



Remise du travail

- Date : 1 décembre 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 cplusplus, les notes de cours, et les forums sur Google peuvent vous donner de bonnes pistes de résolution!
- Veuillez utiliser la version v17 de C++.

 Tout cas de plagiat sera automatiquement reporté au Comité d'examen de fraude (CEF) 

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 (;).
- 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 la STL. Dans ce TP, vous aurez donc à implémenter certaines méthodes les classes et des foncteurs de manière à faire passer tous les tests. Attention, de façon générale, il est interdit d'utiliser des boucles.

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 qui représente un match entre deux équipes.
- ***Tournoi*** : Classe qui représente un tournoi entre plusieurs équipes.

Classe Date

La classe représente une date ayant les attributs suivants : Année, Mois et jour. Elle est complètement implémentée. L'operator== a été implémenté et rajouté.

Classe Joueur

Classe qui représente un joueur d'une équipe. Il n'y a rien à changer dans cette classe. Elle est complètement implémentée.

Classe Gardien

Classe qui hérite de la classe *Joueur*. Il n'y a rien à changer dans cette classe. Elle est complètement implémentée.

Classe Attaquant

Classe qui hérite de la classe *Joueur*. Il n'y a rien à changer dans cette classe. Elle est complètement implémentée.

Classe Équipe

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

Il faut implémenter

- **getJoueur()** : il faut chercher le joueur dont le nom est passé en paramètre dans l'attribut joueurs_. Il faut penser à utiliser l'algorithme find_if. Si le joueur n'est pas trouvé on retourne nullptr. Il est interdit d'utiliser une ou des boucles.
- **getJoueurs()** : cette méthode trie l'attribut joueurs_ selon l'ordre alphabétique du nom des joueurs. Il faut utiliser l'algorithme sort et un foncteur. La méthode retourne le nouveau vector trié. Il est interdit d'utiliser une ou des boucles.
- **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 faut ré-implémenter cette méthode avec l'algorithme accumulate et le foncteur fRand. Il est interdit d'utiliser une ou des boucles.
- **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 faut ré-implémenter cette méthode avec l'algorithme accumulate et le foncteur fRand. Il est interdit d'utiliser une ou des boucles.
- **la surcharge operator -** qui retire un membre de l'équipe. Il faut l'implémenter avec l'algorithme remove_if et erase. Il est interdit d'utiliser une ou des boucles.
- **la surcharge operator+** qui ajoute un membre de l'équipe.

Classe Match

Classe qui représente un match entre 2 équipes. Elle n'est plus générique. Il n'y a rien à changer dans cette classe. Elle est complètement implémentée.

Classe Tournoi

Classe qui représente un tournoi entre différentes équipes.

- matches_ est un attribut qui contient tous les matchs ordonnés selon leur date.
- historiqueMatches_ est un attribut qui contient un map des anciens match selon leur date.
- equipes_ : conteneur qui conserve toutes les équipes qui participent au tournoi.
- **ajouterEquipe()** : ajoute une équipe dans la liste equipes_. Il faut vérifier que l'équipe n'est pas dans la liste equipes_. Il est interdit de faire une boucle. Pensez à utiliser l'algorithme count.
- **ajouterMatch()** : ajoute un match dans le multiset de matchs triés par date et met à jour historiqueMatches_. Il est interdit de faire une boucle. Pensez à utiliser l'algorithme intégré find d'un multiset et souvenez-vous qu'il est possible d'avoir plusieurs matchs ayant lieu à la même date.

- `getMatchesParDate()` : retourne tous les matchs qui ont eu lieu à cette date. Il est interdit de faire une boucle. Il faut penser à la méthode `equals` de la classe `vector` et l'algorithme `transform`.
- `getEquipes()` : retourne toutes les équipes inscrites au tournoi dans l'ordre spécifié par le comparateur.
- `attribuerMatches()` : une méthode privée déjà implémentée. Elle simule l'attribution aléatoire d'équipes dans plusieurs matchs qui sont ajoutés dans un tournoi et retourne ces matchs dans un multiset.
- `simulerTournoi()` méthode déjà implémentée. Simule un tournoi avec des matchs entre plusieurs équipes.

Fichier de Foncteurs.h

Le fichier `foncteurs.h` contient tous les foncteurs. Vous êtes libre d'ajouter d'autres foncteurs pour les appels aux algorithmes de la STL ou d'utiliser les fonctions `lambda` dans les algorithmes.

FoncteurComparerParVictoires : est un foncteur générique ayant 2 opérateurs ()

- Operator 1 : `operator ()` qui reçoit 2 pointeurs bruts et compare (`<`) le nombre de victoires de chaque équipe.
- Operator 2 `operator()` qui reçoit 2 `shared_ptr` et qui fait appel à l'opérateur1 avec les pointeurs bruts des `shared_ptr`.

FoncteurComparerParNom : est un foncteur générique ayant 2 opérateurs

- Operator 1 : `operator ()` qui reçoit 2 pointeurs bruts et compare (`<`) le nom de chaque équipe
- Operator 2 `operator()` qui reçoit 2 `shared_ptr` et qui fait appel à l'opérateur1 avec les pointeurs bruts des `shared_ptr`.

FoncteurComparerParDate : est un foncteur générique ayant 2 opérateurs

- Operator 1 : `operator ()` qui reçoit 2 pointeurs bruts et compare (`<`) la date de chaque équipe.
- Operator 2 `operator()` qui reçoit 2 `shared_ptr` et qui fait appel à l'opérateur1 avec les pointeurs bruts des `shared_ptr`.
- Operator 3: `operator()` qui reçoit 2 `unique_ptr` et qui fait appel à l'opérateur1 avec les pointeurs bruts des `unique_ptr`.

main.cpp

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

```
Tests pour classe foncteurs:
-----
Test 1: FoncteurComparerParVictoires de la classe foncteur      : OK
Test 2: FoncteurComparerParNom de la classe foncteur            : OK
Test 3: FoncteurComparerParDate de la classe foncteur          : OK
-----
Total des tests qui passent pour la section: 8/8 tests

Tests pour classe Equipe:
-----
Test 4: Constructeurs de la classe Equipe                      : OK
Test 5: Operateur+ et Getters de la classe Equipe              : OK
Test 6: Operateur- et Getters de la classe Equipe              : OK
-----
Total des tests qui passent pour la section: 7/7 tests

Tests pour classe Tournoi:
-----
Erreur : l'equipe fait deja partie du tournoi.
Test 7: ajouterEquipe de la classe Tournoi                      : OK
Test 8: ajouterMatch de la classe Tournoi                      : OK
Test 9: getMatchParDate de la classe Tournoi                   : OK
Test 10: getEquipes de la classe Tournoi                       : OK
-----
Total des tests qui passent pour la section: 11/11 tests

Total pour tous les tests: 10/10 arrondi au dixieme
```

Correction

La correction du TP se fait sur 20 points :

- [3 pt] Compilation du programme sans avertissement.
- [3 pt] Exécution du programme.
- [10 pt] Comportement exact de l'application.
- [2 pt] Qualité et documentation du code.
- [2 pt] Bons algorithmes utilisés.

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