TP - Examen: Zoo

C++ - ENSISA 2A

Ali El Hadi ISMAIL FAWAZ

12 Janvier, 2024

Zoo

Votre travail dans ce TP consistera à implémenter un programme orienté objet pour représenter une base de données de zoo. Le zoo contiendra trois types d'animaux, 'Mammal', 'Bird' et 'Reptile. Ces trois animaux ont une chose en commun : ils héritent tous de 'Animal'.

1 Animal

Implémentez la classe Animal qui contiendra les éléments suivants :

```
string name; // nom de l'ainal
string color; // couleur de l'animal
string diet; // regime alimentaire de l'animal
string habitat; // habitat de l'animal
string sounds; // son de l'animal
bool isPet; // si l'animal est considere comme animal de compagnie
int age; // son age
double weight; // son poids
double height; // sa taille
```

Cette classe devrait avoir un constructeur par défaut, un constructeur paramétré et un constructeur de copie. Des méthodes setters et getters sont nécessaires pour tous les éléments. Cette classe devrait également avoir une fonction 'make_sound' pour afficher le son de l'animal et une fonction virtuelle 'printInfo' pour afficher toutes les informations de l'animal. Par exemple, la sortie de la fonction dans l'implémentation de la classe Animal devrait ressembler à :

```
Animal of unkown type:
Name : Flash
Color : black
Diet : Carnivore
Habitat : Mountains
```

```
Age : 9's old
This animal is not a pet
Weight : 9.9kg
Height : 1.8cm
```

1.1 Mammal

Implémentez la classe 'Mammal' qui hérite de la classe de base Animal et de tous ses éléments. Cette classe a son propre élément :

```
string furColor; // couleur de la fourrure
```

Cette classe devrait également avoir les trois types de constructeurs et des fonctions getters/setters pour ses éléments. Implémentez et faites un 'override' la fonction virtuelle 'printInfo' de manière à ce que la sortie ressemble à ceci :

```
Animal of type Mammal:
Name : Leo
Color : Golden
Fur Color : Tawny
Diet : Carnivore
Habitat : Grassland
Age : 5's old
This animal is not a pet
Weight : 180.5kg
Height : 3.5cm
```

1.2 Bird

La même chose pour la classe 'Bird' avec son propre élément :

```
double wingspan; // envergure
```

1.3 Reptile

La même chose pour la classe 'Reptile' avec son propre élément :

```
string scalePattern; // Motif d'ecailles du reptile.
```

2 Zoo

Implémentez maintenant la classe 'Zoo' avec ses propres éléments :

```
string name; // Nom du Zoo
static int MAX_CAPACITY; // la capacite maximal du Zoo
vector<Animal*> animals; // Les animaux dans le Zoo
```

Cette classe doit avoir trois constructeurs, le constructeur par défaut, le constructeur paramétré qui prend juste le nom de le Zoo en paramètre, et le constructeur en copie qui copie le nom et les animaux. Les fonctions setters/getters pour le nom de Zoo est necessaire.

Les animaux dans le Zoo doivent être identifiés uniquement par leur nom, c.a.d. que deux animaux dans une même Zoo peuvent pas avoir le même nom.

La classe ‘Zoo‘ doit avoir les fonctions:

- ‘listAnimals‘ qui imprime les information des animaux dans le Zoo
- ‘addAnimal‘ qui ajoute un animal passé comme argument dans le Zoo, si y en a encore de place. Attention: Les animaux dans le Zoo doivent etre tous le temps ordonnés en ordre croissant alphabétique de leur nom. Hint: utilisez la fonction ‘lower_bound‘ en C++ sur les ‘vector‘ leur de l'ajout d'un élément.
- ‘searchAnimalByName‘ qui cherche un animal dans le Zoo en prenant le nom comme argument. La fonction doit rendre l'indice de cet animal dans le ‘vector animals‘ ou bien ‘-1‘ si l'animal n'existe pas. Points Bonus: Vous auriez des points en plus si la complexitee de la fonction est $\mathcal{O}(\log(n))$
- ‘removeAnimalByName‘ pour enlever un animal du Zoo en prenant son nom comme argument, rien faire si l'animal n'existe pas dans le Zoo (imprimer un message qui le dit).
- ‘averageAgeForType‘ qui prend comme argument le type d'animal (”Mammal”, ”Bird” ou bien ”Reptile”) et rend l'âge moyen des animaux de ce type dans le Zoo.

3 K Plus Proches Voisins (K Nearest Neighbour

Dans ce qui suit, le but est maintenant de classer un nouveau animal qui vient d'arriver au Zoo mais qu'on connaît pas a priori son type (Mammal, Bird ou bien Reptile), donc cette élément doit être cree avec la classe ‘Animal‘. Vous allez implémenter l'algorithme de K Plus Proches Voisins pour savoir de quel type est ce nouveau animal.

Sachant que les information d'un animal peuvent etre des ‘string‘ ou bien des ‘double‘, vous allez implementer une classe ‘KNN‘ avec une template et puis 2 classes qui herite la version de cette classe avec un type specifique.

Implémentez la classe ‘KNN‘ qui est en <template T>. Cette classe contient l'élément:

```
int k; le nombre de proche voisin
```

Cette classe doit avoir un constructeur par défaut et un constructeur paramétré.
Seulement un getter est nécessaire pour cette classe pour l'élément ‘k‘. Cette classe doit avoir une fonction ‘findNearestNeighbours‘ qui prend comme paramètre ‘vector<T>& trainData‘ et ‘T& target‘ dans le but de rendre un ‘vector<int> neighbors‘ qui contient les indices des ‘k‘ plus proches voisins. Cette fonction est implémentée dans la classe de Base ‘KNN‘ et doit être indépendante de type ‘<T>. La fonction doit appeler une

autre fonction membre de classe ‘KNN‘ mais c’est une fonction virtuelle pure ‘similarityMeasure(T&, T&)‘ qui rend un ‘double‘ contenant la mesure de similarité entre deux éléments de type ‘<T>‘. Cette fonction sera implémentée dans les classes fils.

3.1 KNNDouble

Pour le cas ou on veut utiliser les information de type ‘double‘, on va chosir juste les 2 information par animal qui sont le ‘weight‘ et le ‘height‘.

Implémentez la classe ‘KNNDouble‘ qui herite la classe ‘KNN‘ en choisissant que ‘T=pair<double, double>‘. Implémentez la distance Euclidienne entre deux paires dans la fonction ‘similarityMeasure‘ en faisant un override.

3.2 KNNString

Dans le cas ou on veut choisir les information de type string, on va prendre en considération juste les éléments ‘color‘, ‘diet‘, ‘habitat‘ et ‘sound‘.

Implémentez la classe ‘KNNString‘ qui hérite ‘KNN‘ en spécifiant que ‘T=vector<string>‘. Implémentez la Levenshtein Edit Distance‘[1] **d’une manière itérative (section 2 sur le lien indiqué)** dans la fonction ‘similarityMeasure‘ en faisant un override.

4 Utilisation de KNN dans Zoo

Implémentez la fonction ‘predictTypeWithKNN‘ dans la classe Zoo qui prend comme paramètre: un élément de type classe ‘Animal‘, le nombre des voisins ‘k‘ et une variable ‘string‘ pour indiquer quel type d’information à utiliser dans le KNN:

- ”numerical” pour utiliser ‘KNNDouble‘
- ”categorical” pour utiliser ‘KNNString‘

Cette fonction doit préparer les variables ‘trainData‘ et ‘target‘ pour les utiliser dans la fonction ‘findNearestNeighbors‘.

5 Main

Créez un fichier ‘main‘ qui utilise tous ces fonctions.

Bonne Chance !

[1] https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/