EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours d'Informatique aux sections MA et PH
Chappelier J.-C.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Programmation I (SMA/SPH) : SÉRIE NOTÉE

(27 novembre 2014)

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre série annulée dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (10h15 - 12h00).
2. Vous avez droit à toute documentation papier.
   Par contre, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
3. Répondez aux questions directement sur la donnée ; utilisez aussi le verso des feuilles, **MAIS** n'utilisez *que* le verso de la feuille sur laquelle se trouve la question, et non **pas** celui de la feuille précédente !
   Vous pouvez, si nécessaire, joindre des feuilles supplémentaires, mais veillez dans ce cas à *n'utiliser que le papier officiel fourni par les assistants* ; **aucune feuille non officielle ne sera corrigée**.
   N'oubliez pas dans ce cas d'*indiquer votre numéro interne* (0000) sur chacune des feuilles supplémentaires, en haut à gauche sous la rubrique « Anonymisation ». Aucune feuille sans ce numéro ne sera corrigée.
4. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
5. L'examen comporte deux exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 47). Tous les exercices comptent pour la note finale.

Question 1 Elément le plus fréquent dans un tableau [sur 11 points]

Le but de cet exercice est d'écrire un programme permettant d'identifier l'élément apparaissant le plus fréquemment dans un tableau d'entiers.

Ce programme devra également afficher le nombre d'occurrences de cet élément. Par exemple, pour le tableau `{2, 7, 5, 6, 7, 1, 6, 2, 1, 7}`, votre programme devra indiquer que l'élément le plus fréquent est le 7 et que son nombre d'occurences est 3 ; par exemple afficher :

```
Le nombre le plus fréquent dans le tableau est le 7 (3 fois).
```

Le code à écrire doit pouvoir s'appliquer à n'importe quel tableau dynamique d'entiers, mais vous pouvez supposer que ces tableaux sont toujours non vides. Notez en particulier que nous n'avons fait aucune hypothèse sur la taille des entiers stockés dans le tableau et que l'on pourrait très bien tester votre programme avec un tableau comme

`{ 2147483647, -2147483648, -2147483648 }`.

Par ailleurs, si dans le tableau donné, il y a plus d'un nombre ayant le plus grand nombre d'occurrences, alors votre programme ne retiendra que celui qui apparaît en premier dans le tableau.

Par exemple, pour le tableau `{2, 7, 5, 6, 7, 1, 6, 2, 1, 7, 6}`, où 6 et 7 sont tous deux les nombres les plus fréquents (les deux apparaissant 3 fois), votre programme ne retiendra que le 7 et affichera :

```
Le nombre le plus fréquent dans le tableau est le 7 (3 fois).
```

Ecrivez ici votre programme :

Question 2 Le gentil peuple de la forêt [sur 36 points]

On cherche ici à écrire un programme permettant de simuler l'évolution des populations d'espèces peuplant une forêt, plantes et animaux. Au niveau de cet exercice le modèle sera bien sûr très simple.

La réalisation de ce programme est décomposée en cinq étapes :

1. structures de données ;
2. affichage d'une espèce ;
3. affichage de l'ensemble des espèces de la forêt ;
4. gestion de la nourriture ;
5. évolutions des populations d'espèces.

Pour vous donner une idée, voici typiquement le genre de `main()` que l'on pourrait écrire :

```
int main()
{
  Foret foret = {
    // Animaux
    { {"loup"    ,  20.0,  10.0 } ,
      {"blaireau",  17.0,  40.0 } ,
      {"lapin"   ,  70.0, 100.0 } ,
      {"limace"  , 200.0,  30.0 }
    },

    // Plantes
    { {"fougère"  ,   100.0,   9.0 },
      {"herbe"    , 10000.0,  60.0 },
      {"mousse"   ,  2000.0, 600.0 },
      {"myrtille" ,   250.0,   6.0 }
    }
  };

  affiche(foret);

  for (int i(1); i <= 12; ++i) {
    evolue(foret);
    cout << "---- " << i << " --------------" << endl;
    affiche(foret);
  }

  return 0;
}
```

[NE RIEN ÉCRIRE SUR CETTE PAGE]

Question 2.1 Structures de données [sur 5 points]

Dans ce programme nous auront besoin de deux sortes de données :

- les « espèces », qui sont caractérisées par : un nom, un nombre d'individus (que l'on représentera par un `double` pour prendre en compte les incertitudes sur le nombre exact) et une grandeur réelle – appelons-la « taux » – qui caractérisera l'évolution de l'espèce (voir plus bas) ;
- la « forêt » qui regroupe deux listes différentes : celle de ses espèces animales (« animaux ») et celle de ses espèces végétales (« plantes »).

Ecrivez ici la définition des *types* de données qui vous semblent appropriés pour représenter les données requises :

Question 2.2 Affichage d'une espèce [sur 6 points]

On veut maintenant afficher une espèce. Pour cela, écrivez ci-dessous une fonction `affiche` qui reçoit en premier argument une espèce et a un second argument, *optionnel*, qui est le verbe utilisé pour l'affichage (voir exemple ci-dessous) et dont la valeur par défaut est `"mange"`.

La fonction devra afficher (voir exemple ci-dessous) :

- le nombre d'individus de l'espèce ;
  si ce nombre est inférieur à 0.5, il faudra afficher « `pas de` » ;
  si ce nombre est compris entre 0.5 et 1.5, il faudra afficher « `un(e)` » ;
- le nom de l'espèce, suivi des mots « `, qui ` » ;
  si le nombre d'individus est supérieur à 1.5, ce nom devra être « mis au pluriel » en ajoutant un `'s'` à l'affichage
  (note : on tolérera donc ici, pour simplifier, la faute d'orthographe « `blaireaus` ») ;
- le verbe reçu comme deuxième paramètre ;
  si le nombre d'individus est supérieur à 1.5, ce verbe devra être mis au pluriel en ajoutant `"nt"`
- le « taux » suivi de « `fois par mois.` ».

Voici quatre exemples d'affichages possibles :

```
70 lapins, qui mangent 100 fois par mois.
69 fougères, qui repoussent 9 fois par mois.
```

```
un(e) loup, qui mange 10 fois par mois.
pas de myrtille.
```

suite au dos ☞

Question 2.3 Affichage de la forêt [sur 3 points]

Ecrivez une fonction `affiche` qui prend simplement une forêt comme argument et qui affiche tous ses animaux puis toutes ses plantes. Par exemple :

```
Dans la foret il y a les animaux suivants :
20 loups, qui mangent 10 fois par mois.
17 blaireaus, qui mangent 40 fois par mois.
70 lapins, qui mangent 100 fois par mois.
200 limaces, qui mangent 30 fois par mois.
et les plantes suivantes :
100 fougères, qui repoussent 9 fois par mois.
10000 herbes, qui repoussent 60 fois par mois.
2000 mousses, qui repoussent 600 fois par mois.
250 myrtilles, qui repoussent 6 fois par mois.
```

Question 2.4 Manger [sur 4 points]

On veut maintenant modéliser, de façon simpliste, la chaîne alimentaire, c'est-à-dire qui mange qui.

Ecrivez pour cela une fonction `mange` qui prend en paramètres deux espèces, disons `a` et `b`, et qui retourne le fait que `a` mange `b`.

Vous ne devez coder que les situations suivantes :

- la limace mange la fougère, l'herbe et la mousse ;
- le blaireau mange les myrtilles et les limaces ;
- le lapin mange l'herbe ;
- le loup mange tout, sauf un autre loup ;
- et par défaut `a` ne mange pas `b`.

Question 2.5 Évolution des populations d'espèces [sur 18 points]

On arrive enfin au but principal du programme : modéliser l'évolution de toutes les espèces de la forêt. En suivant l'approche modulaire, nous allons décomposer cela en deux temps :

— modéliser la recherche de nourriture, que nous appelons ici simplement « chasse » ;
— modéliser toute l'évolution : recherche de nourriture puis évolution des populations en fonction de la nourriture trouvée.

Question 2.5.1 Chasse [sur 8 points]

Ecrivez une fonction sans valeur de retour prenant trois arguments : une espèce (le « chasseur », disons `a`), une quantité de nourriture recherchée (`double`, disons `q`) et une liste d'espèces (« proies » possibles parmi lesquelles le « chasseur » va « chasser »).

La fonction implémente l'algorithme (simple) suivant :
pour toutes les « proies » possibles (disons `b`), si la quantité de nourriture recherchée `q` est encore positive et si le « chasseur » `a` mange la « proie » `b` (fonction précédente), alors :

— si le nombre d'individus de l'espèce « proie » `b` est plus grand que `q`, alors
ce nombre est diminué de `q` et `q` est mis à 0
(tous les animaux de l'espèce « chasseur » ont trouvé à manger, ils ne cherchent plus rien) ;
— sinon, `q` est diminué du nombre d'individus de l'espèce « proie » `b` et ce nombre est mis à 0
(tous les animaux de l'espèce « chasseur » n'ont pas pu trouver à manger, mais ils ont exterminé l'espèce `b`).

A noter que cet algorithme peut donc modifier les deux derniers arguments reçus.

suite au dos ☞

Question 2.5.2 Evolution des populations [sur 10 points]

Pour finir, écrivez une fonction `evolue` qui ne retourne rien et prend simplement une forêt comme paramètre.

L'algorithme d'évolution que l'on vous demande d'implémenter ici est le suivant (modèle volontairement simpliste) :

- on commence par faire manger tous les animaux, qui cherchent d'abord à manger des plantes et sinon d'autres animaux :
  pour tous les animaux de la forêt :
  - initialiser la quantité de nourriture recherchée à leur propre « taux » ;
  - puis « chasser » parmi les plantes (fonction précédente) ;
  - si la quantité de nourriture recherchée est toujours positive, « chasser » parmi les animaux ;
  - si, après tout cela, la quantité de nourriture recherchée est toujours positive, alors l'espèce dépérit : son nombre d'individus est alors multiplié par une valeur égale à 1 moins « la quantité de nourriture recherchée restant divisée par le « taux » de l'espèce » ;
- ensuite on fait pousser les plantes :
  si elles ne sont pas éteintes (c'est-à-dire si leur nombre d'individus est supérieur à 0.5) alors leur nombre d'individus augmente (linéairement, addition) de leur « taux » ;
- et on fait se reproduire les animaux :
  leur nombre d'individus est multiplié par 1.2.