C++ - TP 7
Problème: Bataille (4h)

L'objectif de ce TP est d'écrire un programme de jeu de cartes bien connu, la bataille. Dans cette version, on jouera avec un jeu de 32 cartes. Après mélange et distribution des cartes à deux joueurs, chacun retourne la première carte de son tas. La carte la plus forte gagne la manche et les deux cartes sont mises en dernier dans le tas du gagnant. L'ordre est : As, Roi, Dame, Valet, 10, 9, 8, 7. Dans un premier temps, en cas d'égalité, la couleur la plus forte est gagnante dans l'ordre : Pique, Coeur, Carreau, Trèfle (on évite donc les batailles). Le joueur qui ramasse toutes les cartes est déclaré gagnant.

Nous allons coder les cartes avec des entiers allant de 0 à 31, de la façon suivante, où P, C, K, T représentent respectivement pique, coeur, carreau et trèfle :

0	1	2	3	4	5	6	7	8	9	..	14	15	16	...	23	24	...	31
7P	8P	9P	10P	VP	DP	RP	1P	7C	8C	...	RC	1C	7K	...	1K	7T	...	1T

Question : Selon ce codage, quelle propriété est satisfaite par les indices des 7 ? Par ceux des rois ?

On vous demande de réaliser un programme simulant une partie automatique, en respectant le découpage en fonctions que nous allons vous décrire par la suite. Pour vous simplifier la tâche, vous trouverez le squelette du programme sous Claroline dans le fichier `bataille_etudiant.cpp`.
En plus des entêtes de fonctions, vous verrez que le `main` contient les déclarations des structures de données qui nous intéressent : `jeu` contiendra l'ensemble des cartes, `jeu1` et `jeu2` les cartes des deux joueurs, les variables $n1$ et $n2$ le indices des dernières cartes des deux joueurs dans leurs tableaux `jeu1` et `jeu2` (et pas le nombre de cartes). Autrement dit, si $n1 = 0$, le joueur 1 a une carte (accessible en `jeu1[0]`), si $n1 = 1$, il en a deux (`jeu1[0]` et `jeu1[1]`), etc. Au départ les joueurs auront chacun seize cartes, d'où $n1 = n2 = 15$.

1 Initialisation du jeu et affichages

Codez la fonction `initjeu` qui remplit le tableau passé en paramètre avec les valeurs des cartes rangées dans l'ordre, autrement dit, $0, 1, \ldots, 31$. Codez ensuite les deux fonctions d'affichage :

- `affiche_carte` affiche la carte passée en paramètre (comprise entre 0 et 31) sous sa forme compréhensible. Par exemple, si le paramètre vaut 3, votre fonction affichera $10P$.
- `affiche` qui affiche les n premières cartes du tableau passé en paramètre, séparées par des espaces.

Dans votre main, testez le bon fonctionnement de vos trois fonctions, en initialisant le tableau *jeu*, puis en l'affichant (vous devez voir l'ensemble des 32 cartes affichées dans l'ordre).

2 Mélange des cartes

Avant de distribuer le jeu, on souhaite tout d'abord le mélanger. L'algorithme de mélange qu'on vous propose consiste à prendre deux cartes au hasard dans le tableau *jeu*, et à les échanger. En répétant cette opération plusieurs fois, on arrive à un résultat satisfaisant.
Commencez par écrire la fonction `permute`, qui échange les valeurs des deux cartes passées en paramètre.
Codez ensuite la fonction `melange`, qui va tirer deux indices de cartes au hasard (entre 0 et 31) dans le tableau

jeu, les échanger, et répéter cette opération disons une centaine de fois. Testez votre fonction dans le `main`, en affichant le jeu avant et après mélange.

3 Distribution des cartes

Codez la fonction `distribution` qui à partir du tableau `jeu`, distribue seize cartes à chaque joueur dans les tableaux `jeu1` et `jeu2`. Testez votre fonction dans le `main` en affichant ces deux tableaux après distribution.

4 Jeu de la carte

On vous demande maintenant de coder la fonction `jouer1coup`, qui a pour paramètres les tableaux de jeu des deux joueurs ainsi que les indices des dernières cartes. Voilà en quoi doit consister votre algorithme : les cartes jouées sont celles d'indice 0. Gardez les en mémoire, puis décalez toutes les cartes restantes de chaque joueur vers la gauche (cartes 1 à $n1$ (resp. $n2$) mises de 0 à $n1-1$ (resp. $n2-1$)).
Les deux cartes gardées en mémoire sont alors comparées (valeur et éventuellement couleur en cas d'égalité) et rangées dans le tas du gagnant aux positions $n1$ et $n1+1$ si le joueur 1 gagne ou $n2$ et $n2+1$ si c'est le joueur 2. Les valeurs de $n1$ et $n2$ doivent bien sur être ajustées en conséquence.

Attention : Les cartes doivent être mises dans le tas du gagnant en commençant par celle(s) du gagnant.

Tester le bon fonctionnement sur quelques coups du jeu en affichant par exemple les cartes restantes des deux joueurs après chaque coup.

5 Programme principal

Complétez le programme principal à l'aide des commentaires qui vous sont donnés dans le fichier fourni. Veillez notamment à faire attention à la condition de fin de partie. Vous afficherez également le nom du joueur vainqueur, et le nombre de coups que la partie aura duré.

6 Nombre moyens de coups

Lancez un grand nombre de parties successives (>1000), et calculez le nombre moyens de coups que dure une partie.

7 Ajout de la bataille

Intégrez le principe de la bataille aux règles du jeu lorsqu'il y a égalité, càd que tant qu'il y a égalité, les deux joueurs jouent leur carte suivante. Dès qu'un joueur l'emporte, il gagne toutes les cartes mises en jeu lors des égalités précédentes. Si un joueur ne peut plus fournir de cartes alors qu'il y a bataille, il perd la partie.

Pour les plus rapides... (*"Pour en terminer au plus vite, il est indispensable d'en finir au plus tôt !"*, Pierre Dac)

8 Nombre McNugget

Un nombre n est dit *McNugget* si l'on peut, en choisissant bien ses boîtes de Chicken McNuggets, parvenir à un total de n nuggets. On utilisera les boîtes classiques qui contiennent 6, 9 ou 20 nuggets.

A faire : Ecrire un programme qui demande un nombre n à l'utilisateur est affiche si n est un nombre McNugget (et dans l'affirmative, affiche le nombre de boîtes).

Dans un second temps, on veut aller un peu plus loin et écrire un programme qui réponde à la question suivante :

Question : Quel est le plus grand entier qui n'est pas un nombre McNugget (s'il existe) ? (autrement dit, existe-t-il un nombre à partir duquel tous les nombres sont des nombres McNugget ?)

9 Plus grand produit palindromique

Un nombre palindromique est un nombre qui se lit dans les 2 sens (exemples 1221, 26762). Le plus grand palindrome généré par le produit de 2 nombres à 2 digits est $9009 = 99 \times 91$.

Question : Quel le plus grand palindrome généré par le produit d'un nombre à 2 digits par un nombre à 3 digits ?