Algorithmique - TP3

IUT 1ère Année

18 octobre 2015

1 Exercices sur les structures

Dans les exercices suivants, nous utiliserons la structure suivante pour représenter les cartes.

```
enum Couleur {trefle, carreau, pique, coeur};
enum Face {sept, huit, neuf, dix, valet, dame, roi, as};
struct Carte
    {
    Couleur couleur;
    Face face;
    };

typedef Carte JeuDeCartes[32];
```

Exercice 1. Construire un programme permettant d'implémenter les deux fonctions suivantes, vues en TD4 (ainsi que dans les derniers exercices du TP2).

```
void Construire(JeuDeCartes& jeu);

void Melanger(JeuDeCartes& jeu);
```

Exercice 2. Construire un programme permettant de distribuer une *main* (5 cartes) au joueur à partir d'un jeu de 32 cartes mélangées.

Exercice 3. Construire un programme permettant de tester si le joueur possède une *paire* au Poker.

- Données : une main distribuée à partir d'un jeu de 32 cartes mélangées
- Résultat : *vrai* si le joueur possède une paire, et faux *sinon*.

Rappelons qu'une paire est constituée par deux cartes de même face.

Exercice 4. Construire un programme permettant de tester si le joueur possède une *paire*, *deux paires*, un *brelan*, un *full* ou un *carré* au Poker.

- Données : 5 cartes distribuées à partir d'un jeu de 32 cartes mélangées
- Résultat : par ordre décroissant :
  - *un carré* si le joueur possède quatre cartes de même face,
  - *un full* si le joueur possède un brelan et une paire,
  - *un brelan* si le joueur possède trois cartes de même face,
  - *deux paires* si le joueur possède deux paires, les faces de la première étant différente des faces de la seconde,
  - *une paire* si le joueur possède deux cartes de même face,
  - *rien* si aucune des combinaisons précédentes n'apparaît.

Il est conseillé d'utiliser des fonctions, chacune associée au test d'une combinaison. Par exemple la fonction `estUnCarre` prend en entrée une main et retourne en sortie *vrai* si la main contient un carré, et *faux* sinon.

Optionnel : étendre le problème au test de la *quinte flush*, la *couleur* et la *quinte*.

2 Exercices sur les algorithmes de chaînes

Exercice 5. Construire un programme permettant de résoudre le problème suivant :
- Données : une chaîne représentant un mot en minuscules (nom simple, verbe, adjectif) de la langue française.
- Résultat : le nombre de voyelles et le nombre de consonnes dans la chaîne.

Par exemple, le mot *élève* contient trois voyelles et deux consonnes.

Exercice 6. Construire un programme permettant de résoudre le problème suivant :
- Données : deux chaînes x et y représentant chacune un mot en majuscules (on n'utilise pas les accents) de la langue française.
- Résultat : *vrai* si x et y sont des anagrammes et *faux* sinon.

Rappelons que x est un *anagramme* de y si x est une permutation des lettres de y Par exemple, le mot CHIEN est un anagramme du mot NICHE. De la même manière, le mot LIERRE est un anagramme du mot RELIER. Mais LIRE n'est pas un anagramme de LIERRE puisqu'il n'utilise qu'un seul R et qu'un seul E.

Exercice 7. Construire un programme permettant de résoudre le problème suivant :
- Données : une phrase de la langue française ne contenant pas de symbole de ponctuation autre que le point final.
- Résultat : le nombre de mots dans la phrase.

Optionnel : étendre le problème à des phrases pouvant contenir les symboles de ponctuation habituels :

`! ? , ; : ( ) ...`

Rappelons que l'apostrophe `'` n'est pas un symbole de ponctuation, mais un signe grammatical (et donc un mot terminé par une apostrophe est compté comme tel).

3 Exercices sur les fichiers

Exercice 8. En vous basant sur l'algorithme de lecture de fichiers vu en TD5, construire un tableau de chaînes à partir du fichier `mots.txt` accessible dans le répertoire TP du module Algorithmique sous Moodle (ou Google Drive).
- Données : le fichier `mots.txt` contenant 22740 mots de la langue française, chacun apparaissant dans une ligne du fichier.
- Résultat : un tableau de chaînes appelé `lexique`, chaque entrée correspondant à un mot du fichier.

En particulier, l'index 0 du tableau `lexique` est associé au mot `Aaron`, l'index 1 est associé au mot `abaissé`, etc.

Exercice 9. En vous basant sur l'exercice précédent, construire votre premier analyseur orthographique :
- Données : une phrase de la langue française ne contenant pas de symbole de ponctuation autre que le point final.
- Résultat : les mots de la phrase non reconnus (mal orthographiés) par le tableau `lexique`.

Note : pour résoudre ce problème, vous pourrez découper la phrase en mots, et tester pour chaque mot s'il appartient ou non au lexique.

Exercice 10. Dans cet exercice final vous construirez votre premier analyseur lexical pour des fichiers "texte".
- Données : un fichier contenant un texte en langue française.
- Résultat : le nombre d'occurrences de chaque mot apparaissant dans le texte.

Par exemple, dans le texte *Édouard, après deux victoires remportées en deux jours, prit Calais, qui resta aux Anglais deux cent dix ans*, le nom *deux* apparaît trois fois. Pour tester votre programme, vous utiliserez les trois poèmes de Prévert :
- `chez_la_fleuriste.txt`
- `dejeuner_du_matin.txt`
- `inventaire.txt`

accessibles dans le répertoire TP sous Moodle (ou Google Drive). Le dernier poème contient plusieurs symboles de ponctuation (qu'il ne faudra pas compter). Bien entendu, vous pourrez utiliser aussi d'autres textes de votre choix.