Des classes

On va utiliser des classes pour représenter les différents éléments du jeu.

Pour chaque classe, vous pouvez définir des données membres, des fonctions membres ou des fonctions externes supplémentaires.

Carte

Chaque carte sera représentée par un objet de la classe `Carte`.

1. Créer une classe `Carte`. Une carte a :
   - un nom,
   - une valeur d'attaque,
   - une valeur de défense
   - une valeur de puissance magique.
2. On va avoir besoin de plusieurs constructeurs.
   a. Définir un constructeur vide initialisant une carte à des valeurs par défaut : Mains nues, 1 d'attaque, 1 de défense et 0 de puissance magique
   b. Définir un constructeur qui prend quatre arguments : une chaîne de caractères et trois entiers pour initialiser les différents champs d'une carte.
3. Écrire deux fonctions membres permettant l'affichage d'une carte.
   Proposer un affichage long et un court sur une ligne, comme sur les exemples suivants :

```
***************       ou          Cuillere [ 2 ; 0 ; 0 ]
 Nom : Cuillere
 Attaque : 2
 Defense : 0
 Magie : 0
***************
```

4. Définir une fonction membre `int degatPhysique(const Carte &)const;` dont les paramètres sont deux cartes (dont l'objet cible) et qui retourne les dégâts physiques subis, en positif si c'est le joueur qui a joué la carte cible qui les subit et en négatif si c'est l'autre joueur qui les subit.
   Par exemple, si la `Carte c1` est la cuillere (Att = 2 , Def = 0) et la `carte c2` est l'Epee simple (Att = 5 , Def = 1), alors `c1.degatPhysique(c2)` va retourner 5 (car le joueur qui joue la cuillere subit 5 dégâts).
5. Définir une fonction membre `int degatMagique(const Carte &)const;` dont les paramètres sont deux cartes et qui retourne les dégâts magiques subis, en positif si c'est le joueur ayant joué la carte cible qui subit les dégâts et en négatif si c'est l'autre joueur qui subit les dégâts.
   Par exemple, si la `Carte c1` est la Anneau Unique (Magie = 12) et la `carte c2` est l'Epee simple (Att = 5 , Def = 1, Magie = 0), alors `c1.degatMagique(c2)` va retourner -12 (car le joueur qui a joué la carte c1 inflige 12 dégât à l'autre.).
6. **Si vous avez le temps**, définir des opérateurs réalisant le traitement effectué par les fonctions des questions 4 et 5.
7. **Si vous avez le temps,** définir l'opérateur << pour qu'il produise l'affichage sur une ligne d'une carte.

Joueur

Chaque joueur sera représenté par un objet de la classe `Joueur`.

1. Créer une classe `Joueur`. Un joueur a :
   - un nom
   - un nombre de point de prestige (initialement 25)
   - une pile de carte représenté par un vecteur de carte (initialement vide).
2. Cette classe devra disposer de plusieurs constructeurs
   a. Définir un constructeur vide initialisant un joueur dont le nom est Yugi qui a 25 de prestige et une pile de cartes composées des cartes suivantes :
   MainDroite [1,0,0], MainGauche [0,1,0], Tête [1,-2,0], Bouche[1,-1,2], Genou[2,0,0]
   b. Un constructeur avec paramètres permettant d'initialiser les données de l'objet.
3. Écrire des fonctions membres `sonNom` pour obtenir le nom du joueur et `sonPrestige` son nombre de point de prestige
4. Écrire une fonction membre permettant l'affichage d'un joueur selon le modèle ci-dessous
   `Le joueur Kaiba a 17 points de prestige, il reste 13 cartes dans sa pile.`
5. Créer une fonction membre permettant à joueur d'initialiser sa pile à partir d'un vecteur de cartes représentant une réserve.
   Dans un premier temps, ce sous-programme prendra les vingt premières cartes de la réserve. Plus tard, vous proposerez des sous-programmes permettant de construire des piles d'une manière plus réfléchie.
6. Écrire une fonction membre booléenne pour savoir si un joueur a encore des cartes et aussi (une fonction membre booléenne pour savoir si un joueur est toujours en jeu (c'est-à-dire si son prestige est strictement positif).
7. Écrire une fonction membre `void carteSuiv(Carte &);` qui permet de connaître la prochaine carte de la pile du joueur et qui la retire de la pile.
   S'il n'y en a pas, une exception est levée.
8. Écrire une fonction membre `void jouentUneCarte(Joueur &);` qui met à jour pour chaque joueur leur nombre de points de prestige en fonction du résultat du combat de la prochaine carte pour chaque joueur.
   Si un des joueurs n'a plus de carte, une exception doit être levée.
9. Amélioration possible **si vous avez le temps** : la pile n'est pas un vecteur de cartes mais un vecteur d'indices pointant dans la réserve de carte en utilisant les principes de l'indirection. Quel est l'avantage de procéder ainsi ? Quels changements sont nécessaires ?

Jeu

On va définir une classe Jeu dont les données membres seront :

- `reserve` un vecteur d'objets `Carte`, représentant la liste des cartes disponibles pour la création des piles de chaque joueur.
- `joueur1` et `joueur2` de la classe Joueur représentant les deux joueurs participant au jeu.

1. Définir la classe `Jeu`.
2. Cette classe devra disposer de deux constructeurs :
   a. Définir un constructeur vide initialisant un jeu à des valeurs par défaut
   b. Un constructeur avec paramètres permettant d'initialiser les données de l'objet.

3. Écrire une fonction membre booléenne `bool fini(Joueur & vainqueur);` pour savoir si le jeu est fini et qui renvoie dans un paramètre le joueur vainqueur
4. **Si vous avez le temps,** écrire un opérateur ++ qui fait avancer le jeu d'un tour (une carte est jouée par chaque joueur).