използвайте **Dictionary<int, BankAccount>** за да пазите сметките.

**Задача 3. Продавач на коли**

Дефинирайте два класа **Car** и **Engine.** Класът за колите **Car** има полета за **модел, двигател, тегло** и **цвят** (**model, engine, weight**, **color**). Всеки двигател (**Engine**) има характеристики модел, мощност, кубатура и КПД (**model**, **power, displacement**, **efficiency**). **Теглото** и **цвета** на колата и **кубатурата** и **КПД-то** на двигателя ѝ са **незадължителни** данни.

На първия ред ще получите число **N**, показващо колко реда с информации за двигатели ще получите, на всеки от следващите **N** реда ще има информация за по един **двигател** в следния формат “<**Model> <Power> <Displacement> <Efficiency>**”. След редовете с двигателите, на следващия ред ще получите число **M** – указващо броя на колите, които следват. На всеки от следващите **M** реда ще има информация за една **кола** в следния формат “<**Model> <Engine> <Weight> <Color>**”, където двигателят ще е **модел на съществуващ (описан вече) двигател**. Когато създавате обект за **кола**, трябва да пазите **указател към** **точния двигател**, вместо само модела на двигателя. Обърнете внимание на това, че незадължителните характеристики **може да липсват** от форматираните данни.

Вашата задача е да отпечатате всяка кола (в реда, в който сте ги получили) и информацията за нея във вида, посочен по-долу; ако някое от незадължителните полета не е посочено, отпечатайте “**n/a**” на негово място:

**<CarModel>:  
 <EngineModel>:  
 Power: <EnginePower>  
 Displacement: <EngineDisplacement>  
 Efficiency: <EngineEfficiency>  
 Weight: <CarWeight>  
 Color: <CarColor>**

Предефинирайте методите ToString(), така че да имате многократно използваем начин за извеждане на тези обекти.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 2  V8-101 220 50  V4-33 140 28 B  3  FordFocus V4-33 1300 Silver  FordMustang V8-101  VolkswagenGolf V4-33 Orange | FordFocus:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: 1300  Color: Silver  FordMustang:  V8-101:  Power: 220  Displacement: 50  Efficiency: n/a  Weight: n/a  Color: n/a  VolkswagenGolf:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: n/a  Color: Orange |
| 4  DSL-10 280 B  V7-55 200 35  DSL-13 305 55 A+  V7-54 190 30 D  4  FordMondeo DSL-13 Purple  VolkswagenPolo V7-54 1200 Yellow  VolkswagenPassat DSL-10 1375 Blue  FordFusion DSL-13 | FordMondeo:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: Purple  VolkswagenPolo:  V7-54:  Power: 190  Displacement: 30  Efficiency: D  Weight: 1200  Color: Yellow  VolkswagenPassat:  DSL-10:  Power: 280  Displacement: n/a  Efficiency: B  Weight: 1375  Color: Blue  FordFusion:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: n/a |

**Задача 4. Треньор на покемони**

Вие искате да сте най-добрия треньор на покемони, по-добър от всеки друг, така че сте се заели да ловите покемони. Дефинирайте клас **Trainer** и клас **Pokemon**. Треньорът трябва да има **име**, **брой значки** и **колекция от покемони. Покемонът** има **име, елемент** и **здраве**, всички стойности са **задължителни**. Всеки треньор **започва с 0 значки**.

От конзолата ще получите неизвестно колко редове, след които ще следва команда “**Tournament**”. Всеки от тези редове ще носи информация за покемона и треньора, който го е хванал във формата “<**ИмеНаТреньор> <ИмеНаПокемон> <ЕлементНаПокемона> <ЗдравеНаПокемона>”** където **ИмеНаТреньор** е името на треньора, хванал покемона; имената са **уникални,** няма как да има двама треньори с еднакви имена. След получаване на команда “**Tournament**” неизвестен брой редове ще съдържат като команда един от трите елемента “**Fire**”, “**Water**”, “**Electricity**”, които продължават, докато се получи команда “**End**”. За всяка от тези команди трябва да проверите дали треньорът има поне един покемон с дадения елемент. Ако да, треньорът получава 1 значка, в противен случай всичките му покемони **губят 10 точки здраве**, а ако даден покемон падне до **0 или по-малко точки здраве той умира** и трябва да бъде изтрит от колекцията на треньора. След като бъде получена команда “**End**” трябва да отпечатите всички треньори, **сортирани според броя на значките, които имат, в намаляващ ред** (ако двама треньори имат еднакъв брой значки те трябва да са сортирани според реда на тяхното появяване във входните данни),във формата “<**ИмеНаТреньор> <Значки> <БройПокемони>**”.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Pesho Charizard Fire 100  Gosho Squirtle Water 38  Pesho Pikachu Electricity 10  Tournament  Fire  Electricity  End | Pesho 2 2  Gosho 0 1 |
| Stamat Blastoise Water 18  Nasko Pikachu Electricity 22  Jicata Kadabra Psychic 90  Tournament  Fire  Electricity  Fire  End | Nasko 1 1  Stamat 0 0  Jicata 0 1 |

**Задача 5. Google**

Google винаги ви наблюдава, така че не би трябвало да сте изненадани, че те знаят всичко за вас (дори и за вашата покемон колекция). И понеже вие сте наистина добри в писането на класове от Google са ви помолили да напишете клас, който съдържа цялата информация, която те искат да събират за хората.

От конзолата ще получите неясно колко редове, завършващи накрая с команда “**End**”. На всеки от тези редове ще е информацията за един човек в един от следните формати:

* “**<Име> company <имеНаФирма> <отдел> <заплата>**”
* “**<Име> pokemon <имеНаПокемон> <типНаПокемона>”**
* “**<Име> parents <имеНаРодител> <рожденДенНаРодителя>**”
* “**<Име> children <имеНаДете> <рожденДенНаДете>**”
* “**<Име> car <моделНаКолата> <скоростНаКолата>**”

Вие трябва да структурирате цялата информация за всеки човек в клас с вложени класове. Имената на хората са **уникални** - няма двама души с еднакви имена, човек може да има **само 1 фирма** и **кола**, но има **множество родители, деца** и **покемони**. След като се получи команда “**End**” на следващия ред ще получите едно име и трябва да отпечатите цялата информация за този човек. Имайте в предвид, че информацията може да се промени във входните данни, например ако се получат множество редове, които указват фирмата на човека, само **последния** е този, който трябва да бъде запомнен. Заплатата трябва да бъде изведена с **два знака** след десетичния разделител.

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| PeshoPeshev company PeshInc Management 1000.00  TonchoTonchev car Trabant 30  PeshoPeshev pokemon Pikachu Electricity  PeshoPeshev parents PoshoPeshev 22/02/1920  TonchoTonchev pokemon Electrode Electricity  End  TonchoTonchev | TonchoTonchev  Company:  Car:  Trabant 30  Pokemon:  Electrode Electricity  Parents:  Children: |
| JelioJelev pokemon Onyx Rock  JelioJelev parents JeleJelev 13/03/1933  GoshoGoshev pokemon Moltres Fire  JelioJelev company JeleInc Jelior 777.77  JelioJelev children PudingJelev 01/01/2001  StamatStamatov pokemon Blastoise Water  JelioJelev car AudiA4 180  JelioJelev pokemon Charizard Fire  End  JelioJelev | JelioJelev  Company:  JeleInc Jelior 777.77  Car:  AudiA4 180  Pokemon:  Onyx Rock  Charizard Fire  Parents:  JeleJelev 13/03/1933  Children:  PudingJelev 01/01/2001 |

**Примери**

**Задача 6. Създаване на футболен отбор**

Футболен отбор има променлив **брой играчи, име и рейтинг**. Един играч има **име** и **статистика**, които са в основата на неговото ниво на умения. Един играч има статистика са **издръжливост**, **Спринт**, **дрибъл**, **подавания** и **стрелба**. Всяка статистика може да бъде в диапазона [0..100]. Общото ниво на умение на играч се изчислява като средна стойност на статистиките си. Само името на играча и неговата статистика трябва да бъдат видими за всички от външния свят. Всичко останали данни трябва да бъдат скрити. Отборът трябва да показва име, рейтинг (изчислена от нивата на средните умения на всички играчи в отбора и закръглена до цяло число) и методи за добавяне и премахване на играчи. Вашата задача е да моделирате екипа и играчите, чрез правилното използване на принципите на капсулиране. Покажи само свойствата, които трябва да бъдат видими и валидирайте данните по подходящ начин.

**Валидация на данните**

* Името не трябва да е празно, null, empty или да е само от интервали. Иначе, изведете "A name should not be empty. "
* Stats трябва да е в обхвата 0..100. Иначе изведете "[Име на статитистиката] should be between 0 and 100. "
* Ако получите команда да премахнете липсващ играч, изведете "Player [име на играча] is not in [Име на отбора] team. "
* Ако получите команда да добавите играч към липсващ отбор, изведете "Team [име на отбор] does not exists."
* Ако получите команда за показване на stats за липсващ oтбор, изведете "Team [име на отбор] does not exists."

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Team;Arsenal  Add;Arsenal;Kieran\_Gibbs;75;85;84;92;67  Add;Arsenal;Aaron\_Ramsey;95;82;82;89;68  Remove;Arsenal;Aaron\_Ramsey  Rating;Arsenal  END | Arsenal – 81 |
| Team;Arsenal  Add;Arsenal;Kieran\_Gibbs;75;85;84;92;67  Add;Arsenal;Aaron\_Ramsey;195;82;82;89;68  Remove;Arsenal;Aaron\_Ramsey  Rating;Arsenal  END | Endurance should be between 0 and 100.  Player Aaron\_Ramsey is not in Arsenal team.  Arsenal - 81 |
| Team;Arsenal  Rating;Arsenal  END | Arsenal – 0 |

**Задача 7. Калории на Pizza**

Пицата е изработена от тесто и различни гарнитури. Вие трябва да създадете класа Pizza, който трябва да има име, тесто и гарнитурата като полета. Всеки вид на съставка трябва да има свой собствен клас. Всяка съставка има различни свойства: тестото може да бъде бяло или пълнозърнесто и освен това то може да бъде хрупкави, chewy или домашно приготвени. Гарнитурата може да бъде от тип месо, зеленчуци, сирене или сос. Всяка съставка трябва да има тегло в грамове и метод за изчисляване на калориите му, според типа си. Калории на грам се изчисляват чрез модификатори. Всяка съставка има 2 калории на грам като база и Модификатор, която дава точна калории. Например, бяло тесто има Модификатор на 1.5, chewy тестото има Модификатор на 1.1, което означава, че бял chewy тесто 100 грама ще има 100 \* 1,5 \* 1.1 = 330,00 общо калории.

Вашата работа е да създадете класовете по такъв начин, че те правилно да са капсуловани и да предоставят публичен метод за всяка пица която изчислява калориите в зависимост от съставките си.

**Step 1. Създайте клас Dough**

Основната съставка на пицата е тестото. Първо трябва да създадете един клас за него. Той има тип брашно , който може да бъде бял или пълнозърнест. В допълнение има техника на печене, която може да бъде за хрупкави, жилави или домашно приготвени теста. Тестото трябва да има тегло в грамове. Калории на грам от тестото се изчисляват според типа на брашното и техниката на втастване. Всееки вид тестото има 2 калории на грам като база и Модификатор, който дава точните калории. Например, бялото тесто има Модификатор 1.5, жилавото тесто има Модификатор 1.1, което означава, че бяло жилаво тесто с тегло 100 грама ще има (2 \* 100) \* 1,5 \* 1.1 = 330,00 общо калории. По-долу Ви се предоставят модификатори:

* White – 1.5;
* Wholegrain – 1.0;
* Crispy – 0.9;
* Chewy – 1.1;
* Homemade – 1.0;

Everything that the class should expose is a getter for the calories per gram. Your task is to create the class with a proper constructor, fields, getters and setters. Make sure you use the proper access modifiers.

Всичко, което трябва да e видимо за класа е getter за калории на грам. Вашата задача е да се създаде клас с подходящ конструктор, полета, getters и setters. Проверете дали използвате правилните модификатори за достъп.

**Step 2. Валидирайте данните за класа Dough**

Променете вътрешната логика на класа Dough class чрез добавяне на валидация във setters.

Подсигурете при невалиден вход на типа брашно flour type или невалидна техника на печене подходящо изключение да се връща със съобщение "Invalid type of dough.".

Допустимото тегло на тестото е в диапазона [1..200] грама. Ако е извън диапазона да се върне изключение със съобщение "Dough weight should be in the range [1..200].".

**Съобщения на изключенията**

* "Invalid type of dough."
* "Dough weight should be in the range [1..200]."

Направете тест на метода main, който въвежда различни видове теста и извежда техните калории, докато се въведе команда "Край".

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Dough White Chewy 100  END | 330.00 |
| Dough Tip500 Chewy 100  END | Invalid type of dough. |
| Dough White Chewy 240  END | Dough weight should be in the range [1..200]. |

**Step 3. Създайте клас Topping**

След това трябва да създаде класа **Topping**. Тoй може да бъде четири различни видa – **месо, зеленчуци, сирене или сос**. Гарнитурата има **тегло** в грамове. Калориите на грам гарнитура се изчисляват в зависимост от типа ѝ. **Базовите калории** на грам са 2. Всеки различен вид гарнитура има модификатор. Например месото има Модификатор 1.5, така че месната гарнитура ще има 1.5 калории на грам (1 \* 1.5). Всичко, което трябва да изложи класа е getter за калории на грам. По-долу ви се предоставени модификаторите:

* Meat – 1.2;
* Veggies – 0.8;
* Cheese – 1.1;
* Sauce – 0.9;

Вашата задача е да се създаде клас с подходящ конструктор, полета, getters и setters. Проверете дали използвате правилни модификатори за достъп.

**Step 4. Валидиране на данните за класа Topping**

Сменете вътрешната логика на класа Topping, като добавите валидация на данните в setter-а.

Уверете се, че гарнитурата е измежду предоставените типове, в противен случай изведете подходящо изключение със съобщение "Cannot place [name of invalid argument] on top of your pizza“.

Теглото на гарнитурите е в диапазона [1..50] грама. Ако е извън този диапазон да се върне изключение със съобщението "[Topping type name] weight should be in the range [1..50].".

**Съобщения на изключенията**

* "Cannot place [name of invalid argument] on top of your pizza."
* "[Topping type name] weight should be in the range [1..50]."

Make a test in your main method that reads a single dough and a topping after that and prints their calories.

Направете тест на метода main, който въвежда количество тесто и гарнитури и след това извежда техните калории.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Dough White Chewy 100  Topping meat 30  END | 330.00  72.00 |
| Dough White chewy 100  Topping Krenvirshi 500  END | 330.00  Cannot place Krenvirshi on top of your pizza. |
| Dough White Chewy 100  Topping Meat 500  END | 330.00  Meat weight should be in the range [1..50]. |

**Step 5. Създайте клас Pizza !.**

Пицата трябва да има **име**, няколко **гарнитурата** и **тесто**. Използвайте двата класа, които сте направили по-рано. Пицата трябва да имат **публични getters** за нейното **име**, **брой гарнитури и общото количество калории**. **Общото количество калории** **се изчисляват чрез сумиране на калориите на всички съставки, които пицата има**. Създайте клас, използвайки подходящ конструктор, направете метод за добавяне на гарнитура, публични getters за тестото и за общото количество калории. Входът за пицата се състои от няколко реда. На първия ред **е името на пица и броя на гарнитурите, които има**. На втория ред да се въвежда **тестото**. На следващите редове ще получите всяка гарнитура на пицата. **Броят на редовете за гарнитурите** се въвежда на първия ред. Ако създаването на пица е **успешно** да се изведе на един ред името на пицата и **общото количество калории в нея**.

**Step 6. Валидиране на данни за клас Pizza**

**Името** на pizza **не** **трябва да е** **празен низ**. Също **не трябва да е повече от 15 символа**. Ако не отгoваrя на това условие, се връща **изключение** със съобщение "Pizza name should be between 1 and 15 symbols."

Броят на гарнитурите трябва да е в диапазона [0...10]. иначе се връща изключение със съобщение "Number of toppings should be in range [0..10]."

Вашата задача е да изведете името на пицата и общото количество калории в нея, според примера по-долу:

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Pizza Meatless 2  Dough Wholegrain Crispy 100  Topping Veggies 50  Topping Cheese 50  END | Meatless – 370.00 Calories. |
| Pizza Meatfull 5  Dough White cheWy 200  Topping Meat 50  Topping Cheese 50  Topping meat 20  Topping sauce 10  Topping Meat 30  END | Meatfull – 1028.00 Calories. |
| Pizza Bulgarian 20  Dough Tip500 Balgarsko 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Number of toppings should be in range [0..10]. |
| Pizza Bulgarian 2  Dough Tip500 Balgarsko 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Invalid type of dough. |
| Pizza Bulgarian 2  Dough White Chewy 100  Topping Sirene 50  Topping Cheese 50  Topping Krenvirsh 20  Topping Meat 10  END | Cannot place Sirene on top of your pizza. |

**II част. Конзолни MVC приложения.**

**Задача 1. Навреме за изпит**

Студент трябва да отиде **на** **изпит** **в** **определен час** (например в 9:30 часа). Той идва в изпитната зала в даден **час на пристигане** (например 9:40). Счита се, че студентът е дошъл **навреме**, ако е пристигнал в часа на изпита или до половин час преди това. Ако е пристигнал по-рано повече от 30 минути, той е **подранил**. Ако е дошъл след часа на изпита, той е **закъснял**. Напишете програма, която въвежда време на изпит и време на пристигане и отпечатва дали студентът е дошъл **навреме**, дали е **подранил** или е **закъснял** и **с колко часа или минути** е подранил или закъснял.

**Вход**

От конзолата се четат **4 цели числа** (по едно на ред):

* Първият ред съдържа **час на изпита** – цяло число от 0 до 23.
* Вторият ред съдържа **минута на изпита** – цяло число от 0 до 59.
* Третият ред съдържа **час на пристигане** – цяло число от 0 до 23.
* Четвъртият ред съдържа **минута на пристигане** – цяло число от 0 до 59.

**Изход**

На първият ред отпечатайте:

* “**Late**”, ако студентът пристига по-късно от часа на изпита.
* “**On time**”, ако студентът пристига точно в часа на изпита или до 30 минути по-рано.
* “**Early**”, ако студентът пристига повече от 30 минути преди часа на изпита.

Ако студентът пристига с поне минута разлика от часа на изпита, отпечатайте на следващия ред:

* “**mm** **minutes before the start**” за идване по-рано с по-малко от час.
* “**hh:mm** **hours before the start**” за подраняване с 1 час или повече. Минутите винаги печатайте с 2 цифри, например “1:05”.
* “**mm** **minutes after the start**” за закъснение под час.
* “**hh:mm** **hours after the start**” за закъснение от 1 час или повече. Минутите винаги печатайте с 2 цифри, например “1:03”.

**Примерен вход и изход**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Вход** | **Изход** |  | **Вход** | **Изход** |  | **Вход** | **Изход** |
| 9  30  9  50 | Late  20 minutes after the start | 9  00  10  30 | Late  1:30 hours after the start | 10  00  10  00 | On time |
| 9  00  8  30 | On time  30 minutes before the start |  | 14  00  13  55 | On time  5 minutes before the start | 11  30  10  55 | Early  35 minutes before the start |
| 16  00  15  00 | Early  1:00 hours before the start |  | 11  30  8  12 | Early  3:18 hours before the start | 11  30  12  29 | Late  59 minutes after the start |

**Задача 2. Генератор за тъпи пароли**

Да се напише програма, която въвежда две цели числа ***n*** и ***l*** и генерира по азбучен ред всички възможни **“тъпи” пароли**, които се състоят от следните **5 символа**:

* Символ 1: цифра от **1** до ***n***.
* Символ 2: цифра от **1** до ***n***.
* Символ 3: малка буква измежду първите ***l*** букви на латинската азбука.
* Символ 4: малка буква измежду първите ***l*** букви на латинската азбука.
* Символ 5: цифра от 1 до **n**, по-голяма от първите 2 цифри.

**Вход**

Входът се чете от конзолата и се състои от две **цели числа** ***n*** и ***l*** в интервала [**1**…**9**], по едно на ред.

**Изход**

На конзолата трябва да се отпечатат **всички “тъпи” пароли** по **азбучен ред**, разделени с **интервал**.

**Примерен вход и изход**

|  |  |
| --- | --- |
| **вход** | **изход** |
| 2  4 | 11aa2 11ab2 11ac2 11ad2 11ba2 11bb2 11bc2 11bd2 11ca2 11cb2 11cc2 11cd2 11da2 11db2 11dc2 11dd2 |
| 3  1 | 11aa2 11aa3 12aa3 21aa3 22aa3 |
| 3  2 | 11aa2 11aa3 11ab2 11ab3 11ba2 11ba3 11bb2 11bb3 12aa3 12ab3 12ba3 12bb3 21aa3 21ab3 21ba3 21bb3 22aa3 22ab3 22ba3 22bb3 |
| 4  2 | 11aa2 11aa3 11aa4 11ab2 11ab3 11ab4 11ba2 11ba3 11ba4 11bb2 11bb3 11bb4 12aa3 12aa4 12ab3 12ab4 12ba3 12ba4 12bb3 12bb4 13aa4 13ab4 13ba4 13bb4 21aa3 21aa4 21ab3 21ab4 21ba3 21ba4 21bb3 21bb4 22aa3 22aa4 22ab3 22ab4 22ba3 22ba4 22bb3 22bb4 23aa4 23ab4 23ba4 23bb4 31aa4 31ab4 31ba4 31bb4 32aa4 32ab4 32ba4 32bb4 33aa4 33ab4 33ba4 33bb4 |

**Задача 3. Тръби в басейн**

Басейн с **обем V** има **две тръби** от които се пълни. **Всяка тръба има определен дебит** (литрите вода минаващи през една тръба за един час). Работникът **пуска тръбите едновременно** и излиза за **N часа**. Напишете програма, която изкарва състоянието на басейна, **в момента, когато работникът се върне.**

**Вход**

От конзолата се четат **четири реда**:

* Първият ред съдържа числото **V – Обем на басейна в литри** – цяло число в интервала [1…10000].
* Вторият ред съдържа числото **P1 – дебит на първата тръба за час** – цяло число в интервала [1…5000].
* Третият ред съдържа числото **P2 – дебит на втората тръба за час**– цяло число в интервала [1…5000].
* Четвъртият ред съдържа числото **H – часовете които работникът отсъства** – число с плаваща запетая в интервала [1.0…24.00]

**Изход**

Да се отпечата на конзолата **едно от двете възможни състояния**:

* До колко се е запълнил басейна и коя тръба с колко процента е допринесла. Всички проценти се свеждат до цяло число (без закръгляне).
  + "The pool is **[x]%** full. Pipe 1: **[y]%**. Pipe 2: **[z]%**."
* Aко басейнът се е препълнил – с колко литра е прелял за даденото време, число с плаваща запетая
  + "For **[x]** hours the pool overflows with **[y]** liters."

\* **Имайте предвид**, че поради **свеждането до цяло число** се **губят данни** и нормално **сборът на процентите да е 99%, а не 100%.**

**Примерен вход и изход**

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 1000  100  120  3 | The pool is 66% full. Pipe 1: 45%. Pipe 2: 54%. | За 3 часа:  Първата тръба е напълнила – 300 л.  Втората тръба е напълнила – 360 л.  Общо – 660 л. < 1000 л. => 66% са запълнени  Първата тръба е допринесла с 45% (300 от 660 л.).  Втората тръба е допринесла с 54% (360 от 660 л.). |
| 100  100  100  2.5 | For 2.5 hours the pool overflows with 400 liters. | За 2.5 часа:  Първата тръба е напълнила – 250 л.  Втората тръба е напълнила – 250 л.  Общо – 500 л. > 100 л. => 400 л. са преляли. |

**Задача 4. Деление без остатък**

Дадени са **n** **цели числа** в интервала [**1**…**1000**]. От тях някакъв **процент p1 се делят без остатък на 2**, друг **процент** **p2** се **делят без остатък на 3**, друг **процент** **p3** се **делят без остатък на 4**. Да се напише програма, която изчислява и отпечатва процентите **p1**, **p2** и **p3**.

**Пример**: имаме n = 1**0** числа: 680, 2, 600, 200, 800, 799, 199, 46, 128, 65. Получаваме следното разпределение и визуализация:

|  |  |  |  |
| --- | --- | --- | --- |
| **Деление без остатък на:** | **Числа в диапазона** | **Брой числа** | **Процент** |
| 2 | 680, 2, 600, 200, 800, 46, 128 | 7 | p1 = 7.0 / 10 \* 100 = **70.00**% |
| 3 | 600 | 1 | p2 = 1 / 10 \* 100 = **10.00**% |
| 4 | 680, 600, 200, 800, 128 | 5 | p3 = 5 / 10 \* 100 = **50.00**% |

**Вход**

На първия ред от входа стои цялото число **n** (1 ≤ **n** ≤ 1000) – брой числа. На следващите **n** **реда** стои **по едно** **цяло число** в интервала [**1**…**1000**] – числата които да бъдат проверени на колко се делят.

**Изход**

Да се отпечатат на конзолата **3 реда**, всеки от които съдържа процент между 0% и 100%, с точност две цифри след десетичната точка, например 25.00%, 66.67%, 57.14%.

* На **първият ред** – процентът на числата които **се делят на 2**
* На **вторият ред** – процентът на числата които **се делят на** **3**
* На **третият ред** – процентът на числата които **се делят на 4**

**Примерен вход и изход**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вход** | **Изход** |  | **Вход** | **Изход** |
| **10**  680  2  600  200  800  799  199  46  128  65 | 70.00%  10.00%  50.00% | **3**  3  6  9 | 33.33%  100.00%  0.00% |

**Задача 5. Магически числа**

Да се напише програма, която въвежда едно цяло **„магическо“** число и изкарва **всички** възможни **6-цифрени числа**, за които **произведението на неговите цифри** е **равно** на **„магическото“ число.**

**Пример:** „Магическо число“ -> 2

* 111112 -> 1 \* 1 \* 1 \* 1 \* 1 \* 2 = 2
* 111121 -> 1 \* 1 \* 1 \* 1 \* 2 \* 1 = 2
* 111211 -> 1 \* 1 \* 1 \* 2 \* 1 \* 1 = 2
* 112111 -> 1 \* 1 \* 2 \* 1 \* 1 \* 1 = 2
* 121111 -> 1 \* 2 \* 1 \* 1 \* 1 \* 1 = 2
* 211111 -> 2 \* 1 \* 1 \* 1 \* 1 \* 1 = 2

**Вход**

Входът се чете от конзолата и се състои от **едно** **цяло число** в интервала [**1**…**600000**].

**Изход**

На конзолата трябва да се отпечатат **всички “магически” числа**, разделени с **интервал**.

**Примерен вход и изход**

|  |  |
| --- | --- |
| **вход** | **изход** |
| 2 | 111112 111121 111211 112111 121111 211111 |
| 8 | 111118 111124 111142 111181 111214 111222 111241 111412 111421 111811 112114 112122 112141 112212 112221 112411 114112 114121 114211 118111 121114 121122 121141 121212 121221 121411 122112 122121 122211 124111 141112 141121 141211 142111 181111 211114 211122 211141 211212 211221 211411 212112 212121 212211 214111 221112 221121 221211 222111 241111 411112 411121 411211 412111 421111 811111 |
| 531441 | 999999 |

**Задача 6. Ремонт на плочки**

**На** **площадката** пред жилищен блок трябва да се **поставят плочки**. **Площадката** **е** с форма на **квадрат** **със страна N метра**. **Плочките** са **широки „W“ метра** и **дълги „L“ метра**. На площадката има **една** **пейка с ширина M метра** и **дължина O метра**. Под нея **не е нужно** да се слагат плочки. **Всяка плочка** се поставя за **0.2 минути**.

Напишете **програма**, която **чете от конзолата** **размерите** на **площадката**, **плочките** и **пейката** и пресмята **колко плочки са необходими** да се покрие площадката и пресмята **времето за поставяне на всички плочки**.

**Пример:** **площадка** с **размер** **20м.** има **площ 400кв.м**. **Пейка** широка **1м.** и дълга **2м.**, заема **площ 2кв.м**. Една **плочка** е **широка** **5м.** и **дълга 4м.** има **площ = 20кв.м.** **Площта** която трябва да се покрие е **400 – 2 = 398 кв.м.** Необходими са **398 / 20 = 19.90 плочки**. Необходимото **време** е **19.90 \* 0.2 = 3.98 минути**.

**Вход**

От конзолата се четат **5 числа**:

* **N** – **дължината** на **страна** от **площадката** в интервала **[1...100]**
* **W** – **широчината** на една **плочка** в интервала **[0.1...10.00]**
* **L** – **дължината** на една **плочка** в интервала **[0.1...10.00]**
* **М** – **широчината** на **пейката** в интервала **[0...10]**
* **О** – **дължината** на **пейката** в интервала **[0...10]**

**Изход**

Да се отпечата на конзолата две числа: **броят плочки** необходи за ремонта и **времето за поставяне**, всяко на нов ред.

**Примерен вход и изход**

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 20  5  4  1  2 | 19.9  3.98 | **Обща площ** = 20 \* 20 = **400**; площ на **пейката** = 1 \* 2 = **2**  **Площ** **за покриване** = 400 – 2 = **398**  **Площ на плочки** = 5 \* 4 = **20**  **Необходими плочки** = 398 / 20 = **19.9**  **Необходимо време** = 19.9 \* 0.2 = **3.98** |
| 40  0.8  0.6  3  5 | 3302.08333333333  660.416666666667 | |

**Задача 7. Хистограма**

Дадени са **n** **цели числа** в интервала [**1**…**1000**]. От тях някакъв процент **p1** са под 200, друг процент **p2** са от 200 до 399, друг процент **p3** са от 400 до 599, друг процент **p4** са от 600 до 799 и останалите **p5** процента са от 800 нагоре. Да се напише програма, която изчислява и отпечатва процентите **p1**, **p2**, **p3**, **p4** и **p5**.

**Пример**: имаме n = **20** числа: 53, 7, 56, 180, 450, 920, 12, 7, 150, 250, 680, 2, 600, 200, 800, 799, 199, 46, 128, 65. Получаваме следното разпределение и визуализация:

|  |  |  |  |
| --- | --- | --- | --- |
| **Диапазон** | **Числа в диапазона** | **Брой числа** | **Процент** |
| < 200 | 53, 7, 56, 180, 12, 7, 150, 2, 199, 46, 128, 65 | 12 | p1 = 12 / 20 \* 100 = **60.00**% |
| 200 … 399 | 250, 200 | 2 | p2 = 2 / 20 \* 100 = **10.00**% |
| 400 … 599 | 450 | 1 | p3 = 1 / 20 \* 100 = **5.00**% |
| 600 … 799 | 680, 600, 799 | 3 | p4 = 3 / 20 \* 100 = **15.00**% |
| ≥ 800 | 920, 800 | 2 | p5 = 2 / 20 \* 100 = **10.00**% |

**Вход**

На първия ред от входа стои цялото число **n** (1 ≤ **n** ≤ 1000) – брой числа. На следващите **n** **реда** стои **по едно** **цяло число** в интервала [**1**…**1000**] – числата върху които да бъде изчислена хистограмата.

**Изход**

Да се отпечата на конзолата **хистограмата** – **5 реда**, всеки от които съдържа число между 0% и 100%, с точност две цифри след десетичната точка, например 25.00%, 66.67%, 57.14%.

**Примерен вход и изход**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Вход** | **Изход** |  | **Вход** | **Изход** |  | **Вход** | **Изход** |  | **Вход** | **Изход** |  | **Вход** | **Изход** |
| **3**  1  2  999 | 66.67%  0.00%  0.00%  0.00%  33.33% | **4**  53  7  56  999 | 75.00%  0.00%  0.00%  0.00%  25.00% | **7**  800  801  250  199  399  599  799 | 14.29%  28.57%  14.29%  14.29%  28.57% | **9**  367  99  200  799  999  333  555  111  9 | 33.33%  33.33%  11.11%  11.11%  11.11% | **14**  53  7  56  180  450  920  12  7  150  250  680  2  600  200 | 57.14%  14.29%  7.14%  14.29%  7.14% |