



Remise du travail

- Date : **1 novembre 2022 avant 23h30**
- Une note de 0 sera attribuée aux équipes qui remettent leur travail en retard.
- Remettre **tous** les fichiers directement sous une archive **.zip** (sans dossier à l'intérieur et sans autres fichiers).
- ✂ Toute archive autre que zip ou remise ne respectant pas ces consignes sera refusée et **se méritera une note de 0**. Attention, une archive **.7z ne compte pas comme une archive zip**. ✂
- Respectez le format de la remise! Nom de l'archive zip : **matricule1_matricule2_groupe.zip**
Exemple : 1234567_1234568_1.zip
- Vous devez uniquement remettre les fichiers .h, .cpp et le fichier .h pour la classe générique.

Références



Directives

- Notes de cours sur Moodle et le livre Big C++
- Les travaux s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.
- Les fonctions que vous décidez d'ajouter au programme doivent être documentées.

Conseils

- Si vous avez des difficultés au cours du TP, rappelez-vous que de nombreux problèmes demandés sont assez couramment rencontrés en C++. Consulter la documentation sur cppreference, les notes de cours, et les forums sur Google peuvent vous donner de bonnes pistes de résolution!
- Veuillez utiliser la version v17 de C++.

Spécifications générales

 Le non-respect des spécifications générales entraînera des pénalités. 

- Vous pouvez utiliser n'importe quel environnement autant que le compilateur soit C++17.
 - o Des ressources sont disponibles pour vous aider à débiter avec votre environnement:
 - Visual studio: <https://youtu.be/BqLw7RDnNew>
 - Visual studio code: <https://youtu.be/d5jy7YHLX0w>
 - Xcode: <https://youtu.be/oTeotPuDPbY>
- **L'inclusion des fichiers d'en-tête (.h) doit être sensible aux majuscules et minuscules.** Malheureusement, sur Windows les fichiers ne sont pas sensibles aux majuscules et minuscule, donc des erreurs d'inclusions pourraient passer inaperçues. Relisez-vous bien.
- Toutes les méthodes doivent être définies dans le fichier d'implémentation (.cpp) **dans le même ordre** que leur déclaration dans le fichier d'en-tête (.h). Le non-respect de cette règle entraînera une pénalité au niveau du style.
- Utilisez le plus possible la liste d'initialisation des constructeurs. L'utilisation du corps des constructeurs à la place de la liste d'initialisation entraînera une pénalité de style. L'ordre des variables (attributs) dans la liste d'initialisation doit être la même que celle dans la liste des attributs de la classe dans le fichier d'en-tête.
- Suivez le guide de codage sur Moodle.
- Modifications/ajouts au guide de codage à respecter :
 - o Vous pouvez utiliser le style d'accolades que vous désirez, **tant que vous êtes uniformes**. Sachez cependant qu'il est généralement attendu d'un(e) développeur(se) suivre la convention établie par le projet, ou la convention établie par la langue (si elle existe) en créant un nouveau projet. Dans le cas du TP fourni, les accolades ouvrantes sont sur leur propre ligne (style Allman).
 - o Mettez un espace avant la parenthèse ouvrante des énoncés de contrôle comme if, for, while et switch, mais pas avant les parenthèses d'un appel de fonction. Ne mettez jamais d'espace tout juste après une parenthèse ouvrante ou tout juste avant une parenthèse fermante.
 - o Le deux-points (:) et le signe égal (=) devraient être entourés d'espaces, sauf dans le cas des étiquettes de case et les spécificateurs d'accès (private, protected et public) qui ne devraient pas avoir d'espace avant le deux-points. Un espace devrait toujours suivre les éléments de ponctuation comme la virgule (,), le deux points (:) et le point-virgule (;).
 - o N'utilisez pas NULL ou 0 pour les pointeurs, mais bien nullptr. Dans le cas de test de nullptr avec un pointeur, soyez explicite : préférez if (p != nullptr) à if (p).

- N'utilisez pas de `else` après un `return`.

Mise en contexte

Maintenant que vous avez fait l'implémentation des différentes classes lors du dernier TP, vous devez désormais vous familiariser avec l'héritage, le polymorphisme, la conversion statique et dynamique et la classe générique. Dans ce TP, vous aurez donc à implémenter toutes les classes de manière à faire passer tous les tests.

Aperçu des classes du projet

- ***Date***: Classe qui représente une date.
- ***Joueur*** : Classe qui représente un joueur d'une équipe.
- ***Gardien*** : Classe qui représente un gardien d'une équipe et qui hérite de la classe Joueur.
- ***Attaquant*** : Classe qui représente un attaquant d'une équipe et qui hérite de la classe Joueur.
- ***Defenseur*** : Classe qui représente un défenseur d'une équipe et qui hérite de la classe Joueur.
- ***Équipe*** : Classe qui représente l'ensemble de tous les joueurs de l'équipe.
- ***Match*** : Classe générique qui représente un match entre deux équipes.
- ***EquipeSport*** : Classe qui représente un sport en général.

Travail à réaliser

Implémenter les classes en complétant les méthodes manquantes et les fonctions globales des fichiers .cpp fournis. Il faut aussi trouver les méthodes qui doivent être virtuelles dans les fichiers .h

Classe Date

La classe représente une date ayant les attributs suivants : Année, Mois et jour. Elle est complètement implémentée.

Classe Joueur

Classe qui représente un joueur d'une équipe. Les attributs `attaque_` et `defense_` sont ajoutés.

Il faut implémenter ou changer :

- Le constructeur par paramètres et par défaut (`attaque_` et `defense`) sont à zéro).
- Les méthodes `setAttaque()`, `setDefense()`, `getAttaque()`, `getDefense()`.
- La fonction globale `operator <<` pour ajouter l'affichage des nouveaux attributs.

Classe Gardien

Classe qui hérite de la classe Joueur. Les attributs `stabilite_` et `reflexe_` sont ajoutés à la classe de base.

Il faut implémenter ou changer :

- Le constructeur par défaut initialise les attributs de la classe de base Joueur et les attributs : $\text{attaque_} = 0.1$, $\text{defense_} = 1.0$, $\text{stabilite_} = 0.9$, $\text{reflexe_} = 1.2$.
- Le constructeur par paramètres.
- La méthode `getAttaque` est changée pour $\text{attaque_} \text{ (de la classe de base) } * (\text{stabilite_} + \text{reflexe_}) / 2.5$.
- La méthode `getDefense` est changée pour $\text{defense_} \text{ (de la classe de base) } * (\text{stabilite_} + \text{reflexe_}) / 2$.
- Les méthodes `getStabilite()` et `getReflexe()`.
- La fonction globale `operator <<` pour afficher « Gardien », faire appel à la fonction globale `operator <<` de Joueur et afficher les nouveaux attributs.

Classe Attaquant

Classe qui hérite de la classe Joueur. Les attributs `vitesse_`, `agilite_` et `precision_` sont ajoutés à la classe de base.

Il faut implémenter ou changer :

- Le constructeur par défaut initialise les attributs de la classe de base Joueur et les attributs: $\text{attaque_} = 1.0$; $\text{defense_} = 0.3$; $\text{vitesse_} = 1.0$; $\text{agilite_} = 0.8$; $\text{precision_} = 1.0$;
- Le constructeur par paramètres.
- La méthode `getAttaque` est changée pour $\text{attaque_} * (\text{vitesse_} + \text{agilite_} + \text{precision_}) / 3$;
- La méthode `getDefense` est changée pour $\text{defense_} * (\text{vitesse_} + \text{agilite_} + \text{precision_}) / 3.5$;
- Les méthodes `getVitesse()` et `getAgilite()`, `getPrecision()`.
- La fonction globale `operator <<` pour afficher « Attaquant », faire appel à la fonction globale `operator <<` et afficher les nouveaux attributs.

Classe Defenseur

Classe qui hérite de la classe Joueur. Les attributs `reactivite_`, et `vision_` sont ajoutés à la classe de base.

Il faut implémenter ou changer :

- Le constructeur par défaut initialise les attributs de la classe de base Joueur et les attributs: $\text{attaque_} = 0.1$; $\text{defense_} = 0.9$; $\text{reactivite_} = 0.8$; $\text{vision_} = 1.0$;
- Le constructeur par paramètres.
- La méthode `getAttaque` est changée pour $\text{attaque_} * (\text{reactivite_} + \text{vision_}) / 2.5$
- La méthode `getDefense` est changée pour $\text{defense_} * (\text{reactivite_} + \text{vision_}) / 2$
- Les méthodes `getReactivite()` et `getVision()`.
- La fonction globale `operator <<` pour afficher « Defenseur », faire appel à la fonction globale `operator <<` de Joueur et afficher les nouveaux attributs.

Classe Équipe

Classe qui représente l'ensemble des joueurs de l'équipe.

Il faut implémenter

- calculAttaque() : Cette méthode parcourt tous les joueurs de l'équipe (gardien, défenseur et attaquant), somme toute la force d'attaque des joueurs. Chaque valeur des attaques est multipliée par une valeur aléatoire entre [0.8,1.8] avant de l'ajouter à la somme. Il est bien entendu que la force des attaques est différente selon le type des joueurs.
- calculDefense() : Cette méthode parcourt tous les joueurs de l'équipe (gardien, défenseur et attaquant), somme toute la force des défenses des joueurs. Chaque valeur des défenses est multipliée par une valeur aléatoire entre [0.4,1.5] avant de l'ajouter à la somme. Il est bien entendu que la force des défenses est différente selon le type des joueurs.
- getJoueurs() : retourne le tableau des joueurs.
- La fonction globale amie operator << qui affiche le nom de l'équipe, les joueurs, le nombre de victoires, la somme de la force d'attaque de l'équipe, et la somme de la force de défense de l'équipe.

Classe Match

Classe générique qui représente un match entre 2 équipes.

La définition de la classe vous est donnée, ainsi que la méthode simuler qui permet d'affronter les équipes selon leur force d'attaque et de défense.

- Il faut implémenter toutes les autres méthodes.
- La méthode getScores retourne un pair de valeurs entières des scores de l'équipe A et de l'équipe B.
- La méthode estVictorieuse(const **shared_ptr**<E> &equipe) permet de vérifier que le paramètre equipe est l'équipe victorieuse selon le score obtenu des 2 équipes. Si le paramètre equipe ne correspond ni à l'équipeA, ni l'équipeB on retourne faux.
- La fonction globale operator << affiche la date, le nom des 2 équipes, et le score des 2 équipes.

Cette classe générique peut être tester avec la classe Equipe et la classe EquipeSport.

main.cpp

Le programme principal roule une série de tests. L'affichage devrait correspondre à ce qui suit :

```
Tests pour classe Joueur:
-----
Test 1: Constructeur par défaut de la classe Joueur : OK
Test 2: Constructeur par parametres de la classe Joueur : OK
Test 3: Setteurs et getters de la classe Joueur : OK
Test 4: operateur << de la classe Joueur : OK
-----
Total des tests qui passent pour la section: 19/19 tests

Tests pour classe Gardien:
-----
Test 5: Constructeur par défaut de la classe Gardien : OK
Test 6: Constructeur par parametre de la classe Gardien : OK
Test 7: Getters de la classe Gardien : OK
Test 8: operateur << de la classe Joueur : OK
-----
Total des tests qui passent pour la section: 25/25 tests

Tests pour classe Attaquant:
-----
Test 9: constructeur par défaut de la classe Attaquant : OK
Test 10: constructeur par parametre de la classe Attaquant : OK
Test 11: getters de la classe Attaquant : OK
Test 12: operateur << de la classe Attaquant : OK
-----
Total des tests qui passent pour la section: 14/14 tests

Tests pour classe Defenseur:
-----
Test 13: Constructeur par défaut de la classe Defenseur : OK
Test 14: Constructeur par parametre de la classe Defenseur : OK
Test 15: Constructeur par getters de la classe Defenseur : OK
Test 16: operateur << de la classe Defenseur : OK
-----
Total des tests qui passent pour la section: 25/25 tests

Tests pour classe Equipe:
-----
Test 17: test de calculAttaque de la classe Equipe : OK
Test 18: test de calculDefense de la classe Equipe : OK
Test 19: getter de la classe Equipe : OK
Test 20: operateur << de la classe Equipe : OK
-----
Total des tests qui passent pour la section: 19/19 tests
```

```
Tests pour classe Match:
-----
Test 21: Constructeur par défaut de la classe Match           : OK
Test 22: Constructeur par parametre de la classe Match       : OK
Test 23: Constructeur par copie de la classe Match           : OK
Test 24: Constructeur par de la classe Match                 : OK
Test 25: Setters et getters de la classe Match partie 1     : OK
Test 26: Setters et getters de la classe Match partie 2     : OK
Test 27: fonction estVictorieuse de la classe Match          : OK
Test 28: operateur << de la classe Match                    : OK
-----
Total des tests qui passent pour la section: 26/26 tests

Tests pour classe EquipeSport:
-----
Test 29: fonction calculAttaque de classe EquipeSport       : OK
Test 30: fonction calculDefense de classe EquipeSport       : OK
-----
Total des tests qui passent pour la section: 2/2 tests
```

Affichage pour la classe Joueur : nom matchesJoués buts assists attaque défense

```
Stanley 10 5 6 13 2
```

Affichage pour la classe Gardien:

```
Gardien
Andrew 13 10 3 15 21
0.9 1.2
```

Affichage pour la classe Attaquant:

```
Attaquant
Drogba 189 250 235 98 89
30 45 12.5
```

Affichage pour la classe Défenseur:

```
Defenseur
Carles Puyol 32 3 13 35 80
1.2 1
```

Affichage pour la classe Equipe:

```
inconnue
Joueurs :
Stanley 10 5 6 12 15
Shea 81 23 30 85 80
Joueur3 82 21 40 13 60
Joueur4 82 21 40 77 89
Joueur5 82 21 40 60.5 45.4
Joueur6 82 21 40 55.7 78.8
Joueur7 82 21 40 45.3 98.5
Nombre de victoire>>s : 0
Somme de la force d'attaque 412.244
Somme de la force de defense 503.701
```

Affichage pour la classe Match:

```
Match - 2021-1-1

Arsenal vs Chelsea
Resultat : 0 - 0
```

Correction

La correction du TP se fait sur 20 points :

- [3 pt] Compilation du programme sans avertissement.
- [4 pt] Exécution du programme.
- [10 pt] Comportement exact de l'application.
- [2 pt] Qualité et documentation du code.
- [1 pt] Absence de fuites de mémoire.

Relisez bien les spécifications générales et les consignes de remise au début du TP avant de remettre votre travail!