Algorithmique
Correction du DS3

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Exercice 1

Analyseur d'ADN. Rappelons que l'ADN (acide désoxyribonucléique) est une molécule dont la structure (primaire) peut être vue comme une chaîne de caractères. Les caractères sont de quatre sorte : A, T, C et G et correspondent aux quatre nucléotides (adénine, thymine, cytosine, guanine) de la molécule.
Votre collègue biologiste se trouve confronté au problème suivant :

- Données : une chaîne de caractères $\{A, T, G, C\}$ de taille arbitraire.
- Résultat : le nombre de A, de T, de G et de C dans la chaîne.

Construire un algorithme en pseudo-code ou en C++ permettant de résoudre ce problème.

Exercice 1 (suite)

Votre collègue biologiste se trouve confronté au problème suivant :

- Données : une chaîne de caractères $\{A, T, G, C\}$ de taille arbitraire.
- Résultat : le nombre de A, de T, de G et de C dans la chaîne.

Construire un algorithme en pseudo-code ou en C++ permettant de résoudre ce problème.

Tableau ADN

0	1	2	3	4	5	6	7
A	C	G	T	T	A	A	G

3	2	1	2
A	T	C	G

Occurrences

Exercice 1 (suite)

Votre collègue biologiste se trouve confronté au problème suivant :

- Données : une chaîne de caractères $\{A, T, G, C\}$ de taille arbitraire.
- Résultat : le nombre de A, de T, de G et de C dans la chaîne.

Construire un algorithme en pseudo-code ou en C++ permettant de résoudre ce problème.

Tableau ADN

0	1	2	3	4	5	6	7
A	C	G	T	T	A	A	G

3	2	1	2
A	T	C	G

Occurrences

Correction de l'exercice 1 (Il s'agit d'une méthode parmi d'autres)

Algorithme 1: Analyseur d'ADN

```
début
    // On suppose une chaîne d'ADN donnée
    chaîne ADN
    // On utilise un tableau d'occurrences avec 0 pour A, 1 pour T, 2 pour C, 3 pour G
    entier occ[4] ← {0,0,0,0}
    entier i
    pour i de 0 à longueur(ADN) faire
        suivant ADN[i] faire
            cas où 'A'
                occ[0] ← occ[0] + 1
            cas où 'T'
                occ[1] ← occ[1] + 1
            cas où 'C'
                occ[2] ← occ[2] + 1
            cas où 'G'
                occ[3] ← occ[3] + 1
    afficher "Les occurrences sont : " occ[0] "A, " occ[1] "T, " occ[2] "C, " occ[3] "G "
fin
```

Exercice 2

Testeur de mots de passe. En informatique, un *mot de passe* est un moyen d'authentifier un utilisateur pour accéder à une ressource (données, services, etc.) protégée. La robustesse d'un mot de passe dépend, notamment, de sa taille et du nombre de types de caractères utilisés.

Dans cet exercice, nous dirons qu'un mot de passe est *valide* s'il respecte les règles suivantes :

- sa taille fait au moins 8 caractères, et
- le mot contient au moins une lettre majuscule, une lettre minuscule (sans accent) et un chiffre.

En vous basant sur cette règle, proposer un algorithme en C++ permettant de résoudre le problème suivant :

- Données : une chaîne de caractères `mot`.
- Résultat : `1` (vrai) si la chaîne `mot` est un mot de passe valide, et `0` (faux) sinon.

Note

En C++, il est possible de savoir si un caractère est un chiffre, une lettre minuscule ou une lettre majuscule, en utilisant simplement la table ascii.
Rappelons qu'en C++, tout caractère peut être vu comme un entier, cet entier correspondant à l'index du caractère dans la table ascii. Par exemple, si `c` est une variable de type `char`, alors l'expression `c >= 48` indique que le code ascii de `c` doit être plus grand ou égal à celui de l'index 48 de la table ascii (correspondant au chiffre 0). Ainsi l'expression `(c >= 48) && (c <= 57)` est vraie si et seulement si `c` est un chiffre.
Le tableau suivant est une partie de la table ascii correspondant aux chiffres, lettres majuscules, et lettres minuscules.

Index	Caractère
48	0
49	1
...	...
56	8
57	9
65	A
66	B
...	...
89	Y
90	Z
97	a
98	b
...	...
121	y
122	z

Correction de l'exercice 2 (il existe d'autres solutions possibles)

```
#include <iostream>
using namespace std;

int main()
{
string mot;
cout ≪ "Entrer votre mot de passe";
cin ≫ mot;
if(mot.size()<8)
  cout ≪ "faux" ≪ endl;
else
  {
  int chiffres = 0, minuscules = 0, majuscules = 0;
  for(int i = 0; i < mot.size(); i++)
    {
    if(mot[i] >= 48 && mot[i] <= 57) chiffres++;
    if(mot[i] >= 65 && mot[i] <= 90) majuscules++;
    if(mot[i] >= 97 && mot[i] <= 122) minuscules++;
    }
  if(chiffres == 0 || minuscules == 0 || majuscules == 0)
    cout ≪ "faux" ≪ endl;
  else
    cout ≪ "vrai" ≪ endl;
return 0;
}
```

Exercice 3

Testeur d'adresses IP. Dans les réseaux informatiques, l'adresse IP (Internet Protocol) d'une machine est un numéro d'identification (dans le réseau) qui permet à la machine de communiquer avec les autres machines du réseau. L'adresse IP d'une machine est une séquence de 4 nombres, chaque nombre étant un entier entre 0 et 255.
Pour des raisons de lisibilité, les entiers de l'adresse IP sont séparés par des points. Par exemple `86.215.169.62` est une adresse IP.

Dans cet exercice, vous allez faire votre premier travail d'administrateur réseau. Vous êtes responsable du réseau d'une petite entreprise, chaque machine de l'entreprise disposant d'une adresse IP. Une nouvelle machine arrive et vous devez tester si son adresse IP existe déjà dans votre réseau.
Plus formellement, vous devez construire un algorithme (en pseudo-code ou en C++) permettant de résoudre le problème suivant :

- Données : un ensemble E contenant n adresses IP, et une adresse IP A.
- Résultat : "vrai" si $A \in E$ et "faux" sinon.

Note

La partie importante de cet exercice est de *modéliser* la notion d'adresse IP. Plusieurs choix sont possibles (structure, tableau, codage en entier, etc.). L'ensemble E d'adresses IP pourra être modélisé comme un tableau de taille n, dont les éléments sont des adresses IP.

Note

La partie importante de cet exercice est de *modéliser* la notion d'adresse IP. Plusieurs choix sont possibles (structure, tableau, codage en entier, etc.). L'ensemble E d'adresses IP pourra être modélisé comme un tableau de taille n, dont les éléments sont des adresses IP.

```
typedef unsigned short AdresseIP[4];
```

Un modèle utilisant les tableaux

Note

La partie importante de cet exercice est de *modéliser* la notion d'adresse IP. Plusieurs choix sont possibles (structure, tableau, codage en entier, etc.). L'ensemble E d'adresses IP pourra être modélisé comme un tableau de taille n, dont les éléments sont des adresses IP.

```
struct AdresseIP
  {
  unsigned short a;
  unsigned short b;
  unsigned short c;
  unsigned short d;
  };
```

Un modèle utilisant les structures

Note

La partie importante de cet exercice est de *modéliser* la notion d'adresse IP. Plusieurs choix sont possibles (structure, tableau, codage en entier, etc.). L'ensemble E d'adresses IP pourra être modélisé comme un tableau de taille n, dont les éléments sont des adresses IP.

```
typedef string AdresseIP;
```

Un "pas bon" modèle utilisant les chaînes : la chaîne `255.000.000.000` n'est pas la même que la chaîne `255.0.0.0`, or elles désignent la même adresse IP

Correction de l'exercice 3 (il existe d'autres solutions possibles)

```
#include <iostream>
using namespace std;

typedef unsigned short AdresseIP[4];

int main()
{
// On suppose que l'ensemble d'adresses E et l'adresse A sont données
AdresseIP E[n];
AdresseIP A;

bool trouve = 0;
int i = 0;
while(!trouve && i < n)
  {
  trouve = (E[i][0] == A[0]) &&
           (E[i][1] == A[1]) &&
           (E[i][2] == A[2]) &&
           (E[i][3] == A[3]);
  if(!trouve) i++;
  }
cout << trouve << endl;
return 0;
}
```