# Konteinerinė klasė

Susipažinsite su:

* vartotojo klasės metodu ToString(), paveldėtu iš klasės Object ir užklotu vartotojo klasėje;
* eilučių formavimo operatoriumi string.Format();
* klase įtraukiančia kitų klasių objektus;
* palyginimo ir loginių operacijų užklojimu;
* rikiavimo, pašalinimo algoritmais;

Visi programiniai kodai, kurie pateikiami šiame praktiniame darbe pateikti šioje saugykloje:

<https://bitbucket.org/blaztoma/lab2/downloads>

Naudokite juos, jei nesiseka savarankiškai išspręsti uždavinių.

## Konteinerinės klasės objektų rinkiniams naudojant masyvus

**Užduotis**.

Projektas sulaukė pasisekimo, be pagrindinio filialo Kaune firma atidarė filialus Vilniuje ir Šiauliuose.

Dėl to buvo pakeistas duomenų failo formatas, pirmoje eilutėje – miestas, o toliau nurodomi šunų duomenys.

Iš vieno miesto gali būti keli duomenų failai.

1. Nuskaityti Kaune, Vilniuje ir Šiauliuose registruotų šunų duomenis ir atspausdintų Vilniuje registruotus šunis.
2. Rasti ir atspausdinti, kokių veislių šunys užregistruoti Šiauliuose.
3. Suskaičiuoti ir atspausdinti, kiek agresyvių šunų yra Kaune.
4. Surasti ir atspausdinti populiariausią Vilniuje registruotų šunų veislę.
5. Surasti besidubliuojančius šunis. Žmonės kartais persikelia į gyventi į kitą miestą, ir priregistruoja ten savo augintinį. O ankstesnėje vietoje išsiregistruoti pamiršta. Raskite šunis, užregistruotus keliuose miestuose (lyginti mikroschemos identifikatorių ir šuns vardą).
6. Besidubliuojantys duomenys registre turi būti pašalinti. Realizuokite besidubliuojančių šunų pašalinimą iš nurodyto filialo registro.
7. Šunų mikroschemų identifikatoriai priskiriami didėjimo tvarka. Realizuokite Kauno filialo šunų rikiavimą identifikatorių didėjimo tvarka.

**Pradiniai duomenys.**

|  |
| --- |
| **Pradiniai duomenys** |
| Failas: **L2Data-Kaunas.csv** |
| Kaunas  Reksas,91357,Amerikiečių buldogas,Petrauskas J.,877724882,2015-07-24,TRUE  Meškis,92468,Biglis,Sofauskas A.,878617827,2014-05-17,FALSE  Nora,93579,Labradoro retriveris,Pagalvytė G.,878362552,2015-03-11,FALSE  Margis,94680,Neveislinis,Kėdžius J.,878622676,2014-12-07,FALSE  Linksmutis,90864,Kaukazo aviganis,Stalčiukaitė R.,877625252,2014-09-06,TRUE  Bonė,99753,Taksas,Lovauskas I.,875736251,2015-07-25,FALSE  Džekas,98642,Biglis,Langaitė I.,878736252,2014-05-20,FALSE  Baronas,97531,Stafordšyro bulterjeras,Paveikslaitė D.,877462625,2014-08-18,TRUE  Džema,96420,Jorkšyro terjeras,Lentauskas D.,877536225,2015-05-02,FALSE |
| Failas: **L2Data-Kaunas2.csv** |
| Kaunas  Džekas,98642,Biglis,Langaitė I.,878736252,2014-05-20,FALSE  Liksė,95790,Labradoro retriveris,Foteliūtė R.,877383273,2014-06-19,FALSE  Grantas,91470,Amerikiečių buldogas,Knygius R.,878649380,2015-05-24,TRUE  Miledi,92683,Pudelis,Palangiauskas G.,877201928,2015-02-14,FALSE  Aksis,99827,Taksas,Dalgiauskas A.,874930283,2014-10-27,FALSE  Lapcis,94725,Pudelis,Grindžiūtė A.,874938273,2014-05-13,FALSE  Džekas,91567,Amerikiečių buldogas,Grindžiūtė A.,874938273,2014-05-13,FALSE |
| Failas: **L2Data-Šiauliai.csv** |
| Šiauliai  Džema,96420,Jorkšyro terjeras,Lentauskas D.,877536225,2015-05-02,FALSE  Liksė,95790,Labradoro retriveris,Foteliūtė R.,877383273,2014-06-19,FALSE |
| Failas: **L2Data-Vilnius.csv** |
| Vilnius  Grantas,91470,Amerikiečių buldogas,Knygius R.,878649380,2015-05-24,TRUE  Miledi,92683,Pudelis,Knygius R.,878649380,2015-09-01,FALSE  Berta,90273,Taksas,Plaktukaitė R.,876283736,2014-05-11,FALSE  Lopsis,90876,Stafordšyro bulterjeras,Pjūklaitė I.,876403928,2014-11-09,TRUE  Nita,93261,Neveislinis,Grėbliauskas G.,876473928,2015-05-13,FALSE  Ambra,97365,Biglis,Lygiauskas V.,878739276,2014-07-15,FALSE  Bartas,92725,Jorkšyro terjeras,Dėklaitė J.,878392074,2014-06-30,FALSE  Kebabas,97362,Kaukazo aviganis,Tinklaitė S.,877049387,2015-05-29,TRUE |

**Programos kūrimo eiga.**

* Naudosime pirmame laboratoriniame darbe sukurtą projektą, kurį modifikuosime ir papildysime nauju funkcionalumu:
  + Nukopijuokite ankstesnio laboratorinio darbo visą sprendimą (*solution*) į atskirą katalogą ir pavadinkite jį **Lab2**.
* Jei norite (nebūtina) – galite pritaikyti sprendimo struktūrą prie antro laboratorinio darbo
  + Pakeiskite sprendimo projektų pavadinimus (pvz. **Lab1.Step1** į **Lab2.Step1**)
  + Pakeiskite *namespace* visuose projektuose taip, kad jie atspindėtų projekto pavadinimą
* Išmeskite iš projekto seną duomenų failą ir sukurkite naujus.
* Filialams sukursime naują klasę Branch, kuri saugos kiekvieno filialo šunų registracijos duomenis.
* Šuns klasėje Dog užklosime metodą ToString() tam, kad suformuotume eilutę išvedimui.
* Pagrindinėje programoje sukursime naujus arba modifikuosime esamus įvedimo ir spausdinimo metodus.
* Realizuojami metodai užduočių sprendimui.

⮋**Pirmas žingsnis.**

* Sukurkite atskirame faile klasę Branch filialuose registruotų šunų duomenims saugoti:

namespace Lab2.Step1

{

class Branch

{

public const int MaxNumberOfDogs = 50;

public string Town { get; set; }

public Dog[] Dogs { get; set; }

public int DogCount { get; private set; }

public Branch(string town)

{

Town = town;

Dogs = new Dog[MaxNumberOfDogs];

}

public void AddDog(Dog dog)

{

Dogs[DogCount] = dog;

DogCount++;

}

}

}

Kaip matome – filialas turi miesto pavadinimą (*Town*), jame registruotų šunų masyvą (*Dogs*) ir šunų skaitliuką (*DogCount*). Sukuriant filialą konstruktoriui reikia pateikti miesto pavadinimą. Inicializacijos metu filialo objektui priskiriama miesto reikšmė bei sukuriamas (tuščias) šunų masyvas. Be konstruktoriaus ši klasė turi *AddDog* metodą, kuris prideda naują registruotą šunį į šunų masyvą.

* Realizuokite duomenų nuskaitymą iš failo pagrindinėje programoje:

//------------------------------------------------------------

private static Branch GetBranchByTown(Branch[] branches, string town)

{

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

{

if(branches[i].Town == town)

{

return branches[i];

}

}

return null;

}

private static void ReadDogData(string file, Branch[] branches)

{

string town = null;

using (StreamReader reader = new StreamReader(@file))

{

string line = null;

line = reader.ReadLine();

if (line != null)

{

town = line;

}

Branch branch = GetBranchByTown(branches, town);

while (null != (line = reader.ReadLine()))

{

string[] values = line.Split(',');

string name = values[0];

int chipId = int.Parse(values[1]);

string breed = values[2];

string owner = values[3];

string phone = values[4];

DateTime vd = DateTime.Parse(values[5]);

bool aggressive = bool.Parse(values[6]);

Dog dog = new Dog(name, chipId, breed, owner, phone, vd, aggressive);

if (!branch.Dogs.Contains(dog))

{

branch.AddDog(dog);

}

}

}

}

Atkreipkite dėmesį, kad sukūrėme papildomą metodą (*GetBranchByTown*). Taip padarėme todėl, kad filialo objekto suradimas pagal miesto pavadinimą neturėtų būti metodo, kuris skaito duomenis atsakomybė.

* Padarysime nuskaitytų duomenų spausdinimą. Kad būtų paprasčiau suformuoti šuns duomenų eilutę – *Dog* klasėje perklosime metodą ToString():

public override String ToString()

{

return String.Format("ChipId: {0,5}, Name: {1,10}, Owner: {2,16} ({3}), Last vaccination date: {4:yyyy-MM-dd}", ChipId, Name, Owner, Phone, VaccinationDate);

}

Atkreipkite dėmesį į metodo aprašo eilutę. Direktyva override nusako, jog perrašomas standartinis ToString() metodas.

Dabar modifikuosime PrintDogsToConsole metodą, kuris į konsolę išrašo pateiktų šunų sąrašą:

/// <summary>

/// Show dog data in the Console

/// </summary>

/// <param name="dogs">Array of dogs</param>

/// <param name="count">Count of the dogs</param>

static void PrintDogsToConsole(Dog[] dogs, int count)

{

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

{

Console.WriteLine("Nr {0}: {1}\n", (i+1), dogs[i].ToString());

}

}

* Main() metode organizuokite Vilniaus, Kauno ir Šiaulių filialų nuskaitymą ir duomenų išvedimą į konsolę. Jei modifikuojate ankstesnę programą – pašalinkite ankstesnį šio metodo turinį.

using System;

using System.IO;

namespace Lab2.Step1

{

class Program

{

public const int NumberOfBranches = 3;

public const int MaxNumberOfBreeds = 10;

public const int MaxNumberOfDogs = 50;

static void Main(string[] args)

{

Branch[] branches = new Branch[3];

int branchCount = 3;

branches[0] = new Branch("Kaunas");

branches[1] = new Branch("Vilnius");

branches[2] = new Branch("Šiauliai");

string[] filePaths = Directory.GetFiles(Directory.GetCurrentDirectory(), "\*.csv");

foreach (string path in filePaths)

{

ReadDogData(path, branches);

}

PrintDogsToConsole(branches[1].Dogs, branches[1].DogCount);

}

}

}

⮋**Antras žingsnis.**

* Pasinaudosime pirmame laboratoriniame darbe sukurtu metodu GetBreeds tam, kad atspausdinti tam tikruose filialuose užregistruotus šunis. Metodas lieka toks pats, tiesiog paduosime reikalingo (šiuo atveju – Šiaulių) filialo šunis. Main() metode įterpkite šį kodą:

int breedCount;

string[] breeds = GetBreeds(branches[2].Dogs, branches[2].DogCount, out breeds, out breedCount);

Console.WriteLine("Šiauliuose užregistruotos šunų veislės:");

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

{

Console.WriteLine(breeds[i]);

}

Console.WriteLine();

⮋**Trečias žingsnis.**

* Sukurkite metodą, kuris skaičiuotų agresyvių šunų kiekį pateiktame šunų masyve. Pirmame darbe mes buvome sukūrę metodą, kuris suskaičiuoja agresyvius tam tikro amžiaus šunis. Šį metodą galite palikti - sukursime naują metodą tuo pačiu pavadinimu, tačiau skirtingu parametrų rinkinių. Tuo būdu perdengsime seną metodą ir esant reikalui galėsite naudoti abu metodus.

private static int CountAggressive(Dog[] dogs, int dogCount)

{

int counter = 0;

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

{

if (dogs[i].Aggressive)

{

counter++;

}

}

return counter;

}

* Iškvieskite šį metodą pagrindinėje programoje ir atspausdinkite rezultatą:

Console.WriteLine("Agresyviausi Kauno šunys: {0}", CountAggressive(branches[0].Dogs, branches[0].DogCount));

⮋**Ketvirtas žingsnis.**

* Parašykite metodą, kuris suranda populiariausią veislę šunų sąraše:

private static string GetMostPopularBreed(Dog[] dogs, int dogCount)

{

String popular = "Nerasta";

int count = 0;

int breedCount = 0;

string[] breeds = GetBreeds(dogs, dogCount, out breeds, out breedCount);

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

{

FilterByBreed(dogs, dogCount, breeds[i], out breedCount);

if(breedCount > count)

{

popular = breeds[i];

count = breedCount;

}

}

return popular;

}

* Panaudokite šį metodą, surasdami ir pateikdami populiariausią Vilniaus miesto veislę:

Console.WriteLine("Populiariausia veislė Vilniuje: {0}", GetMostPopularBreed(branches[1].Dogs, branches[1].DogCount));

* Galbūt pastebėjote, jog šis metodas grąžins pirmą aptiktą populiariausią veislę, tačiau veiks nekorektiškai, jei kita veislė turės lygiai tokį patį skaičių šunų. Patobulinkite metodą padarydami taip, kad metodas grąžintų populiariausių veislių sąrašą.

⮋**Penktas žingsnis.**

* Realizuosime besidubliuojančių šunų aptikimą. Pagal sąlygą – besidubliuojantis šuo registre yra tas, kuris turi tą patį mikroschemos identifikatorių ir vardą.
* Dėl to tokiam specifiniam šunų palyginimui pirmiausiai perdengsime Equals metodą:

// Rekomenduojama, kad bool Equals( object obj ) tik kviestų specifinio tipo (*type-specific*) Equals metodą, todėl palyginimą realizuosime kitame (Equals) metode.

public override bool Equals(object obj)

{

return this.Equals(obj as Dog);

}

* Perdengę Equals metodą realizuosime specifinio tipo Equals metodą:

// Čia *type-specific* Equals metodas.

/\*

\* Realizuojant Equals, reikia laikytis šių taisyklių:

\* 1) x.Equals(x) returns true. This is called the reflexive property.

\* 2) x.Equals(y) returns the same value as y.Equals(x). This is called the symmetric property.

\* 3)if (x.Equals(y) && y.Equals(z)) returns true, then x.Equals(z) returns true. This is called the transitive property.

\* 4) Successive invocations of x.Equals(y) return the same value as long as the objects referenced by x and y are not modified.

\* 5) x.Equals(null) returns false. However, null.Equals(null) throws an exception; it does not obey rule number two above.

\*/

public bool Equals(Dog dog)

{

//tikrina, ar objektas egzistuoja

if (Object.ReferenceEquals(dog, null))

{

return false;

}

//Tikrina, ar tokia pati klasė (reikia tam atvejui, jei Dog klasę paveldėtų tarkim klasės Buldogas ir Pudelis)

if (this.GetType() != dog.GetType())

return false;

// Grąžiname true, jei objektų laukai (savybės) sutampa.

return (ChipId == dog.ChipId) && (Name == dog.Name);

}

* Realizavus Equals metodą reikia atkreipti dėmesį, kad griežtai rekomenduojama perdengti GetHashCode metodą, kuris naudojamas palyginimo optimizavimui.

/\*

\* Also, it is STRONGLY recommended that any class that overrides Equals also override System.Object.GetHashCode.

\* Override Object.GetHashCode so that two objects that have value equality produce the same hash code.

\*/

public override int GetHashCode()

{

//geriausia gražinti hash code, kuris yra Equals lyginamų objekto sąvybių funkcija (mūsų atveju ChipId ir Name)

return ChipId.GetHashCode() ^ Name.GetHashCode();

}

Vienodi Equals objektai visada turi grąžinti tokį pat *hash* kodą. Atkreipkite dėmesį į tai, kad grąžiname ChipId ir Name *hash* kodų sankirtą, tai yra – *hash* kodą sukuriame iš visų palyginime esančių objektų savybių.

* Perdengiame palyginimo operatorius == ir != tam, kad objektus galėtume palyginti ne tik su metodu Equals, bet ir su įprastais palyginimo operatoriais.

/\*

\* Optional but recommended: Overload the == and != operators

\*/

public static bool operator ==(Dog lhs, Dog rhs)

{

// Patikriname kairę pusę (ar egzistuoja objektas).

// Negalima naudoti lhs==null. Amžinas ciklas.

if (Object.ReferenceEquals(lhs, null))

{

if (Object.ReferenceEquals(rhs, null))

{

// jei objektas neegzistuoja nei kairėje nei dešinėje

// palyginimo operatoriaus pusėje, grąžiname true (null == null = true).

return true;

}

// Jei objektas neegzistuoja tik kairėje pusėje.

return false;

}

// Equals metodas padengia kitus atvejus.

return lhs.Equals(rhs);

}

public static bool operator !=(Dog lhs, Dog rhs)

{

return !(lhs == rhs);

}

Atkreipkite dėmesį į tai kaip veikia visas šis mechanizmas:

* Sakykime, turime tokį kodą: *if (dog1 != dog2) ...*
* Vykdant jį pirmiausia iškviečiamas užklotas operatoriaus != metodas.
* Šio metodo viduje iškviečiamas užklotas operatoriaus == metodas.
* Pastarajame metode iškviečiamas Equals metodas, kuris palygina objektus pagal duotas taisykles.
* Dabar esame pasiruošę realizuoti besidubliuojančių šunų metodą:

private static Dog[] GetDoublePlacedDogs(Branch branch1, Branch branch2, out int doublePlacedDogsCount)

{

Dog[] doublePlacedDogs = new Dog[MaxNumberOfDogs];

doublePlacedDogsCount = 0;

for (int i = 0; i < branch1.DogCount; i++)

{

if (branch2.Dogs.Contains(branch1.Dogs[i]))

{

doublePlacedDogs[doublePlacedDogsCount] = branch1.Dogs[i];

doublePlacedDogsCount++;

}

}

return doublePlacedDogs;

}

Metodas Contains naudoja mūsų perdengtą metodą, kuris lygina šunis pagal mikroschemos identifikatorių ir vardą.

* Main() metode realizuojame besidubliuojančių šunų suradimą Vilniaus ir Kauno registre panaudodami GetDoublePlacedDogs metodą:

Console.WriteLine("Dvigubai užregistruoti šunys");

Console.WriteLine();

Console.WriteLine("Vilniuje ir Kaune:");

int doublePlacedDogsCount;

Dog[] doublePlacedDogs = GetDoublePlacedDogs(branches[1], branches[0], out doublePlacedDogsCount);

PrintDogsToConsole(doublePlacedDogs, doublePlacedDogsCount);

⮋ **Šeštas žingsnis.**

* Realizuosime besidubliuojančių šunų pašalinimą iš nurodyto filialo. Filiale registruoto šuns pašalinimas yra filialo klasės atsakomybė, todėl į filialo klasę (Branch) įdedame metodą RemoveDog:

public void RemoveDog(Dog dog)

{

int i = 0;

while (i < DogCount)

{

if(Dogs[i] == dog) // naudoja užklotą == operatorių

{

DogCount--;

for (int j = i; j < DogCount; j++)

{

Dogs[j] = Dogs[j + 1];

}

}

break;

}

}

* Dabar esame pasiruošę sukurti besidubliuojančių šunų šalinimo iš filialo metodą (pagrindinėje programoje):

private static void RemoveDoublePlacedDogs(Branch branch, Dog[] doublePlacedDogs, int doublePlacedDogsCount)

{

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

{

branch.RemoveDog(doublePlacedDogs[i]);

}

}

* Pagrindiniame (*Main*) metode iškviečiame besidubliuojančių šunų pašalinimą iš Vilniaus filialo registro:

Console.WriteLine();

Console.WriteLine("Sąrašas, iš Vilniaus registro pašalinus besikartojančius:");

Console.WriteLine();

RemoveDoublePlacedDogs(branches[1], doublePlacedDogs, doublePlacedDogsCount);

PrintDogsToConsole(branches[1].Dogs, branches[1].DogCount);

⮋ **Septintas žingsnis.**

* Realizuosime rikiavimo šunų rikiavimą. Filialų šunų sąrašo rikiavimas yra Filialos klasės atsakomybė, todėl šunų sąrašo rikiavimą realizuojame filialo (Branch) klasėje:

public void SortDogs()

{

for (int i = 0; i < DogCount - 1; i++)

{

Dog minValueDog = Dogs[i];

int minValueIndex = i;

for (int j = i + 1; j < DogCount; j++)

if (Dogs[j] <= minValueDog)

{

minValueDog = Dogs[j];

minValueIndex = j;

}

Dogs[minValueIndex] = Dogs[i];

Dogs[i] = minValueDog;

}

}

Čia realizuotas vadinamasis *min-max* rikiavimo metodas. Kol kas nesirūpiname kokias šuns savybes lyginame nuspręsdami kuris šuo turi atsidurti aukščiau ar žemiau sąraše. C# kalba to žinoma negali nuspręsti už jus, todėl rodys klaidą. Mes užklosime <= ir >= operatorius parodydami kaip mes norime palyginti šunis.

* Šuns klasėje (Dog) realizuokite užklotų operatorių (<= ir >=) metodus. Atkreipkite dėmesį į tai, kad reikia realizuoti abu šiuos operatorius, kitaip – rodys klaidą. Lyginsime mikroschemos identifikatorių, nes taip reikalauja sąlyga.

public static bool operator <=(Dog lhs, Dog rhs)

{

return (lhs.ChipId <= rhs.ChipId);

}

public static bool operator >=(Dog lhs, Dog rhs)

{

return (lhs.ChipId >= rhs.ChipId);

}

* Pagrindiniame (*Main*) metode iškviečiame Kauno filialo šunų rikiavimą ir pateikiame rezultatus:

Console.WriteLine();

Console.WriteLine("Surūšiuotas Kauno šunų sąrašas:");

Console.WriteLine();

branches[0].SortDogs();

PrintDogsToConsole(branches[0].Dogs, branches[0].DogCount);

* **Pabaigai**. Parodėme kaip realizuoti operatorių užklojimą. **Nepiktnaudžiaukite** šia programavimo technika. Operatorių užklojimas įveda lankstumo, tačiau **sumažina programos skaitomumą**. Naudokite tik tuomet kai objektų tarpusavio palyginimas yra tikrai komplikuotas.

## Konteinerinės klasės objektų rinkiniams naudojant sąrašus

**Programos kūrimo eiga.**

* Jei naudosite pirmame laboratoriniame darbe sukurtą projektą, kurį modifikuosime ir papildysime nauju funkcionalumu:
  + Nukopijuokite ankstesnio laboratorinio darbo visą sprendimą (*solution*) į atskirą katalogą ir pavadinkite jį **Lab2**.
* Jei norite (nebūtina) – galite pritaikyti sprendimo struktūrą prie antro laboratorinio darbo
  + Pakeiskite sprendimo projektų pavadinimus (pvz. **Lab1.Step2** į **Lab2.Step2**)
  + Pakeiskite *namespace* visuose projektuose taip, kad jie atspindėtų projekto pavadinimą
* Išmeskite iš projekto seną duomenų failą ir sukurkite naujus.
* Filialams sukursime naują klasę Branch, kuri saugos kiekvieno filialo šunų registracijos duomenis.
* Šuns klasėje Dog užklosime metodą ToString() tam, kad suformuotume eilutę išvedimui.
* Pagrindinėje programoje sukursime naujus arba modifikuosime esamus įvedimo ir spausdinimo metodus.
* Realizuojami metodai užduočių sprendimui.

⮋**Pirmas žingsnis.**

* Sukurkite atskirame faile klasę Branch filialuose registruotų šunų duomenims saugoti:

using System.Collections.Generic;

using System.Linq;

namespace Lab2.Step2

{

class Branch

{

public string Town { get; set; }

public Dog[] Dogs { get; set; }

public Branch(string town)

{

Town = town;

Dogs = new List<Dog>();

}

}

}

Kaip matome – filialas turi miesto pavadinimą (*Town*), jame registruotų šunų masyvą (*Dogs*) ir šunų skaitliuką (*DogCount*). Sukuriant filialą konstruktoriui reikia pateikti miesto pavadinimą. Inicializacijos metu filialo objektui priskiriama miesto reikšmė bei sukuriamas (tuščias) šunų sąrašas.

* Realizuokite duomenų nuskaitymą iš failo pagrindinėje programoje:

private static void ReadDogData(string file, Dictionary<string, Branch> branches)

{

string town = null;

using (StreamReader reader = new StreamReader(@file))

{

string line = null;

line = reader.ReadLine();

if (line != null)

{

town = line;

}

Branch branch = branches[town];

while (null != (line = reader.ReadLine()))

{

string[] values = line.Split(',');

string name = values[0];

int chipId = int.Parse(values[1]);

string breed = values[2];

string owner = values[3];

string phone = values[4];

DateTime vd = DateTime.Parse(values[5]);

bool aggressive = bool.Parse(values[6]);

Dog dog = new Dog(name, chipId, breed, owner, phone, vd, aggressive);

//nedėti, jei toks jau yra

if(!branch.Dogs.Contains(dog))

{

branch.Dogs.Add(dog);

}

}

}

}

Atkreipkite dėmesį, kad mums nebereikia papildomo metodo (*GetBranchByTown*). Dictionary struktūra leidžia daug paprasčiau viską realizuoti.

* Padarysime nuskaitytų duomenų spausdinimą. Kad būtų paprasčiau suformuoti šuns duomenų eilutę – *Dog* klasėje perklosime metodą ToString():

public override String ToString()

{

return String.Format("ChipId: {0,5}, Name: {1,10}, Owner: {2,16} ({3}), Last vaccination date: {4:yyyy-MM-dd}", ChipId, Name, Owner, Phone, VaccinationDate);

}

Atkreipkite dėmesį į metodo aprašo eilutę. Direktyva override nusako, jog perrašomas standartinis ToString() metodas.

Dabar modifikuosime PrintDogsToConsole metodą, kuris į konsolę išrašo pateiktų šunų sąrašą:

static void PrintDogsToConsole(List<Dog> dogs)

{

foreach (Dog dog in dogs)

{

Console.WriteLine("{0}", dog.ToString());

}

}

* Main() metode organizuokite Vilniaus, Kauno ir Šiaulių filialų nuskaitymą ir Vilniaus registro duomenų išvedimą į konsolę. Jei modifikuojate ankstesnę programą – pašalinkite ankstesnį šio metodo turinį.

using System;

using System.IO;

namespace Lab2.Step2

{

class Program

{

static void Main(string[] args)

{

Dictionary<string, Branch> branches = new Dictionary<string, Branch>();

branches.Add("Kaunas", new Branch("Kaunas"));

branches.Add("Vilnius", new Branch("Vilnius"));

branches.Add("Šiauliai", new Branch("Šiauliai"));

string[] filePaths = Directory.GetFiles(Directory.GetCurrentDirectory(), "\*.csv");

foreach(string path in filePaths)

{

ReadDogData(path, branches);

}

Console.WriteLine("Vilniuje užregistruoti šunys:");

PrintDogsToConsole(branches["Vilnius"].Dogs);

}

}

}

⮋**Antras žingsnis.**

* Pasinaudosime pirmame laboratoriniame darbe sukurtu metodu GetBreeds tam, kad atspausdinti tam tikruose filialuose užregistruotus šunis. Metodas lieka toks pats, tiesiog paduosime reikalingo (šiuo atveju – Šiaulių) filialo šunis. Main() metode įterpkite šį kodą:

Console.WriteLine("Šiauliuose registruotų šunų veislės:");

List<string> breeds = GetBreeds(branches["Šiauliai"].Dogs);

foreach (string breed in breeds)

{

Console.WriteLine(breed);

}

⮋**Trečias žingsnis.**

* Sukurkite metodą, kuris skaičiuotų agresyvių šunų kiekį pateiktame šunų masyve. Pirmame darbe mes buvome sukūrę metodą, kuris suskaičiuoja agresyvius tam tikro amžiaus šunis. Šį metodą galite palikti - sukursime naują metodą tuo pačiu pavadinimu, tačiau skirtingu parametrų rinkinių. Tuo būdu perdengsime seną metodą ir esant reikalui galėsite naudoti abu metodus.

private static int CountAggressive(List<Dog> dogs)

{

int counter = 0;

foreach (Dog dog in dogs)

{

if (dog.Aggressive.Equals(true))

{

counter++;

}

}

return counter;

}

* Iškvieskite šį metodą pagrindinėje programoje ir atspausdinkite rezultatą:

Console.WriteLine("Agresyviausi Kauno šunys: {0}", CountAggressive(branches["Šiauliai"].Dogs));

⮋**Ketvirtas žingsnis.**

* Parašykite metodą, kuris suranda populiariausią veislę šunų sąraše:

private static string GetMostPopularBreed(List<Dog> dogs)

{

String popular = "not found";

int count = 0;

List<string> breeds = GetBreeds(dogs);

foreach(string breed in breeds)

{

int currentCount = FilterByBreed(dogs, breed).Count;

if (currentCount > count)

{

popular = breed;

count = currentCount;

}

}

return popular;

}

* Panaudokite šį metodą, surasdami ir pateikdami populiariausią Vilniaus miesto veislę:

Console.WriteLine("Populiariausia veislė Vilniuje: {0}", GetMostPopularBreed(branches["Vilnius"].Dogs));

* Galbūt pastebėjote, jog šis metodas grąžins pirmą aptiktą populiariausią veislę, tačiau veiks nekorektiškai, jei kita veislė turės lygiai tokį patį skaičių šunų. Patobulinkite metodą padarydami taip, kad metodas grąžintų populiariausių veislių sąrašą.

⮋**Penktas žingsnis.**

* Realizuosime besidubliuojančių šunų aptikimą. Pagal sąlygą – besidubliuojantis šuo registre yra tas, kuris turi tą patį mikroschemos identifikatorių ir vardą.
* Dėl to tokiam specifiniam šunų palyginimui pirmiausiai perdengsime Equals metodą:

// Rekomenduojama, kad bool Equals( object obj ) tik kviestų specifinio tipo (*type-specific*) Equals metodą, todėl palyginimą realizuosime kitame (Equals) metode.

public override bool Equals(object obj)

{

return this.Equals(obj as Dog);

}

* Perdengę Equals metodą realizuosime specifinio tipo Equals metodą:

// Čia *type-specific* Equals metodas.

/\*

\* Realizuojant Equals, reikia laikytis šių taisyklių:

\* 1) x.Equals(x) returns true. This is called the reflexive property.

\* 2) x.Equals(y) returns the same value as y.Equals(x). This is called the symmetric property.

\* 3)if (x.Equals(y) && y.Equals(z)) returns true, then x.Equals(z) returns true. This is called the transitive property.

\* 4) Successive invocations of x.Equals(y) return the same value as long as the objects referenced by x and y are not modified.

\* 5) x.Equals(null) returns false. However, null.Equals(null) throws an exception; it does not obey rule number two above.

\*/

public bool Equals(Dog dog)

{

//tikrina, ar objektas egzistuoja

if (Object.ReferenceEquals(dog, null))

{

return false;

}

//Tikrina, ar tokia pati klasė (reikia tam atvejui, jei Dog klasę paveldėtų tarkim klasės Buldogas ir Pudelis)

if (this.GetType() != dog.GetType())

return false;

// Grąžiname true, jei objektų laukai (savybės) sutampa.

return (ChipId == dog.ChipId) && (Name == dog.Name);

}

* Realizavus Equals metodą reikia atkreipti dėmesį, kad griežtai rekomenduojama perdengti GetHashCode metodą, kuris naudojamas palyginimo optimizavimui.

/\*

\* Also, it is STRONGLY recommended that any class that overrides Equals also override System.Object.GetHashCode.

\* Override Object.GetHashCode so that two objects that have value equality produce the same hash code.

\*/

public override int GetHashCode()

{

//geriausia gražinti hash code, kuris yra Equals lyginamų objekto sąvybių funkcija (mūsų atveju ChipId ir Name)

return ChipId.GetHashCode() ^ Name.GetHashCode();

}

Vienodi Equals objektai visada turi grąžinti tokį pat *hash* kodą. Atkreipkite dėmesį į tai, kad grąžiname ChipId ir Name *hash* kodų sankirtą, tai yra – *hash* kodą sukuriame iš visų palyginime esančių objektų savybių.

* Perdengiame palyginimo operatorius == ir != tam, kad objektus galėtume palyginti ne tik su metodu Equals, bet ir su įprastais palyginimo operatoriais.

/\*

\* Optional but recommended: Overload the == and != operators

\*/

public static bool operator ==(Dog lhs, Dog rhs)

{

// Patikriname kairę pusę (ar egzistuoja objektas).

// Negalima naudoti lhs==null. Amžinas ciklas.

if (Object.ReferenceEquals(lhs, null))

{

if (Object.ReferenceEquals(rhs, null))

{

// jei objektas neegzistuoja nei kairėje nei dešinėje

// palyginimo operatoriaus pusėje, grąžiname true (null == null = true).

return true;

}

// Jei objektas neegzistuoja tik kairėje pusėje.

return false;

}

// Equals metodas padengia kitus atvejus.

return lhs.Equals(rhs);

}

public static bool operator !=(Dog lhs, Dog rhs)

{

return !(lhs == rhs);

}

Atkreipkite dėmesį į tai kaip veikia visas šis mechanizmas:

* Sakykime, turime tokį kodą: *if (dog1 != dog2) ...*
* Vykdant jį pirmiausia iškviečiamas užklotas operatoriaus != metodas.
* Šio metodo viduje iškviečiamas užklotas operatoriaus == metodas.
* Pastarajame metode iškviečiamas Equals metodas, kuris palygina objektus pagal duotas taisykles.
* Dabar esame pasiruošę realizuoti besidubliuojančių šunų metodą:

//prieš realizuojant šį metodą, realizuoti Equals

private static List<Dog> GetDoublePlacedDogs(Branch b1, Branch b2)

{

List<Dog> results = new List<Dog>();

foreach (Dog dog in b1.Dogs)

{

if (b2.Dogs.Contains(dog))

{

results.Add(dog);

}

}

return results;

}

Metodas Contains naudoja mūsų perdengtą metodą, kuris lygina šunis pagal mikroschemos identifikatorių ir vardą.

* Main() metode realizuojame besidubliuojančių šunų suradimą Vilniaus ir Kauno registre panaudodami GetDoublePlacedDogs metodą:

Console.WriteLine();

Console.WriteLine("Dvigubai užregistruoti šunys");

Console.WriteLine();

Console.WriteLine("Vilniuje ir Kaune:");

List<Dog> doublePlacedDogs = GetDoublePlacedDogs(branches["Vilnius"], branches["Kaunas"]);

PrintDogsToConsole(doublePlacedDogs);

⮋ **Šeštas žingsnis.**

* Realizuoti *RemoveDog* metodo šį kartą nebereikia, nes *List* struktūra jau turi tam parengtus metodus.
* Sukuriame besidubliuojančių šunų šalinimo iš filialo metodą (pagrindinėje programoje):

private static void RemoveDoublePlacedDogs(Branch branch, List<Dog> doublePlacedDogs)

{

foreach (Dog dog in doublePlacedDogs)

{

branch.Dogs.Remove(dog);

}

}

* Pagrindiniame (*Main*) metode iškviečiame besidubliuojančių šnų pašalinimą iš Vilniaus filialo registro:

Console.WriteLine();

Console.WriteLine("Sąrašas, iš Vilniaus registro pašalinus besikartojančius:");

Console.WriteLine();

RemoveDoublePlacedDogs(branches["Vilnius"], doublePlacedDogs);

PrintDogsToConsole(branches["Vilnius"].Dogs);

⮋ **Septintas žingsnis.**

* Realizuosime rikiavimo šunų rikiavimą. Filialų šunų sąrašo rikiavimas yra Filialo klasės atsakomybė, todėl šunų sąrašo rikiavimą realizuojame filialo (Branch) klasėje:

public void SortDogs()

{

Dogs = Dogs.OrderBy(o => o.ChipId).ToList();

}

Daug nesikankiname ir pasinaudojome *Linq*, nes visi kiti struktūros *List* rūšiavimo būdai yra gana griozdiški.

* Pagrindiniame (*Main*) metode iškviečiame Kauno filialo šunų rikiavimą ir pateikiame rezultatus:

Console.WriteLine();

Console.WriteLine("Surūšiuotas Kauno šunų sąrašas:");

Console.WriteLine();

branches["Kaunas"].SortDogs();

PrintDogsToConsole(branches["Kaunas"].Dogs);

* **Pabaigai**. Atkreipkite dėmesį, kaip supaprastėjo programavimas ir sumažėjo klaidų galimybė naudojant *List* struktūrą. Realiuose projektuose tai ir naudokite. Masyvų programavimu užsiimkite tik retais atvejais, kai reikia itin aukšto programos veikimo našumo.

## Konteinerinės klasės objektų rinkiniams naudojant Linq (Lambda išraiškas)

**Programos kūrimo eiga.**

* Jei naudosite pirmame laboratoriniame darbe sukurtą projektą, kurį modifikuosime ir papildysime nauju funkcionalumu:
  + Nukopijuokite ankstesnio laboratorinio darbo visą sprendimą (*solution*) į atskirą katalogą ir pavadinkite jį **Lab2**.
* Jei norite (nebūtina) – galite pritaikyti sprendimo struktūrą prie antro laboratorinio darbo
  + Pakeiskite sprendimo projektų pavadinimus (pvz. **Lab1.Step3** į **Lab2.Step3**)
  + Pakeiskite *namespace* visuose projektuose taip, kad jie atspindėtų projekto pavadinimą
* Išmeskite iš projekto seną duomenų failą ir sukurkite naujus.
* Filialams sukursime naują klasę Branch, kuri saugos kiekvieno filialo šunų registracijos duomenis.
* Šuns klasėje Dog užklosime metodą ToString() tam, kad suformuotume eilutę išvedimui.
* Pagrindinėje programoje sukursime naujus arba modifikuosime esamus įvedimo ir spausdinimo metodus.
* Realizuojami metodai užduočių sprendimui.

⮋**Pirmas žingsnis.**

* Sukurkite atskirame faile klasę Branch filialuose registruotų šunų duomenims saugoti:

using System.Collections.Generic;

using System.Linq;

namespace Lab2.Step3

{

class Branch

{

public string Town { get; set; }

public Dog[] Dogs { get; set; }

public Branch(string town)

{

Town = town;

Dogs = new List<Dog>();

}

}

}

Kaip matome – filialas turi miesto pavadinimą (*Town*), jame registruotų šunų masyvą (*Dogs*) ir šunų skaitliuką (*DogCount*). Sukuriant filialą konstruktoriui reikia pateikti miesto pavadinimą. Inicializacijos metu filialo objektui priskiriama miesto reikšmė bei sukuriamas (tuščias) šunų sąrašas.

* Realizuokite duomenų nuskaitymą iš failo pagrindinėje programoje:

private static void ReadDogData(string file, Dictionary<string, Branch> branches)

{

string town = null;

using (StreamReader reader = new StreamReader(@file))

{

string line = null;

line = reader.ReadLine();

if (line != null)

{

town = line;

}

Branch branch = branches[town];

while (null != (line = reader.ReadLine()))

{

string[] values = line.Split(',');

string name = values[0];

int chipId = int.Parse(values[1]);

string breed = values[2];

string owner = values[3];

string phone = values[4];

DateTime vd = DateTime.Parse(values[5]);

bool aggressive = bool.Parse(values[6]);

Dog dog = new Dog(name, chipId, breed, owner, phone, vd, aggressive);

//nedėti, jei toks jau yra

if(!branch.Dogs.Contains(dog))

{

branch.Dogs.Add(dog);

}

}

}

}

Atkreipkite dėmesį, kad mums nebereikia papildomo metodo (*GetBranchByTown*). Dictionary struktūra leidžia daug paprasčiau viską realizuoti.

* Padarysime nuskaitytų duomenų spausdinimą. Kad būtų paprasčiau suformuoti šuns duomenų eilutę – *Dog* klasėje perklosime metodą ToString():

public override String ToString()

{

return String.Format("ChipId: {0,5}, Name: {1,10}, Owner: {2,16} ({3}), Last vaccination date: {4:yyyy-MM-dd}", ChipId, Name, Owner, Phone, VaccinationDate);

}

Atkreipkite dėmesį į metodo aprašo eilutę. Direktyva override nusako, jog perrašomas standartinis ToString() metodas.

Dabar modifikuosime PrintDogsToConsole metodą, kuris į konsolę išrašo pateiktų šunų sąrašą:

private static void PrintDogsToConsole(List<Dog> dogs)

{

dogs.ForEach(d => Console.WriteLine("{0}", d.ToString()));

}

* Main() metode organizuokite Vilniaus, Kauno ir Šiaulių filialų nuskaitymą ir Vilniaus registro duomenų išvedimą į konsolę. Jei modifikuojate ankstesnę programą – pašalinkite ankstesnį šio metodo turinį.

using System;

using System.IO;

namespace Lab2.Step3

{

class Program

{

static void Main(string[] args)

{

Dictionary<string, Branch> branches = new Dictionary<string, Branch>();

branches.Add("Kaunas", new Branch("Kaunas"));

branches.Add("Vilnius", new Branch("Vilnius"));

branches.Add("Šiauliai", new Branch("Šiauliai"));

string[] filePaths = Directory.GetFiles(Directory.GetCurrentDirectory(), "\*.csv");

foreach(string path in filePaths)

{

ReadDogData(path, branches);

}

Console.WriteLine("Vilniuje užregistruoti šunys:");

PrintDogsToConsole(branches["Vilnius"].Dogs);

}

}

}

⮋**Antras žingsnis.**

* Pasinaudosime *Linq* galimybėmis tam, kad elegantiškiau užrašyti GetBreeds metodą:

private static List<string> GetBreeds(List<Dog> dogs)

{

return dogs.Select(o => o.Breed).Distinct().ToList();

}

* Iškviesime šį metodą ir atspausdinsime rezultatus. Main() metode įterpkite šį kodą:

Console.WriteLine("Šiauliuose registruotų šunų veislės:");

List<string> breeds = GetBreeds(branches["Šiauliai"].Dogs);

foreach (string breed in breeds)

{

Console.WriteLine(breed);

}

⮋**Trečias žingsnis.**

* Sukurkite metodą, kuris skaičiuotų agresyvių šunų kiekį pateiktame šunų masyve. Pirmame darbe mes buvome sukūrę metodą, kuris suskaičiuoja agresyvius tam tikro amžiaus šunis. Šį metodą galite palikti - sukursime naują metodą tuo pačiu pavadinimu, tačiau skirtingu parametrų rinkinių. Tuo būdu perdengsime seną metodą ir esant reikalui galėsite naudoti abu metodus.

private static int CountAggressive(List<Dog> dogs)

{

return dogs.Where(o => o.Aggressive.Equals(true)).Count();

}

* Iškvieskite šį metodą pagrindinėje programoje ir atspausdinkite rezultatą:

Console.WriteLine("Agresyviausi Kauno šunys: {0}", CountAggressive(branches["Šiauliai"].Dogs));

⮋**Ketvirtas žingsnis.**

* Parašykite metodą, kuris suranda populiariausią veislę šunų sąraše:

private static string GetMostPopularBreed(List<Dog> dogs)

{

List<string> popular = dogs.GroupBy(o => o.Breed)

.OrderByDescending(op => op.Count())

.Take(1)

.Select(g => g.Key).ToList();

return popular[0];

}

* Panaudokite šį metodą, surasdami ir pateikdami populiariausią Vilniaus miesto veislę:

Console.WriteLine("Populiariausia veislė Vilniuje: {0}", GetMostPopularBreed(branches["Vilnius"].Dogs));

* Galbūt pastebėjote, jog šis metodas grąžins pirmą aptiktą populiariausią veislę, tačiau veiks nekorektiškai, jei kita veislė turės lygiai tokį patį skaičių šunų. Patobulinkite metodą padarydami taip, kad metodas grąžintų populiariausių veislių sąrašą.

⮋**Penktas žingsnis.**

* Realizuosime besidubliuojančių šunų aptikimą. Pagal sąlygą – besidubliuojantis šuo registre yra tas, kuris turi tą patį mikroschemos identifikatorių ir vardą.
* Dėl to tokiam specifiniam šunų palyginimui pirmiausiai perdengsime Equals metodą:

// Rekomenduojama, kad bool Equals( object obj ) tik kviestų specifinio tipo (*type-specific*) Equals metodą, todėl palyginimą realizuosime kitame (Equals) metode.

public override bool Equals(object obj)

{

return this.Equals(obj as Dog);

}

* Perdengę Equals metodą realizuosime specifinio tipo Equals metodą:

// Čia *type-specific* Equals metodas.

/\*

\* Realizuojant Equals, reikia laikytis šių taisyklių:

\* 1) x.Equals(x) returns true. This is called the reflexive property.

\* 2) x.Equals(y) returns the same value as y.Equals(x). This is called the symmetric property.

\* 3)if (x.Equals(y) && y.Equals(z)) returns true, then x.Equals(z) returns true. This is called the transitive property.

\* 4) Successive invocations of x.Equals(y) return the same value as long as the objects referenced by x and y are not modified.

\* 5) x.Equals(null) returns false. However, null.Equals(null) throws an exception; it does not obey rule number two above.

\*/

public bool Equals(Dog dog)

{

//tikrina, ar objektas egzistuoja

if (Object.ReferenceEquals(dog, null))

{

return false;

}

//Tikrina, ar tokia pati klasė (reikia tam atvejui, jei Dog klasę paveldėtų tarkim klasės Buldogas ir Pudelis)

if (this.GetType() != dog.GetType())

return false;

// Grąžiname true, jei objektų laukai (savybės) sutampa.

return (ChipId == dog.ChipId) && (Name == dog.Name);

}

* Realizavus Equals metodą reikia atkreipti dėmesį, kad griežtai rekomenduojama perdengti GetHashCode metodą, kuris naudojamas palyginimo optimizavimui.

/\*

\* Also, it is STRONGLY recommended that any class that overrides Equals also override System.Object.GetHashCode.

\* Override Object.GetHashCode so that two objects that have value equality produce the same hash code.

\*/

public override int GetHashCode()

{

//geriausia gražinti hash code, kuris yra Equals lyginamų objekto sąvybių funkcija (mūsų atveju ChipId ir Name)

return ChipId.GetHashCode() ^ Name.GetHashCode();

}

Vienodi Equals objektai visada turi grąžinti tokį pat *hash* kodą. Atkreipkite dėmesį į tai, kad grąžiname ChipId ir Name *hash* kodų sankirtą, tai yra – *hash* kodą sukuriame iš visų palyginime esančių objektų savybių.

* Perdengiame palyginimo operatorius == ir != tam, kad objektus galėtume palyginti ne tik su metodu Equals, bet ir su įprastais palyginimo operatoriais.

/\*

\* Optional but recommended: Overload the == and != operators

\*/

public static bool operator ==(Dog lhs, Dog rhs)

{

// Patikriname kairę pusę (ar egzistuoja objektas).

// Negalima naudoti lhs==null. Amžinas ciklas.

if (Object.ReferenceEquals(lhs, null))

{

if (Object.ReferenceEquals(rhs, null))

{

// jei objektas neegzistuoja nei kairėje nei dešinėje

// palyginimo operatoriaus pusėje, grąžiname true (null == null = true).

return true;

}

// Jei objektas neegzistuoja tik kairėje pusėje.

return false;

}

// Equals metodas padengia kitus atvejus.

return lhs.Equals(rhs);

}

public static bool operator !=(Dog lhs, Dog rhs)

{

return !(lhs == rhs);

}

Atkreipkite dėmesį į tai kaip veikia visas šis mechanizmas:

* Sakykime, turime tokį kodą: *if (dog1 != dog2) ...*
* Vykdant jį pirmiausia iškviečiamas užklotas operatoriaus != metodas.
* Šio metodo viduje iškviečiamas užklotas operatoriaus == metodas.
* Pastarajame metode iškviečiamas Equals metodas, kuris palygina objektus pagal duotas taisykles.
* Dabar esame pasiruošę realizuoti besidubliuojančių šunų metodą:

private static List<Dog> GetDoublePlacedDogs(Branch b1, Branch b2)

{

return (b1.Dogs.Intersect(b2.Dogs)).ToList();

}

* Main() metode realizuojame besidubliuojančių šunų suradimą Vilniaus ir Kauno registre panaudodami GetDoublePlacedDogs metodą:

Console.WriteLine();

Console.WriteLine("Dvigubai užregistruoti šunys");

Console.WriteLine();

Console.WriteLine("Vilniuje ir Kaune:");

List<Dog> doublePlacedDogs = GetDoublePlacedDogs(branches["Vilnius"], branches["Kaunas"]);

PrintDogsToConsole(doublePlacedDogs);

⮋ **Šeštas žingsnis.**

* Realizuoti *RemoveDog* metodo šį kartą nebereikia, nes *List* struktūra jau turi tam parengtus metodus.
* Sukuriame besidubliuojančių šunų šalinimo iš filialo metodą (pagrindinėje programoje):

private static void RemoveDoublePlacedDogs(Branch branch, List<Dog> doublePlacedDogs)

{

branch.Dogs = branch.Dogs.Except(doublePlacedDogs).ToList();

}

* Pagrindiniame (*Main*) metode iškviečiame besidubliuojančių šnų pašalinimą iš Vilniaus filialo registro:

Console.WriteLine();

Console.WriteLine("Sąrašas, iš Vilniaus registro pašalinus besikartojančius:");

Console.WriteLine();

RemoveDoublePlacedDogs(branches["Vilnius"], doublePlacedDogs);

PrintDogsToConsole(branches["Vilnius"].Dogs);

⮋ **Septintas žingsnis.**

* Realizuosime rikiavimo šunų rikiavimą. Filialų šunų sąrašo rikiavimas yra Filialo klasės atsakomybė, todėl šunų sąrašo rikiavimą realizuojame filialo (Branch) klasėje:

public void SortDogs()

{

Dogs = Dogs.OrderBy(o => o.ChipId).ToList();

}

Čia tikrai tinka *Linq*, nes visi kiti struktūros *List* rūšiavimo būdai yra gana griozdiški.

* Pagrindiniame (*Main*) metode iškviečiame Kauno filialo šunų rikiavimą ir pateikiame rezultatus:

Console.WriteLine();

Console.WriteLine("Surūšiuotas Kauno šunų sąrašas:");

Console.WriteLine();

branches["Kaunas"].SortDogs();

PrintDogsToConsole(branches["Kaunas"].Dogs);

* **Pabaigai**. Kaip pamatėte – *Linq* realizuoja daugelį duomenų apdorojimo metodų, įtraukiant veiksmus su duomenų aibėmis. Be to, *Linq* leidžia trumpai ir elegantiškai užrašyti duomenų apdorojimo kodą, kas gerina programinio kodo skaitomumą.

**Savarankiško darbo užduotis.**

Pirmoje failo eilutėje nurodytas fakulteto pavadinimas. Tekstiniame faile yra fakulteto studentų žiemos sesijos dalykų įvertinimai. Eilutėje apie studentą yra tokie duomenys: pavardė, vardas, grupė, pažymių kiekis, pažymiai. Nustatykite kiekvienos grupės bendrą mokymosi vidurkį. Rezultatus išspausdinkite surikiuotus mažėjančiai pagal vidurkį ir pagal grupes abėcėliškai.

## Kontroliniai klausimai

1. Kokio raktinio žodžio papildomai reikia, norint skelbti klasėje metodą ToString()? Kokia šio raktinio žodžio prasmė?
2. Kokią klasę vadiname konteinerine klase?
3. Kelių tipų objektų rinkinius galima aprašyti vienoje konteinerinėje klasėje?
4. Kaip vadinamas konteinerinės klasė objektas?
5. Kokie raktiniai žodžiai sudaro užklojamo operatoriaus metodo antraštę?
6. Kiek parametrų turi užklojamos operacijos metodo antraštė?
7. Kokį duomenų tipą aprašo raktinis žodis List?
8. Kuo yra patogus ciklo operatorius foreach?
9. Kokią direktyvą reikia įtraukti, norint naudoti List?

## Užduotys

1. **Orai**

Lietuvos hidrometeorologijos centre kiekvieną dieną kaupiama informacija apie oro stebėjimo duomenis. Pradiniai duomenys duoti tekstiniame faile Stebėjimai.txt: data (metai, mėnuo, diena), valanda, vietovės pavadinimas, rajono pavadinimas, oro temperatūra (°C), kritulių kiekis (mm), oro drėgnumas (%), oro slėgis (mmHg).

Duomenis saugokite objektų rinkinyje. Realizuokite duomenų atrinkimą į kitą tos pačios klasės objektų rinkinį pagal nurodytą požymį (mėnuo, vietovės pavadinimas, rajono pavadinimas, oro temperatūra didesnė/mažesnė nei 0°C, kritulių kiekis didesnis/lygus 0). Sukurtos klasės testavimui parašykite testavimo klasę.

Raskite ir spausdinkite rezultatų faile Rezultatai.txt ir parodykite ekrane:

* kur ir kada buvo užregistruota aukščiausia/žemiausia oro temperatūra;
* kur ir kada buvo užregistruotas didžiausias kritulių kiekis;
* kuris rajonas vidutiniškai yra šilčiausias/šalčiausias;
* kuriame rajone lyja daugiausiai/mažiausiai;
* kurios vietovės orai pagal vidutinę oro temperatūrą / kritulių kiekį mažiausiai skiriasi nuo visos Lietuvos orų;
* kur ir kada buvo užregistruota aukščiausia/žemiausia tam tikro mėnesio (įvedama klaviatūra) temperatūra;
* kokia yra kiekvieno mėnesio vidutinė oro temperatūra / kritulių kiekis;
* kur ir kada užregistruotas didžiausias skirtumas tarp vietovės ir rajono vidutinės temperatūros;
* kur ir kada užregistruotas didžiausias oro slėgio pokytis tarp gretimų oro stebėjimų rezultatų;
* kur ir kada užregistruotas didžiausias oro drėgnumas nelyjant;
* kur ir kada užregistruotas mažiausias oro slėgis esant neigiamai temperatūrai;
* kurioje vietovėje, kada ir kiek laiko ilgiausiai lijo / nelijo;

1. **Skaičiuotė**

Vaikams žaidžiant slėpynes, reikia išskaičiuoti, kas liks nežiūrėti. Skaičiuojant pirmą kartą, pradedama nuo pirmojo eilėje stovinčio vaiko. Pasiekus eilės galą, tęsiama skaičiuotė vėl nuo eilės pradžioje stovinčio vaiko. Vaikas, kuriam tenka pasku­tinis skaičiuotės žodis, išeina iš eilės. Skaičiuojama iš naujo, pradedant nuo sekančio eilėje vaiko. Skaičiavi­mai baigiami, kai eilėje lieka tik vienas vaikas. Jis ir lieka nežiūrėti.

Atbėga Petriukas ir prašo priimti žaisti. Vaikai sutinka, jeigu Petriukas atsistos į eilę taip, kad skaičiavimus baigus jis liktų nežiūrėti.

Parašykite programą, kuri patartų Petriukui, kur reikia atsistoti.

Duomenys yra tekstiniame faile U3.txt. Pirmoje eilutėje užrašytas vaikų skaičius N (N <= 100). Toliau N eilutėse surašyti vaikų vardai po vieną eilutėje. Tai pradinė vaikų eilė. Po to kitoje eilutėje užrašytas vaikams žinomų skaičiuočių skaičius K (K <= 5). Paskutinėje eilutėje užrašyta K skaičių, kurių kiekvienas rodo, kiek kurioje skaičiuotėje yra žodžių.

Pavyzdžiui:

|  |  |
| --- | --- |
| U3.txt |  |
| 5  Rasa  Rita  Rima  Rimas  Romas  3  15 4 10 | Vaikų skaičius  Skaičiuočių skaičius  1-os, 2-os ir 3-ios skaičiuočių žodžių skaičius. |

Rezultatai surašomi į tekstinį failą U3Rez.txt. Čia reikia surašyti vaikų vardus su Petriuko vardu reikia­moje vietoje. Vardus rašyti po vieną eilutėje. Po eilės palikti eilutę, sudarytą iš žvaigždučių.

Pavyzdžiui:

|  |  |
| --- | --- |
| U3rez.txt |  |
| Rasa  Rita  Rima  Petriukas  Rimas  Romas  \*\*\*\*\*\*  Rasa  Rita  Rima  Rimas  Petriukas  Romas  \*\*\*\*\*\*  Rasa  Petriukas  Rita  Rima  RimasRomas  \*\*\*\*\*\* | Pirmoji eilė, kai skaičiuotėje yra 15 žodžių.  Antroji eilė, kai skaičiuotėje yra 4 žodžiai.  Trečioji eilė, kai skaičiuotėje yra 10 žodžių. |

1. **Geometrinės figūros**

Tekstiniame faile DuomU2.txt turime duomenis apie iškilias geometrines figūras (trikampius, keturkampius, penkiakampius, …) ir apskritimus. Duomenys apie vieną figūrą surašyti vienoje eilutėje tokia tvarka: figūros vardas (6 simboliai), toliau pagal laikrodžio rodyklę surašytos figūros kampų koordinatės:

FigūrosVardas x1 y1 x2 y2 x3 y3 x4 y4 x5 y5 …

Minimalus figūros kampų skaičius 3, maksimalus - 10.

Apskritimas faile aprašomas taip:

FigūrosVardas x1 y1 R

Maksimalus figūrų skaičius faile – 100.

Failo DuomU2.txt pavyzdys:

Tr1 1.5 -5.5 5 -1.4 8.5 -5.2

Aps3 5.5 -5.5 10

Kt4 -3.5 -0.5 -3.5 5.4 2.4 5.4 2.4 -0.5

Reikia parašyti dialoginę programą, kuri ekrane pateiktų informaciją apie figūras:

|  |  |
| --- | --- |
| Meniu | Veiksmas |
| KokiuIrKiek | Rasti kokių ir kiek yra figūrų. |
| MaxPerimetras | Rasti didžiausią perimetrą turinčios figūros vardą ir perimetrą. |
| MaxPlotas | Rasti didžiausią plotą turinčios figūros vardą ir plotą. |
| KiekStacTrikampiu | Rasti stačiųjų trikampių, įbrėžtų į apskritimus, kiekį (Tikslumas: 0.000001). |
| KiekKvadratu | Rasti kvadratų, apibrėžtų apie apskritimus, kiekį (Tikslumas: 0.000001). |
| InfoFiguruGrupe | Pateikti informaciją apie pageidaujamą figūrų grupę atskirose eilutėse tokia seka  ( nurodoma grupės kampų skaičius, pvz.: 5; apskritimams - 0 ):   * figūrų kiekis; * kiek yra lygiakraščių figūrų (Tikslumas: 0.000001) (apskritimams: figūrų kiekis); * bendras figūrų perimetras (apskritimams: bendras apskritimų ilgis); * bendras figūrų plotas (apskritimams: bendras apskritimų plotas). |

1. **Domino**

Imami 7 vieno domino rinkinio kauliukai. Vieną domino kauliuką sudaro dvi dalys, kurių kiekvienoje arba nieko nėra (baltas), arba juodi taškai, kurių yra nuo 1 iki 6. Kauliuką galima nusakyti kaip dviženklį skaičių, kurio pirmas skaitmuo nurodo pirmos dalies taškų skaičių, o antrasis – antros. Jeigu dalis tuščia, tai rašomas skaitmuo 0 (nulis). Parašykite programą, kuri sudarytų iš šių 7 kauliukų visas galimas grandines, kai jungiami kauliukai galais su vienodu taškų skaičiumi. Gali būti, kad tokios grandinės visai nėra. Sudarant grandines, kauliukas gali būti apsukamas, t.y. kauliukas 35 gali būti padėtas, kaip 53.

Kauliukų duomenys įvedami iš tekstinio failo Kur3.txt. Čia vienoje eilutėje yra parašyti 7 (septyni) dviženkliai skaičiai. Rezultatus surašyti į tekstinį failą eilutėmis po vieną grandinę. Grandinę sudaro 7 kauliukai, tarp kiekvieno kauliuko (dviženklio skaičiaus) paliekamas vieno tarpo ženklas.

|  |  |
| --- | --- |
| Kur3.txt | Rezultatas |
| 13 01 02 24 14 12 25 | 31 10 02 24 41 12 25  52 21 14 42 20 01 13  31 12 24 41 10 02 25  52 20 01 14 42 21 13  ir t.t. |

1. **Sportininkai**

Vieno laikraščio skaitytojai rinko miesto geriausių sportininkų penketuką iš *n (5 < n < 50)* sportininkų. Visų vertintų sportininkų pavardės ir vardai pagal abėcėlę surašyti faile Pasv31.txt. Kiekvieno sportininko pavardei ir vardui užrašyti – atskira eilutė (iki 30 simbolių). Pirmoje failo eilutėje – *n* reikšmė.

Skaitytojai atsiuntė *m* *(0 < m < 100)* įvertinimų (spor­tininkų penketukų, nurodant sportininko eilės numerį bendrame sąraše). Šie duomenys surašyti faile Pasv32.txt. Vienam penketukui – viena eilutė. Pirmoje failo eilutėje – *m* reikšmė.

Rasti skaitytojų sudarytą geriausių sportininkų penke­tuką (pavardes ir vardus, surikiuotus skiriamos vietos eilės tvarka). Bendras įvertinimas skaičiuojamas tokiu būdu: už pir­mą vietą skiriami 5 taškai, už antrą – 4 taškai, ..., už penktą – 1 taškas.

Ar yra skaitytojas, atspėjęs tikslų sportininkų penketu­ką? Jei taip, spausdinkite to skaitytojo numerį (jo verti­nimų numerį faile Pasv32.txt), jei ne – eilutę “Įžvalgių skaitytojų nėra”. Jei yra keli tokie skaitytojai, spausdinkite pirmojo iš jų numerį.

Visus rezultatus spausdinkite į failą Pasv3Rez.txt.

Pavyzdys:

|  |  |
| --- | --- |
| Pasv31.txt: | Pasv32.txt: |
| 10  Abromaitis Tomas  Antanaitis Antanas  Aranauskas Vytas  Jonaitis Jonas  Petraitis Petras  Petrauskas Antanas  Valaitis Dainius  Valaitis Vytas  Virbickas Antanas  Zokaitis Zigmas | 6  7 1 4 9 5  10 4 3 7 1  8 4 5 1 10  4 7 5 1 10  5 2 7 4 1  4 7 5 1 10 |
| Pasv3Rez.txt | |
| Geriausieji sportininkai:  1) Jonaitis Jonas  2) Valaitis Dainius  3) Petraitis Petras  4) Abromaitis Tomas  5) Zokaitis Zigmas  Įžvalgiausias skaitytojas: 4 | |

1. **Kompiuteriai.**

Kompiuterių ministerijoje yra daug kompiuterių, kurie sunumeruoti eilės tvarka, pradedant 1. Ne visi kompiuteriai tarpusavyje sujungti į vieną tinklą. Parašykite programą, kuri spausdintų kiekvieno potinklio kompiuterių numerius didėjimo tvarka.

**Duomenys**. Pirmoje duomenų failo U3.txt eilutėje yra užrašytas kompiuterių skaičius n (1 ≤ n ≤ 300) ir jungčių skaičius k (1 ≤ k ≤ 300). Failo tolesnėse k eilučių yra užrašytos jungtys po vieną eilutėje. Jungtis nusakoma dviejų sujungtų kompiuterių numeriais.

|  |  |  |
| --- | --- | --- |
| Kompiuterių jungtys | U3.txt | Rezultatai |
|  | 9 6  2 4  1 2  3 8  9 6  5 6  1 3 | 1 2 3 4 8  5 6 9  7 |

**Rezultatai**. Rezultatų faile ir ekrane spausdinkite kiekvieno potinklio kompiuterių numerius didėjimo tvarka, skiriant tarpo simboliu. Vienam potinkliui skiriama viena eilutė.

1

2

3

4

8

9

5

7

6

1. **DEL komanda.**

Reikia nustatyti, ar galima viena DEL komanda pašalinti visus norimus panaikinti failus. DEL komandos formatas:

DEL pakaitos simboliai

Pakaitos simbolius gali sudaryti pilnas failo vardas, turintis nuo 1 iki 8 simbolių, arba vardas ir vardo plėtinys iki 3 simbolių. Taško ženklas „.“ skiria vardą nuo plėtinio. Pakaitos simboliuose galima naudoti ir specialius simbolius „?“ ir „\*“. Simbolis „?“ pakeičia vieną pilno failo vardo simbolį, išskyrus tašką. Simbolis „\*“ pakeičia bet kokią simbolių seką, neturinčią taško, netgi ir tuščią seką. „\*“ gali pasirodyti tik paskutinėje vardo ar plėtinio pozicijoje. Failų vardus ir plėtinius sudaro tik lotyniškos raidės ir skaitmenys.

**Duomenys**. Tekstinio failo ‘*U3.txt*’ pirmoje eilutėje nurodyta K (≤150). K žymi toliau esančių failo vardų kiekį. Kiekvienas failo vardas yra atskiroje eilutėje. Prieš failo vardą yra simbolis „-“ arba simbolis „+“. Simbolis „-“ rodo, kad šį failą reikia šalinti, o simbolis „+“ rodo, kad šio failo šalinti negalima.

**Rezultatai**. Išveskite komandą DEL su suformuotais pakaitos simboliais, jei yra tokia komanda, kuri leidžia pašalinti visus naikinamus failus iš karto. Priešingu atveju išveskite pranešimą „Negalima“.

|  |  |
| --- | --- |
| U3.txt | Rezultatas |
| 3  -BP.EXE  +TURBO.EXE  -BPC.EXE | DEL ?P\*.EXE |

1. **Šachmatai.**

Gamykloje yra daug robotų, kurie pagal iš anksto paruoštą programą gamina šachmatų figūras. Pagamintos figūros patenka į bendrą konteinerį, kurio duomenys automatiškai fiksuojami faile. Reikia parašyti programą, kuri pasakytų, kiek kokių figūrų yra konteineryje, kiek iš jų galima paruošti žaidimo komplektų ir kiek po to liks dar nepanaudotų figūrų.

Žinoma, kad žaidimo komplektą sudaro 16 pėstininkų, 4 bokštai, 4 žirgai, 4 rikiai 2 valdovės ir 2 karaliai. Pusė komplektui reikalingų figūrų bus vėliau nudažytos juodai, nes robotai gamina tik baltos spalvos figūras.

**Duomenys.** Failo U3.txt pirmoje eilutėje yra bendras konteineryje esančių figūrų skaičius n (1 ≤ n ≤ 1000). Kitose eilutėse surašyti bet kokia tvarka figūrų tipai, kurie žymimi tokiais simboliais: p – pėstininkas, b – bokštas, z – žirgas, r – rikis, v – valdovė, k - karalius.

**Rezultatai.** Ekrane ir faile spausdinkite turimų figūrų skaičius pagal jų tipą: tipas ir skaičius, po to – kiek galima suformuoti žaidimo komplektų ir kiek kokių pagal tipą figūrų lieka. Tipų seka tokia: p, b, z, r, v, k

|  |  |
| --- | --- |
| U3.txt | Rezultatai |
| 48  p p b b z z r  r v k v k z  z k p p r r r  p p p p p p p  b b b b b r  r v p p p p  p p p p p p  p p p | Figūrų skaičiai  p 24  b 7  z 4  r 7  v 3  k 3  Komplektų skaičius 1  Liko figūrų  p 8  b 3  z 0  r 3  v 1  k 1 |

1. **Daugiakampiai**

Duota n iškilių daugiakampių (trikampiai, keturkampiai, penkiakampiai, ...) viršūnių koordinatės (x,y) išdėstytos pagal laikrodžio rodyklę. Daugiakampis turi m viršūnių, kur 3 m 10.

Parašykite programą kuri rastų:

a) kiek yra daugiakampių, kurių kiekvieno atskirai visos kraštinės vienodos;

b) didžiausio ploto daugiakampį ir suskaičiuotų jo perimetrą;

c) didžiausio ploto kvadratą ir suskaičiuotų jo perimetrą;

Skaičiavimus atlikti 0.1 tikslumu.

Duomenų tekstiniame faile Trecias.txt pirmoje eilutėje užrašytas sveikasis skaičius reiškia daugiakampių skaičių n (1 n 1000). Kitose n eilutėse yra užrašyti daugiakampių duomenys: daugiakampio pavadinimas (6 simboliai), tarpo simbolis, viršūnių skaičius m ir viršūnių koordinatės (realūs skaičiai) (x1,y1), (x2,y2), ... .

Spausdinkite skaičiavimų rezultatus nurodytu formatu (žiūr. Rezultatų pavyzdys).

|  |  |
| --- | --- |
| Trecias.txt | Rezultatų pavyzdys |
| 6  Ketur1 4 1 1 3 4 6 4 6 1  Trik1 3 8 –1 8 3 11 –1  Penk1 5 8 4 8 7 11 11 14 11 14 4  Sesia1 6 0 5 0 9 4 12 8 12 11 8 11 5  Ketur2 4 2 6 2 10 6 10 6 6  K3 4 -3.5 1 –5 4 –2 5 0 3.5 | a) Daugiakampių su vienodomis kraštinėmis: 1  b) Sesia1, plotas = 65.0, perimetras = 32.0  c) Ketur2, plotas = 16.0, perimetras = 16.0 |

|  |  |
| --- | --- |
| U3.txt | Ekranas |
| 12  1 5.0 13.0 11.0 10.0  2 2.0 5.0 4.0 7.0  3 2.0 10.0 5.0 13.0  4 13.0 11.0 16.0 11.0  5 4.0 1.0 4.0 7.0  6 14.0 11.0 18.0 11.0  7 13.0 11.0 13.0 17.0  8 14.0 3.0 14.0 11.0  9 2.0 5.0 4.0 1.0  10 14.0 3.0 18.0 11.0  11 2.0 10.0 11.0 10.0  12 13.0 17.0 16.0 11.0 | Trikampiai  1 : 1 3 11  2 : 2 5 9  3 : 4 7 12  4 : 6 8 10  Panašūs trikampiai  1 ir 2  3 ir 4 |

1. **Panašūs trikampiai**

Stačiakampėje koordinačių plokštumoje yra nubrėžta m (1 ≤ m ≤ 10) porų panašių trikampių. Šie trikampiai neturi bendrų viršūnių. Jų kraštinės nusakomos eilės numeriu ir galų taškų koordinatėmis (x, y). Tokių kraštinių yra n (n=2⋅3⋅m). Parašykite programą, kuri nustatytų, kuriems trikampiams priklauso kraštinės ir suporuotų panašius trikampius. Trikampiai yra panašūs, jeigu vieno trikampio visos trys kraštinės proporcingos kito trikampio kraštinėms (panašių trikampių proporcijų skirtumas neturi viršyti 1.0⋅10-10).

**Duomenys**. Failo U3.txt pirmoje eilutėje parašytas kraštinių skaičius n. Kitose eilutėse aprašytos kraštinės. Kiekvienoje eilutėje yra po penkis skaičius, atskirtus tarpo simboliu: kraštinės numeris k (1 ≤ k ≤ n), pradžios taško x ir y koordinatės ir pabaigos taško x ir y koordinatės (realieji skaičiai).

**Rezultatai**. Faile ir ekrane spausdinami trikampius sudarančių kraštinių numeriai ir panašių trikampių numeriai. Pirmiausiai programa sunumeruoja sudarytus trikampius ir spausdina juos sudarančių kraštinių numerius. Po to, pasinaudodama trikampių numeriais, atspausdina panašių trikampių poras. Kraštinių, kurios sudaro vieną trikampį, numeriai spausdinami didėjimo tvarka. Trikampiams suteikiami numeriai pirmųjų kraštinių numerių didėjimo tvarka (pirmoji kraštinė visada priklausys pirmajam trikampiui).

1. **Robotas labirinte.**

Duotas neilučių ir m stulpelių dydžio labirintas (n < 60, m < 60). Labirintas turi sienas ir koridorius. Duotojoje labirinto pozicijoje (i, j) siena žymima žvaigždutės simboliu ('\*'), o koridorius – tarpo simboliu (' '). Labirinto atskaitos taškas yra viršutinis kairysis kampas. Eilutės (x ašis) ir stulpeliai (y ašis) numeruojami nuo vieneto. Visas labirintas yra apribotas sienomis. Duotojoje pradinėje pozicijoje (x, y) pastatomas robotas. Robotas gali būti atsuktas į vieną iš krypčių: *N* – šiaurė (robotas žiūri į labirinto viršutiniąją eilutę), *W* – vakarai (robotas žiūri į labirinto dešinįjį stulpelį), *S* – pietūs (robotas žiūri į labirinto apatiniąją eilutę), *E* – rytai (robotas žiūri į labirinto kairįjį stulpelį). Pradinėje pozicijoje robotas stovi atsuktas į šiaurę (*N*). Robotas gali judėti tik pagal pateiktas komandas:

*R* – pasisukti 90 laipsnių pagal laikrodžio rodyklę;

*L* – pasisukti 90 laipsnių prieš laikrodžio rodyklę;

*F*– judėti į priekį per vieną poziciją esama kryptimi, jei dar nepasiekta siena; jei siena ta kryptimi jau pasiekta – nieko nedaryti;

*Q*– baigti judėjimą (spausdinti rezultatus).

Parašykite programą, kuri atspausdintų galutinę roboto poziciją ir jo kryptį baigus robotui judėti pagal pateiktą komandų seką.

**Duomenys**. Tekstinio failo U3.txt pirmoje eilutėje nurodytas labirinto eilučių skaičius n ir stulpelių skaičius m. Tolimesnėse n eilučių surašyta labirinto sienų ir koridorių informacija. Kiekvieną eilutę sudaro m simbolių (stulpelių). Kitoje eilutėje pateikiama pradinė roboto pozicija x ir y. Tolesnėje eilutėje pateikiamos komandos roboto judėjimui. Komandos pateiktos vienoje eilutėje, viena nuo kitos neskiriamos jokiais skyrikliais. Komandų skaičius neviršija 255.

**Rezultatai**. Ekrane ir faile reikia spausdinkite galutinę roboto poziciją ir kryptį. Visi dydžiai vienas nuo kito skiriami vienu tarpo simboliu.

|  |  |
| --- | --- |
| U3.txt | Rezultatai |
| 7 8  \*\*\*\*\*\*\*\*  \* \* \* \*\*  \* \* \*  \* \* \*\* \*  \* \* \* \*  \* \* \*\*  \*\*\*\*\*\*\*\*  3 5  RRFLFFFFRFFRFFQ | 5 6 W |

1. **Bankomatas**

Duotos bankomato išduodamų banknotų vertės ir kiek kokių banknotų bankomatas turi. Eilėje prie bankomato laukia n klientų, kurių kiekvienas nori pasiimti x litų. Yra žinoma, kiek pinigų kiekvienas banko klientas turi savo sąskaitoje.

Parašykite programą, kuri išmokėtų klientų pageidaujamas pinigų sumas galimai mažesniu banknotų skaičiumi. Apskaičiuoti pinigų likutį bankomate: kiek kokių banknotų bankomate liko ir kokia bendra jų piniginė vertė?

**Duomenys.** Tekstinio failo U3.txt pirmoje eilutėje surašytos 6 banknotų nominalų vertės išdėstytos mažėjimo tvarka. Antroje eilutėje – atitinkamai kiekvieno nominalo banknotų skaičius, trečioje eilutėje – eilėje laukiančių klientų skaičius n (1 ≤ n ≤ 100), tolimesnėse n eilučių – kliento sąskaitos numeris (4 simboliai), banke laikoma pinigų suma ir pageidaujama pasiimti pinigų suma. Visi faile esantys skaičiai yra sveikieji.

**Rezultatai.** Spausdinkite:

Kliento sąskaitos numerį, buvusią banke pinigų sumą, minuso ženklą (-), skaičiavimų rezultatą: banknoto vertę, daugybos ženklą (\*), išduotų banknotų skaičių, minuso ženklą (-) ir t.t. Eilutės gale lygybės ženklą ir likusią pinigų sumą. Jeigu pageidaujamos pinigų sumos negalima išmokėti, spausdinkite pranešimą ‘Negalima išmokėti’ (žr. pavyzdį). Paskutinėje eilutėje spausdinkite bankomate likusią pinigų sumą (žr. pavyzdį).

|  |  |
| --- | --- |
| U3.txt | Rezultatai |
| 500 200 100 50 20 10  4 6 8 4 3 5  9  LT21 1500 160  LT22 2660 250  LT23 3555 1720  LT24 966 690  LT25 777 1110  LT26 845 560  LT27 300 60  LT28 850 120  LT29 1000 205 | LT21 1500-100\*1-50\*1-10\*1 = 1340  LT22 2660-200\*1-50\*1 = 2410  LT23 3555-500\*3-200\*1-20\*1 = 1835  LT24 966-500\*1-100\*1-50\*1-20\*2 = 276  LT25 777 1110 Negalima išmokėti  LT26 845-200\*2-100\*1-50\*1-10\*1 = 285  LT27 300 60 Negalima išmokėti  LT28 850-100\*1-10\*2 = 730  LT29 1000 205 Negalima išmokėti  500\*0+200\*2+100\*4+50\*0+20\*0+10\*1 = 810 |

1. **Darbininkai**

Duoti du duomenų failai: U21.txt, U22.txt. Faile U21.txt atskirose eilutėse pateikta tokia informacija: pavardė, vardas, kategorija, brigados numeris (gali būti nurodyti keli darbininkai iš tos pačios brigados). Toliau nurodomos darbo pradžios ir darbo pabaigos valandos šešioms savaitės dienoms: pirmadieniui, antradieniui, trečiadieniui, ketvirtadieniui, penktadieniui, šeštadieniui. Duomenų failo turinio pavyzdys parodytas 3.1 pav.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pavardenis1 | Vardenis1 | 6 | 10 | 8 17 | 9 17 | 10 18 | 8 17 | 8 18 | 9 17 |
| Pavardenis2 | Vardenis2 | 4 | 5 | 8 16 | 10 17 | 10 17 | 10 17 | 9 18 | 9 16 |
| Pavardenis3 | Vardenis3 | 3 | 10 | 9 16 | 8 17 | 10 16 | 9 17 | 10 18 | 10 16 |
| Pavardenis4 | Vardenis4 | 2 | 1 | 10 18 | 8 17 | 9 17 | 8 16 | 10 19 | 8 16 |
| Pavardenis5 | Vardenis5 | 1 | 7 | 10 19 | 8 18 | 8 17 | 8 17 | 9 19 | 8 17 |
| Pavardenis6 | Vardenis6 | 1 | 5 | 10 18 | 9 18 | 8 17 | 9 18 | 8 17 | 9 17 |
| Pavardenis7 | Vardenis7 | 2 | 1 | 9 17 | 8 18 | 8 17 | 8 18 | 8 16 | 8 16 |
| Pavardenis8 | Vardenis8 | 5 | 1 | 8 17 | 8 16 | 9 16 | 9 18 | 9 19 | 8 16 |

3.1 pav. Pirmojo duomenų failo pavyzdys

Faile U22.txt atskirose eilutėse surašyti tokie duomenys: darbininko kategorija, valandinio atlygio tarifas (litais). Duomenų failo pavyzdys parodytas 3.2 pav.

|  |  |
| --- | --- |
| 1 | 1.98 |
| 2 | 2.99 |
| 3 | 4.56 |
| 4 | 6.78 |
| 5 | 7.81 |
| 6 | 9.00 |

3.2 pav. Antrojo duomenų failo pavyzdys

Bendras dienos atlygis gaunamas, sudauginus dirbtas valandas iš atitinkamos kategorijos valandinio atlygio tarifo. Pastaba: dirbant po 17 valandos ir šeštadieniais, valandinio atlygio tarifas padvigubinamas.

Reikia:

— apskaičiuoti vidutinį darbo laiką ir vidutinį darbininkų atlygį atskiromis savaitės dienomis;

— apskaičiuoti kiekvieno darbininko vidutinį dienos uždarbį;

— surasti darbininką, kuris uždirbo už visus daugiausiai; apskaičiuoti jo uždirbtą pinigų sumą;

— nustatyti, kuri brigada uždirbo daugiausiai ir koks uždirbtos pinigų sumos dydis.

1. **Skaičiai - I**

Duotas „keturvietis“ masyvas *K*(*M*), 1 ≤ *M* ≤ 20. Kitaip tariant, yra keturios eilutės (žr. 3.3 pav.), kurių elementai formuojami tokiu būdu:

— pirmosios eilutės elementai: apskaičiuoti plotai trikampių, kurių kraštinių ilgiai yra *a*, *b*, *c*. Reikšmės *a*, *b*, *c* tenkina sąlygą: *a* + *b* > *c* ir *a* + *c* > *b* ir *b* + *c* > *a*. Reikšmės *a*, *b*, *c* imamos iš duomenų failo U2.txt, kurį sudaro trys natūrinių skaičių eilutės. Kiekvienos iš eilučių ilgis (skaičių kiekis) yra lygus *M*;

— antrosios eilutės elementai: skaičiai (elementai) sudaro Fibonačio skaičių seką, t. y. *K*(1) = *K*(2) =1, *K*(*i*) = *K*(*i* − 1) + *K*(*i* − 2), *i* = 3, 4, 5, ...;

— trečiosios eilutės elementai: skaičiai surašyti taip, jog skirtumas tarp gretimų skaičių (tarp esamo skaičiaus ir prieš tai einančio skaičiaus) yra lygus *i*\*3; čia *i* žymi esamo skaičiaus vietą (*i* = 2, 3, 4, ...); pirmasis skaičius yra lygus 100;

— ketvirtosios eilutės elementai: skaitmenys, esantys *i*-osiose (1 ≤ *i* ≤ 20) sekos 10111213141516...9899 pozicijose (pvz., *K*(1) = 1, *K*(2) = 0, *K*(3) = *K*(4) = *K*(5) = 1 ir t. t.).

Atspausdinti kiekvieną iš keturių suformuotų skaičių eilučių. Kiekvienai iš sudarytų skaičių eilučių apskaičiuoti bendrą skaičių sumą, minimalų skaičių, maksimalų skaičių, aritmetinį vidurkį, medianą.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.3 pav. „Keturvietis“ masyvas

1. **Studentų studijos**

Duoti du duomenų failai: U21.txt, U22.txt. Faile U21.txt atskirose eilutėse pateikta tokia informacija: studento pavardė, vardas, studento bilieto numeris, stojimo balas (nuo 0.0 iki 100.0), akademinė grupė (gali būti nurodyti keli studentai iš tos pačios akademinės grupės). Toliau nurodomi atitinkamų disciplinų egzaminų pažymiai (nuo 0 iki 10; įrašas NE reiškia, jog egzaminas nelaikytas). Iš viso nurodomi tokie penki pažymiai: programavimo pažymys, informatikos ir informacinių technologijų pažymys, matematikos pažymys, fizikos pažymys, filosofijos pažymys. Duomenų failo turinio pavyzdys parodytas 3.4 pav.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pavardenis1 | Vardenis1 | B0001 | 30.5 | IF-4/1 | 8 | NE | 5 | 4 | 6 |
| Pavardenis2 | Vardenis2 | B0002 | 25.0 | IF-4/2 | 8 | 9 | 7 | 6 | 8 |
| Pavardenis3 | Vardenis3 | B0003 | 95.0 | IF-4/2 | 10 | 10 | 10 | 10 | 10 |
| Pavardenis4 | Vardenis4 | B0004 | 20.0 | F-4/1 | 8 | 10 | 8 | 7 | 6 |
| Pavardenis5 | Vardenis5 | B0005 | 32.5 | F-4/2 | 7 | 9 | 6 | 5 | 5 |
| Pavardenis6 | Vardenis6 | B0006 | 37.0 | F-4/2 | 6 | 8 | 9 | 6 | 7 |
| Pavardenis7 | Vardenis7 | B0007 | 50.0 | ME-4/1 | 9 | 9 | 10 | 10 | 9 |
| Pavardenis8 | Vardenis8 | B0008 | 90.0 | ME-4/1 | 10 | 9 | 10 | 10 | 9 |
| Pavardenis9 | Vardenis9 | B0009 | 22.5 | EV-4/1 | NE | 6 | 7 | 5 | 6 |

3.4 pav. Pirmojo duomenų failo pavyzdys

Faile U22.txt atskirose eilutėse surašyti tokie duomenys: akademinė grupė, fakulteto pavadinimas. Duomenų failo pavyzdys parodytas 3.5 pav.

|  |  |
| --- | --- |
| IF-4/1 | Informatikos fakultetas |
| IF-4/2 | Informatikos fakultetas |
| F-4/1 | Fundamentaliųjų mokslų fakultetas |
| F-4/2 | Fundamentaliųjų mokslų fakultetas |
| ME-4/1 | Mechatronikos fakultetas |
| EV-4/1 | Ekonomikos ir vadybos fakultetas |

3.5 pav. Antrojo duomenų failo pavyzdys

Reikia:

— apskaičiuoti visų studentų laikytų egzaminų vidutinį pažymį;

— apskaičiuoti kiekvienam studentui stipendijos dydį pagal formulę: [lt], čia *b* žymi stojimo balą, *e* − laikytų egzaminų vidurkį (jei bent vienas egzaminas neišlaikytas, stipendija neskiriama) (pastaba: jeigu visi įvertinimai yra dešimtukai, stipendija didinama 10%);

— surasti discipliną, kurios egzaminą studentams sekėsi laikyti geriausiai; koks šios disciplinos egzamino įvertinimų vidurkis?

— nustatyti, kurios akademinės grupės ir kurio fakulteto studentai yra pažangiausi (geriausiai išlaikė egzaminus).

1. **Skaičiai - II**

Duotas „keturvietis“ masyvas *A*(*N*), 1 ≤ *N* ≤ 20. Kitaip sakant, yra keturios eilutės (žr. 3.6 pav.), kurių elementai formuojami tokiu būdu:

— pirmosios eilutės elementai: duoto natūrinio skaičiaus skaitmenų suma. Natūriniai skaičiai imami iš duomenų failo U21.txt. Natūriniai skaičiai skiriami tarpais, o jų kiekis lygus *N* (1 ≤ *N* ≤ 20);

— antrosios eilutės elementai: skaitmenys, esantys (*i*\*2)-osiose (1 ≤ *i* ≤ 20) sekos 11010010001000010000... pozicijose (pvz., *A*(1) = 1, *A*(2) = 1, *A*(3) = 0, *A*(4) = 1, *A*(5) = 0 ir t. t.);

— trečiosios eilutės elementai: šios eilutės elementai turi būti lygūs duoto teigiamo realaus (su trupmenine dalimi) skaičiaus pirmajam trupmeninės dalies skaitmeniui (pvz., jeigu realus skaičius (*x*) yra lygus 32.697, tai tas skaitmuo yra lygus 6). Realieji skaičiai (*x*1, *x*2, *x*3, ...) imami iš duomenų failo U22.txt. Realiųjų skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys;

— ketvirtosios eilutės elementai: duotos eilutės elementai (*a*1, *a*2, ..., *ai*, ..., *aN*) apskaičiuojami pagal formulę: . Skaičių *a*1, *a*2, ..., *aN* tikslumas − 8 ženklai po kablelio. Skaičiai *x*1, *x*2, ..., *xi*, ..., *xN* imami iš failo U23.txt. Skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys.

Atspausdinti kiekvieną iš keturių suformuotų skaičių eilučių. Kiekvienai iš sudarytų skaičių eilučių apskaičiuoti bendrą skaičių sumą, minimalų skaičių, maksimalų skaičių, aritmetinį vidurkį, medianą.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.6 pav. „Keturvietis“ masyvas

1. **Mokslininkai**

Duoti du duomenų failai: U21.txt, U22.txt. Faile U21.txt atskirose eilutėse pateikta tokia informacija: mokslininko pavardė, vardas, fakultetas (fakulteto kodas) (gali būti nurodyti keli mokslininkai iš to pačio fakulteto), publikacijos žurnalo pavadinimas (viename žurnale gali būti kelios publikacijos), publikavimo data, publikacijos citavimų skaičius. Duomenų failo turinio pavyzdys parodytas 3.7 pav.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Petraitis | Petras | F01 | Operations Research | 2005 | 5 |
| Jonaitis | Jonas | F01 | Informatica | 2006 | 0 |
| Juozaitis | Juozas | F02 | Management Science | 2004 | 10 |
| Mataitis | Matas | F03 | Information Technology | 1999 | 6 |
| Simaitis | Simas | F03 | Operations Research | 2004 | 12 |
| Rimaitis | Rimas | F04 | Operations Research | 2007 | 7 |
| Valaitis | Valius | F04 | Information Technology | 2006 | 3 |
| Saulaitis | Saulius | F04 | Informatica | 2005 | 15 |
| Pranaitis | Pranas | F05 | Informatica | 2004 | 1 |
| Antanaitis | Antanas | F06 | Information Technology | 2005 | 2 |

3.7 pav. Pirmojo duomenų failo pavyzdys

Faile U22.txt atskirose eilutėse surašyti tokie duomenys: fakulteto kodas, universiteto pavadinimas. Duomenų failo pavyzdys parodytas 3.8 pav.

|  |  |
| --- | --- |
| F01 | Kauno technologijos universitetas |
| F02 | Kauno technologijos universitetas |
| F03 | Vilniaus universitetas |
| F04 | Vilniaus universitetas |
| F05 | Klaipėdos universitetas |
| F06 | Šiaulių universitetas |

3.8 pav. Antrojo duomenų failo pavyzdys

Reikia:

— surasti didžiausią ir mažiausią mokslininkų publikacijų citavimų skaičių; nustatyti vidutinį publikacijų citavimų skaičių ir kiek mokslininkų viršija vidurkį;

— apskaičiuoti citavimų kiekį pradedant 2004 ir baigiant 2006 metais;

— surasti universitetą, kurio mokslininkų darbai buvo cituojami didžiausią kartų skaičių;

— kuriuose žurnaluose dažniausiai publikuotos Kauno technologijos ir Vilniaus universitetų mokslininkų publikacijos.

1. **Skaičiai - III**

Duotas „keturvietis“ masyvas *Z*(*M*), 1 ≤ *M* ≤ 20. Kitaip tariant, yra keturios eilutės (žr. 3.9 pav.), kurių elementai formuojami tokiu būdu:

— pirmosios eilutės elementai: esamas faile natūrinis skaičius, parašytas iš kito galo (iš dešinės į kairę) (Pvz., jei duotas skaičius 123, tai turi būti gauta 321.) Natūriniai skaičiai imami iš duomenų failo U21.txt. Natūriniai skaičiai skiriami tarpais, o jų kiekis lygus *M* (1 ≤ *M* ≤ 20);

— antrosios eilutės elementai: skaitmenys, esantys (*i*\*3)-osiose (1 ≤ *i* ≤ 20) sekos 1248163264128256... pozicijose (pvz., *Z*(1) = 4, *Z*(2) = 6, *Z*(3) = 6, *Z*(4) = 2, *Z*(5) = 5 ir t. t.);

— trečiosios eilutės elementai: šios eilutės elementai turi būti lygūs duoto teigiamo realaus (su trupmenine dalimi) skaičiaus antrajam trupmeninės dalies skaitmeniui (pvz., jeigu realus skaičius (*x*) yra lygus 64.891, tai tas skaitmuo yra lygus 9). Realieji skaičiai (*x*1, *x*2, ..., *xM*) imami iš duomenų failo U22.txt. Realiųjų skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys;

— ketvirtosios eilutės elementai: duotos eilutės elementai (*z*1, *z*2, ..., *zi*, ..., *zM*) apskaičiuojami pagal formulę: . Skaičių *z*1, *z*2, ..., *zM* tikslumas − 8 ženklai po kablelio. Skaičiai *x*1, *x*2, ..., *xi*, ..., *xM* imami iš failo U23.txt. Skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys.

Atspausdinti kiekvieną iš keturių suformuotų skaičių eilučių. Kiekvienai iš sudarytų skaičių eilučių apskaičiuoti bendrą skaičių sumą, minimalų skaičių, maksimalų skaičių, aritmetinį vidurkį, medianą.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.9 pav. „Keturvietis“ masyvas

1. **Stačiakampiai**

Stačiakampių padėtis plokštumoje nusakoma jų priešingų (kairiojo-apatinio ir dešiniojo-viršutinio) kampų koordinatėmis (žiūr. 3.10 pav.). Stačiakampių skaičius yra lygus *n*; 2 ≤ *n* ≤ 20. Stačiakampių koordinatės įvedamos iš duomenų failo U2.txt. Stačiakampių kraštinės yra lygiagrečios koordinačių sistemos ašims (abscisei ir ordinatei). Tarsime, jog vienai koordinačių padalai atitinka vienas centimetras.

Reikia sudaryti metodą atstumams tarp stačiakampių kairiųjų-apatinių kampų skaičiuoti. Turi būti skaičiuojamas ir euklidinis, ir vadinamasis Manheteno atstumai. Rasti didžiausią ir mažiausius atstumus. Taip pat turi būti surasti visi atstumai nuo stačiakmapių kairiųjų-apatinių kampų iki koordinačių sistemos pradžios taško (0, 0).

Taip pat reikia apskaičiuoti: a) bendrą visų stačiakampių užimamą plotą, neatsižvelgiant į stačiakampių persidengimus; b) bendrą visų stačiakampių užimamą plotą, atsižvelgiant į tai, jog stačiakampiai gali tarpusavyje persidengti (kaip parodyta paveikslėlyje); c) ilgiausios perimetro linijos ilgį, atsižvelgiant į stačiakampių persidengimus (jei persidengiančių stačiakampių nėra, skaičiuojamas didžiausio stačiakampio perimetras).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

9

8

7

6

5

4

3

2

1

0

1 2 3 4 5 6 7 8 9 ...

3.10 pav. Stačiakampių išsidėstymo pavyzdys

1. **Skaičiai - IV**

Duotas „keturvietis“ masyvas *W*(*N*), 1 ≤ *N* ≤ 20. Kitaip sakant, yra keturios eilutės (žr. 3.11 pav.), kurių elementai formuojami tokiu būdu:

— pirmosios eilutės elementai: elementai apskaičiuojami taip: jeigu visi duoto natūrinio skaičiaus skaitmenys yra skirtingi, tai elementas prilyginamas vienetui (1); priešingu atveju elementas prilyginamas nuliui (0). Natūriniai skaičiai imami iš duomenų failo U21.txt. Natūriniai skaičiai skiriami tarpais, o jų kiekis lygus *N* (1 ≤ *N* ≤ 20);

— antrosios eilutės elementai: skaitmenys, esantys (*i*\*4)-osiose (1 ≤ *i* ≤ 20) sekos 149162536496481100121144... pozicijose (pvz., *W*(1) = 1, *W*(2) = 3, *W*(3) = 6, *W*(4) = 1, *W*(5) = 2 ir t. t.);

— trečiosios eilutės elementai: šios eilutės elementai turi būti lygūs duoto teigiamo realaus (su trupmenine dalimi) skaičiaus priešpaskutiniam sveikosios dalies skaitmeniui (pvz., jeigu realus skaičius (*x*) yra lygus 132.697, tai tas skaitmuo yra lygus 3). Realieji skaičiai (*x*1, *x*2, ..., *xN*) imami iš duomenų failo U22.txt. Realiųjų skaičių sveikąją dalį sudaro ne daugiau kaip 10 skaitmenų;

— ketvirtosios eilutės elementai: duotos eilutės elementai (*w*1, *w*2, ..., *wi*, ..., *wN*) apskaičiuojami pagal formulę: . Skaičių *w*1, *w*2, ..., *wN* tikslumas − 8 ženklai po kablelio. Skaičiai *x*1, *x*2, ..., *xi*, ..., *xN* imami iš failo U23.txt. Skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys.

Atspausdinti kiekvieną iš keturių suformuotų skaičių eilučių. Kiekvienai iš sudarytų skaičių eilučių apskaičiuoti bendrą skaičių sumą, minimalų skaičių, maksimalų skaičių, aritmetinį vidurkį, medianą.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.11 pav. „Keturvietis“ masyvas

1. **„Vaizdo eilutės“ apdorojimas**

Duomenų faile U2.txt duota: *n* − natūrinis skaičius, nusakantis realių skaitmenų eilutės (masyvo) dydį (ilgį). Kitoje failo eilutėje pateikiamas tam tikru būdu užkoduotas vaizdas („vaizdo eilutė“). Vaizdas užkoduotas taip, kad atskiriems skaičiams atitinka tam tikri (pseudo)grafiniai simboliai: 0 atitinka tarpo simbolis („  “) (balta spalva) (simbolio ASCII kodas: 32); 1 − simbolis „░“ (kodas: 176); 2 − simbolis „▒“ (kodas: 177); 3 − simbolis „▓“ (kodas: 178); 4 − simbolis „█“ (juoda spalva) (kodas: 219). Eilutės su užkoduotu vaizdu pavyzdys pateiktas 3.12 paveikslėlyje.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ▓ | ▒ | ░ |  | ░ | ▓ | █ |

3.12 pav. „Vaizdo eilutės“ pavyzdys

Vaizdo apdorojimas atliekamas trimis etapais (žingsniais). Iš pradžių sudaroma pradinio eilutės reikšmių apdorojimo procedūra, kuri suvidurkina eilutės reikšmes pagal tokią taisyklę, jog centrinė (vidurinė) reikšmė pakeičiama gretimų (kaimyninių) reikšmių (įskaitant pačią reikšmę) − viso trijų reikšmių − aritmetiniu vidurkiu (žiūr. 3.13 pav.). Šio pobūdžio suvidurkinimas atliekamas visiems galimiems tokiems eilutės „trejetukams“.

|  |  |  |
| --- | --- | --- |
| 3 | 4 | 2 |

|  |  |  |
| --- | --- | --- |
| 3 | **3** | 1 |

3.13 pav. Eilutės reikšmių vidurkinimas

Antrojo etapo metu atliekamas eilutės reikšmių „diferencijavimas“ tokiu principu. Nagrinėjami iš eilės visi eilutės reikšmių „dvejetukai“ (*a*, *b*). Viena iš dvejetuko (*a*, *b*) reikšmių, pvz., *a* pakeičiama tokia reikšme: . Tokio reikšmių diferencijavimo tikslas − palengvinti vaizdo kontrastingų (šviesesnių-tamsesnių) dalių identifikavimą.

Trečiajame etape vykdomas papildomas reikšmių diferencijavimas („filtravimas“), kurio metu gaunama dvejetainio pavidalo eilutė, sudaryta tik iš reikšmių 0 ir 1. Tai galima atlikti tokiu nesudėtingu būdu. Patikrinamos visos eilutės reikšmės. Jeigu duota reikšmė yra mažesnė (arba lygi) už pasirinktą slenkstinę reikšmę (slenkstį), tai ši eilutės reikšmė prilyginama 0; priešingu atveju reikšmė prilyginama 1 (slenksčio vaidmenyje gali būti esamos eilutės visų reikšmių vidurkis). Tokiu būdu, gaunama „0-1“ tipo eilutė, kurioje nuliukai asocijuojasi su šviesesnėmis vaizdo sritimis, o vienetukai − su tamsesnėmis (skaičiui 0 atitinka tarpo simbolis („  “) (balta spalva), o 1 − simbolis „█“ (juoda spalva)).

Reikia atspausdinti pradinę eilutę (neapdorotą vaizdą) ir po visų apdorojimo etapų gautą apdorotą vaizdo eilutę.

1. **Skaičiai - V**

Duotas „keturvietis“ masyvas *C*(*M*), 1 ≤ *M* ≤ 20. Kitaip sakant, yra keturios eilutės (žr. 3.14 pav.), kurių elementai formuojami tokiu būdu:

— pirmosios eilutės elementai: elementai lygūs natūrinių skaičių *n*1, *n*2, ..., *nM*, imamų iš duomenų failo U21.txt, kubams; skaičiuojant kubus, reikia remtis savybe: natūrinio skaičiaus *n* kubas lygus sumai *n* nelyginių skaičių, einančių iš eilės po skaičių, kurių suma sudarė skaičiaus *n*−1 kubą: 13 = 1, 23 = 3 + 5, 33 = 7 + 9 + 11, 43 = 13 + 15 + 17 + 19, ...;

— antrosios eilutės elementai: skaitmenys, esantys (*i*\*5)-osiose (1 ≤ *i* ≤ 20) sekos 01101110010111011110001001... pozicijose (pvz., *C*(1) = 1, *C*(2) = 1, *C*(3) = 0, *C*(4) = 0, *C*(5) = 0 ir t. t.);

— trečiosios eilutės elementai: šios eilutės elementai formuojami taip: pertvarkomas duotas teigiamas realus (su trupmenine dalimi) skaičius taip, kad jo sveikosios dalies pradžioje ir pabaigoje būtų 1 (pvz., jeigu duotas (esamas) skaičius lygus 123.67, tai po pertvarkymo turėtų būti gaunamas skaičius 11231.67); iš gautojo skaičiaus ištraukiama kvadratinė šaknis. Teigiami realieji skaičiai (*x*1, *x*2, ..., *xM*) imami iš duomenų failo U22.txt. Skaičių sveikąją dalį sudaro ne daugiau kaip 10 skaitmenų, o trupmeninę dalį − ne daugiau kaip 8 skaitmenys;

— ketvirtosios eilutės elementai: duotos eilutės elementai (*c*1, *c*2, ..., *ci*, ..., *cM*) apskaičiuojami pagal formulę: . Skaičių *c*1, *c*2, ..., *cM* tikslumas − 8 ženklai po kablelio. Skaičiai *α*, *x*1, *x*2, ..., *xi*, ..., *xM* imami iš failo U23.txt. Skaičių trupmeninę dalį sudaro ne daugiau kaip 8 skaitmenys.

Atspausdinti kiekvieną iš keturių suformuotų skaičių eilučių. Kiekvienai iš sudarytų skaičių eilučių apskaičiuoti bendrą skaičių sumą, minimalų skaičių, maksimalų skaičių, aritmetinį vidurkį, medianą.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.14 pav. „Keturvietis“ masyvas

1. **Šachmatų lenta**

Duotas „keturvietis“ masyvas *Š*(*M*), 1 ≤ *M* ≤ 20. Kitaip tariant, to masyvo kiekvienose atitinkamose pozicijose (žr. 3.15 pav.) yra keturi natūriniai skaitmenys *a*, *b*, *c*, *d* ir tokie, kad nė vienas iš šių skaičių nėra didesnis už 8. Visos masyvo reikšmės imamos iš failo U2.txt. Konkrečiai, tų skaičių interpretacija yra tokia. Šachmatų lentos atskiras laukelis (langelis) apibrėžiamas natūrinių skaičių pora ir tokia, jog, kaip minėta, nė vienas skaičius neviršija 8. Pirmasis poros skaičius nurodo vertikalės numerį (skaičiuojama iš kairės į dešinę); antrasis poros skaičius nurodo horizontalės numerį (skaičiuojama iš apačios į viršų). Reikia:

a) tvarkingai atspausdinti visas įvestas iš failo U2.txt reikšmes;

b) išsiaiškinti, ar laukeliai (*a*, *b*) bei (*c*, *d*) yra vienos ir tos pačios spalvos (jei taip, spausdinti „+“; jei ne − „×“);

c) sakykime, kad laukelyje (*a*, *b*) yra valdovė. Ar valdovė, esanti šiame laukelyje, puola laukelį (*c*, *d*)? Jei taip, spausdinti „VP“ („valdovė puola“);

d) o dabar tarkime, kad laukelyje (*a*, *b*) yra žirgas. Ar šis žirgas grąsina laukeliui (*c*, *d*)? Jei taip, spausdinti „ŽP“ („žirgas puola“);

e) nustatyti, ar bokštas iš laukelio (*a*, *b*) gali vienu ėjimu patekti į laukelį (*c*, *d*)? Jei taip, spausdinti „B1“ („bokštas 1 ėjimu“). Priešingu atveju spausdinti „B2“ („bokštas 2 ėjimais“). Pastaruoju atveju reikia nurodyti tarpinį laukelį (*e*, *f*), iš kurio vienu bokšto ėjimu galima patekti į laukelį (*c*, *d*);

f) nustatyti, ar rikis iš laukelio (*a*, *b*) gali vienu ėjimu eiti į laukelį (*c*, *d*)? Jei taip, spausdinti „R1“ („rikis 1 ėjimu“). Priešingu atveju spausdinti „R2“ („rikis 2 ėjimais“) − jei tik tai yra įmanoma. Jei tai įmanoma, reikia nurodyti tarpinį laukelį (*e*, *f*), iš kurio vienu rikio ėjimu galima eiti į laukelį (*c*, *d*).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 eilutė |  | *a* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 eilutė |  | *b* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 eilutė |  | *c* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 eilutė |  | *d* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

3.15 pav. „Keturvietis“ masyvas su reikšmėmis *a*, *b*, *c*, *d* (1 ≤ *a*, *b*, *c*, *d* ≤ 8)

1. **Kuprinės uždavinys**

Yra duota *n* skirtingų daiktų. Žinomi kiekvieno iš šių daiktų svoriai ir jų kainos. Svorių ir kainų matavimo vienetai nėra svarbu; tai gali būti, pavyzdžiui, kilogramai (kg) ir litai (lt).

Užduotis (tikslas) yra sudėti į kuprinę kuo daugiau daiktų tokiu būdu, jog sudėtų į kuprinę daiktų bendra kaina būtų kuo didesnė. Kita vertus, bendras supakuotų daiktų svoris negali viršyti iš anksto užduoto (nustatyto) svorio limito.

Uždavinys sprendžiamas, kai yra žinomi visų daiktų svoriai (kilogramais): *s*1, *s*2, ..., *si*, ..., *sn*. Taip pat yra žinomos daiktų kainos (litais): *c*1, *c*2, ..., *ci*, ..., *cn*. Supakuotos kuprinės svoris negali viršyti 113 kg. Uždavinio sprendimo rezultatas yra gauta daiktų suminė kaina (kartu ir bendras svoris) bei sutalpintų į kuprinę atskirų daiktų numeriai ir svoriai bei kainos.

Daiktų svoriai ir kainos yra imami iš duomenų failo U2.txt. Failo turinio pavyzdys parodytas 3.16 paveikslėlyje (turinys paimtas iš literatūros šaltinio: G.Zäpfel, R.Braune, M.Bögl. *Metaheuristic Search Concepts. A Tutorial with Applications to Production and Logistics*, Berlin-Heidelberg: Springer, 2010).

Pastaba: rekomenduojama uždavinį išspręsti ne mažiau kaip trimis būdais: a) panaudojant pilnojo perrinkimo algoritmą, b) panaudojant „gobšųjį“ algoritmą (į kuprinę pirmiausiai dedami brangiausi daiktai; algoritmo vykdymas baigiamas, kai tik į kuprinę patalpintų daiktų svoris viršija duotą limitą), c) panaudojant savo sugalvotą algoritmą.

|  |  |  |
| --- | --- | --- |
| Daikto  Eil Nr. | Daikto svoris, kg | Daikto  kaina, lt |
| 1 | 32 | 727 |
| 2 | 40 | 763 |
| 3 | 44 | 60 |
| 4 | 20 | 606 |
| 5 | 1 | 45 |
| 6 | 29 | 370 |
| 7 | 3 | 414 |
| 8 | 13 | 880 |
| 9 | 6 | 133 |
| 10 | 39 | 820 |

3.16 pav. Duomenų failo pavyzdys