Commencé le	mercredi 13 décembre 2023, 13:30
État	Terminé
Terminé le	mercredi 13 décembre 2023, 15:53
Temps mis	2 heures 22 min
Note	**18,38** sur 20,00 (**91,88**%)

Description

Indiquez l'ordre de complexité de temps des algorithmes ci-dessous. On considère l'ordre moyen et/ou amorti. S'il y a plusieurs possibilités, indiquez l'ordre qui est le plus petit possible.

La fonction *f* a un temps en *O*(1) et *range* n'ajoute pas de complexité par rapport à une boucle faite à la main.

Question 1

Terminé

Note de 0,00 sur 0,75

```
deque<int> v;
for (int i : range(n))
    v.insert(v.begin(), f(i));
```

Est *O*(n^2)

Votre réponse est incorrecte.

Question 2

Terminé

Note de 0,75 sur 0,75

```
set<int> v;
for (int i : range(n))
    v.insert(f(i));
```

Est *O*($n \cdot \log(n)$)

Votre réponse est correcte.

Question 3

Terminé

Note de 0,00 sur 0,75

```
string v = "x";
for (int i : range(n))
    v += v;
```

Est *O*(n^2)

Votre réponse est incorrecte.

Question 4

Terminé

Note de 1,00 sur 1,00

Choisissez dans la liste **TOUTES** les affirmations **VRAIES** au sujet de l'architecture modèle-vue-controlleur

- [ ] a. Le modèle doit connaitre les contrôleurs.
- [x] b. Le modèle informe la vue lorsqu'il y a un changement dans son état.
- [x] c. Le modèle doit fournir les méthodes nécessaires pour manipuler ses données.
- [ ] d. Le modèle doit adapter ses données à la sortie.

Votre réponse est correcte.

Description

Soit le code suivant qui représente une situation d'étudiants qui suivent un cours en ligne. Le code ne compile bien sûr pas et des commentaires sont mis à certains endroit pour remplacer le code fonctionnel. Ce code est la base pour la question qui suit.

```
1. struct ErreurCoursEnLigne : std::logic_error {
2.     using logic_error::logic_error;
3. };
4.
5. struct ErreurIntraTropDifficile : ErreurCoursEnLigne {
6.     using ErreurCoursEnLigne::ErreurCoursEnLigne;
7. };
8.
9. struct ErreurRienCompris : ErreurIntraTropDifficile {
10.    using ErreurIntraTropDifficile::ErreurIntraTropDifficile;
11. };
12.
13. struct ErreurPasAssezÉtudié : ErreurIntraTropDifficile {
14.    using ErreurIntraTropDifficile::ErreurIntraTropDifficile;
15. };
16.
17. struct ErreurTwitch : ErreurCoursEnLigne {
18.    using ErreurCoursEnLigne::ErreurCoursEnLigne;
19. };
20.
21. struct ErreurColocsDuChargéOntRebranchéLeRouteur : ErreurTwitch {
22.    using ErreurTwitch::ErreurTwitch;
23. };
24.
25.
26. class Étudiant {
27. public:
28.    void étudier(bool pourVrai) {
29.        if (pourVrai) {
30.            // Refaire ses TD par soi-même et refaire les exemples faits en classe. :D
31.            prêtPourExamen_ = true;
32.        } else {
33.            // Meh, relire les notes de cours une fois la veille de l'exam. :(
34.            prêtPourExamen_ = false;
35.        }
36.    }
37.
38.    void assisterAuCours() {
39.        // Assister au stream sur Twitch.
40.        bool coursFini = false;
41.        while (not coursFini) {
42.            if (not streamTwitchActif())
43.                throw ErreurColocsDuChargéOntRebranchéLeRouteur("Ah come on!");
44.
45.            coursFini = /* Est-ce que le chargé a fini de monologuer et/ou commence à avoir faim? */;
46.        }
47.    }
48.
49.    void faireIntra() {
50.        double note = /* Est-ce que ça c'est bien passé? */;
51.        if ( note < 10/20.0 ) {
52.            if (not prêtPourExamen_)
53.                throw ErreurPasAssezÉtudié("Oups! Les questions demandaient de comprendre...");
54.            else
55.                throw ErreurRienCompris("De toute façon, les examens sont juste des constructions sociales.");
56.        } else if (note <= 15/20.0) {
57.            cout << "Pas pire, pas pire." << "\n";
58.        } else if (note <= 20/20.0) {
59.            cout << "Yay!" << "\n";
60.        }
61.    }
62.
63. private:
64.    bool prêtPourExamen_;
65. };
```

La situation présentée relève d'une oeuvre de fiction. Toute ressemblance avec des situations existantes ou ayant existé est une coïncidence.

Question 5

Terminé

Note de 1,50 sur 1,50

Soit le code suivant :

```
1. int main() {
2.     Étudiant lucLeroux;
3.     try {
4.         lucLeroux.étudier(false);
5.         lucLeroux.faireIntra();
6.     } catch (ErreurPasAssezÉtudié& e) {
7.         // Catch 1
8.     } catch (ErreurRienCompris& e) {
9.         // Catch 2
10.    } catch (ErreurIntraTropDifficile& e) {
11.        // Catch 3
12.    } catch (ErreurCoursEnLigne& e) {
13.        // Catch 4
14.    }
15.
16.    // Fin du programme
17. }
```

Qu'arrive-t-il si `étudier()` lance une exception autre que celles présentées (par exemple `std::bad_alloc` car l'élève a essayé de se bourrer de trop de matière trop tard), et si on présume que l'élève obtiendrait une mauvaise note (<10/20)?

Veuillez choisir une réponse.

- a. L'exception n'est pas gérée par le try-catch et fait planter le programme dans l'appel de `étudier()`.
- b. Le catch 4 est exécuté car c'est le plus générique, puis le programme termine normalement.
- c. Le catch 2 est exécuté en raison de la mauvaise note, puis le programme plante en raison de l'exception non-gérée.

Votre réponse est correcte.

Question 6

Correct

Note de 2,00 sur 2,00

Cette question devrait être faite sans utiliser un compilateur.

On désire déterminer l'ordre de construction des différents objets/sous-objets pour une déclaration donnée. On ne listera pas dans cet ordre la *construction* des types fondamentaux, qui n'ont pas de constructeur.

Soit les définitions de classes suivantes (dans des fichiers séparés et on suppose que les *include* sont faits correctement pour que ça compile):

```
class A : public C {
public:
   A() { att2_ = new D(); }
private:
   B att1_;
   D* att2_;
};
class B {
public:
   B() { }
};
class C {
public:
   C() { }
};
class D : public B, public C {
public:
   D() { }
private:
   C att1_;
   B* att2_;
};
```

Soit le programme suivant:

```
int main() {
   A x;
}
```

On veut savoir l'ordre de construction lorsqu'on exécute ce programme. Indice: il y a ***7 objets/sous-objets*** construits.

Indiquez uniquement les noms des types dans le bon ordre, exemple: **A B C D A B C**

Réponse : (régime de pénalités : 0 %)

```
Indiquez l'ordre de début de construction pour les classes ci-haut:
ACBDBCC

Indiquez l'ordre de début d'exécution du corps des constructeurs pour les classes ci-haut:
CBABCCD
```

Votre réponse	Devrait être	
A C B D B C C	A C B D B C C	✔
C B A B C C D	C B A B C C D	✔

Réponse bien enregistrée: ACBDBCC, CBABCCD

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 2,00/2,00.

Description

Soient les classes suivantes

```
class Personne {
public:
    Personne(string nom) : nom_(nom) {}
    string getNom() { return nom_; }
    string getRole(){ return "aucun"; }
private:
    string  nom_;
};
```

```
class Arbitre {
public:
    Arbitre(string nom, string role) : Personne(nom), role_(role) {
        nbMatchArbitre_ = Aleatoire(10, 50)();
    }
    string  getRole() { return role_; }
    int getNbMatchArbitre() { return nbMatchArbitre_; }
    void setNbMatchArbitre(int nb) { nbMatchArbitre_ = nb; }
private:
    string role_;
    int nbMatchArbitre_;
};
```

Question 7

Terminé

Note de 0,50 sur 0,50

Que représente **Aleatoire**, utilisé dans le constructeur de la classe **Arbitre** ?

Veuillez choisir une réponse.

1. une méthode de la classe Arbitre
2. une fonction globale qui retourne une valeur aléatoire
3. une fonction générique qui retourne une valeur aléatoire
4. une classe foncteur
5. un opérateur de la classe Arbitre

Votre réponse est correcte.

Question 8

Terminé

Note de 1,00 sur 1,00

Indiquer tous les changements nécessaires dans les deux classes, afin de faire du polymorphisme sur la méthode getRole().

Veuillez choisir au moins une réponse.

- [ ] 1. Dans la classe Arbitre:
  private:
  string nom_;
  string role_;
- [ ] 2. virtual string getRole() { return role_; }
- [ ] 3. Dans la classe Personne:
  private:
  string nom_;
  string role_;
- [ ] 4. virtual string getNom() { return nom_; }
- [x] 5. class Arbitre : public Personne
- [x] 6. virtual string getRole() { return "aucun"; }

Votre réponse est correcte.

Question 9

Terminé

Note de 0,88 sur 1,00

Identifier les éléments adéquats manquants afin d'appliquer le polymorphisme sur la méthode getRole(). Vous n'avez pas à remplir les "...", c'est du code qui fait déjà correctement ce qui est indiqué, mais qui est omis de la question car non nécessaire à la compréhension du polymorphisme.

```
set< Personne* , comparerPersonne> /*rien*/  listePersonne;
listePersonne.insert( new Arbitre("Sophie","principal") ); // Ajouter une Arbitre
// ... d'autres Personne et Arbitre sont ajoutés
for ( Personne* p  :  listePersonne ) { // Parcourir  listePersonne
   cout << // ... affiche le nom
        << " a le role " << p->getRole() ;
   if ( auto a = dynamic_cast<Arbitre*>(p) ) // Afficher le nombre de matchs arbitrés par l'arbitre
      cout << " a arbitré " << p->getNbMatchArbitre()  << endl;
   else
      cout << endl;
}
```

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 7.

Question 10

Terminé

Note de 1,50 sur 1,50

Écrire le foncteur prédicat binaire **comparerPersonne**, utilisable comme deuxième paramètre d'un conteneur **set** du bon type, qui reçoit comme paramètres deux pointeurs à des objets **Personne** et retourne le résultat de la comparaison selon l'ordre alphabétique croissant des noms des personnes.

Par exemple:

Test	Résultat

`Personne simone{"Simone"}, albert{"Albert"};`
`comparerPersonne{}(&simone, &albert);`
`cout << "ok";`
	ok

`Personne simone{"Simone"}, albert{"Albert"},`
`  marc{"Marc"}, sophie{"Sophie"};`
`cout << comparerPersonne{}(&simone, &albert)`
`  << comparerPersonne{}(&albert, &marc)`
`  << comparerPersonne{}(&sophie, &marc)`
`  << comparerPersonne{}(&simone, &sophie);`
	0101

Réponse : (régime de pénalités : 0 %)

```
struct comparerPersonne : std::binary_function<Personne*, Personne*, bool> {
    bool operator()(Personne* p1, Personne* p2) {
        return (p1->getNom() < p2->getNom());
    };
};
```

	Test	Résultat attendu	Résultat obtenu	

✔	`Personne simone{"Simone"}, albert{"Albert"};`
	`comparerPersonne{}(&simone, &albert);`
	`cout << "ok";`
		ok	ok	✔

✔	`Personne simone{"Simone"}, albert{"Albert"},`
	`  marc{"Marc"}, sophie{"Sophie"};`
	`cout << comparerPersonne{}(&simone, &albert)`
	`  << comparerPersonne{}(&albert, &marc)`
	`  << comparerPersonne{}(&sophie, &marc)`
	`  << comparerPersonne{}(&simone, &sophie);`
		0101	0101	✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 1,50/1,50.

Commentaire : Question comparerPersonne. 1.5/1.5

Question 11

Terminé

Note de 1,25 sur 1,25

Soient les classes **Personne** et **Arbitre** des questions précédentes. Si on voulait que **Professeur** soit une **Personne**, et qu'un **ArbitreProfesseur** soit à la fois un **Arbitre** (qui est une **Personne**) et un **Professeur** mais ne soit qu'une seule **Personne**. Que doit-on minimalement faire?

- [x] a. Arbitre doit avoir Personne comme classe de base avec le mot clé **virtual**
- [ ] b. ArbitreProfesseur doit avoir comme classe de base directe Personne
- [x] c. Le constructeur de ArbitreProfesseur doit indiquer directement la construction de Personne
- [ ] d. ArbitreProfesseur doit utiliser l'héritage **protected** pour ses classes de base
- [x] e. Professeur doit avoir Personne comme classe de base avec le mot clé **virtual**
- [ ] f. ArbitreProfesseur doit avoir les classes de base Arbitre et Professeur avec le mot clé **virtual**

Votre réponse est correcte.

Description

Soient les classes Arbitre (sans polymorphisme, contrairement à la section précédente) et GestionnaireArbitre. L'attribut arbitres_ contient tous les arbitres quel que soit leur rôle. L'attribut map filtreArbitreRole_ a pour cle le rôle de l'arbitre, et comme valeur tous les arbitres qui ont ce rôle.

```
class Arbitre {
public:
    Arbitre (string nom, string role) : nom_(nom), role_(role){
        nbMatchArbitre_ = Aleatoire(10, 50)();
    }
    string getNom() const { return nom_; }
    string getRole() const { return role_; }
    int getNbMatchArbitre() const { return nbMatchArbitre_; }
    void setNbMatchArbitre(int nb) { nbMatchArbitre_ = nb; }
private:
    string  nom_;
    string role_;
    int nbMatchArbitre_;
};

class GestionnaireArbitre {
public:
    GestionnaireArbitre() = default;
    bool ajouterArbitre(Arbitre* a);
    bool supprimerArbitre(string nom);
    Arbitre* getArbitreParNom(string nom) const;
    friend ostream& operator << (ostream&  sortie,
           const GestionnaireArbitre& g);
protected:
    vector<Arbitre*>::const_iterator getIteratorArbitreParNom(string nom) const;
private:
    vector<Arbitre*> arbitres_;
    unordered_map<string, vector<Arbitre*>> filtreArbitreRole_;
};
```

Question 12

Terminé

Note de 1,00 sur 1,00

La fonction membre protégée **getIteratorArbitreParNom** retourne l'itérateur const vers l'arbitre du conteneur **arbitres_** qui a le nom spécifié en paramètre. En utilisant la STL, sans for/while, écrire en une ligne le corps de la fonction.

Par exemple:

Test	Résultat

`GestionnaireArbitre gestionArbitre;`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));`
`cout << gestionArbitre;`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`albert principal`
	`Le conteneur Map`
	`principal albert`

Réponse : (régime de pénalités : 0, 0, 0, 0, 0, 3, ... %)

Réinitialiser la réponse

```
1 vector<Arbitre*>::const_iterator GestionnaireArbitre::getIteratorArbitreParNom(string nom) const {
2     return find_if(arbitres_.begin(), arbitres_.end(), [&](Arbitre* a){return (a->getNom() == nom);} );
3 }
4
```

	Test	Résultat attendu	Résultat obtenu	

✔	`GestionnaireArbitre gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));`
	`cout << gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`albert\tprincipal`
		`Le conteneur Map`
		`principal\talbert\t`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`albert\tprincipal`
			`Le conteneur Map`
			`principal\talbert\t`
				✔

✔	`GestionnaireArbitre gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("Martine","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("Martine","secondaire"));`
	`cout << gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`Martine\tprincipal`
		`Le conteneur Map`
		`principal\tMartine\t`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`Martine\tprincipal`
			`Le conteneur Map`
			`principal\tMartine\t`
				✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 1,00/1,00.

Commentaire : Question getIteratorArbitreParNom. 1.0/1.0

Question 13

Terminé

Note de 1,00 sur 1,00

La fonction membre **getArbitreParNom** retourne le pointeur vers l'arbitre qui a le nom spécifié en paramètre, ou **nullptr** s'il n'y a pas d'arbitre de ce nom. Écrire le corps de cette fonction proprement en évitant la duplication de code.

Par exemple:

Test	Résultat

`GestionnaireArbitre gestionArbitre;`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));`
`cout << gestionArbitre;`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`albert  principal`
	`Le conteneur Map`
	`principal  albert`

Réponse : (régime de pénalités : 0, 0, 0, 0, 0, 3, ... %)

Réinitialiser la réponse

```
1  Arbitre* GestionnaireArbitre::getArbitreParNom(string nom) const
2  {
3      auto it = getIteratorArbitreParNom(nom);
4      if (it != arbitres_.end())
5          return *it;
6      else
7          return nullptr;
8  }
```

	Test	Résultat attendu	Résultat obtenu	

✔	`GestionnaireArbitre gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));`
	`cout << gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`albert\tprincipal`
		`Le conteneur Map`
		`principal\talbert\t`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`albert\tprincipal`
			`Le conteneur Map`
			`principal\talbert\t`
				✔

✔	`GestionnaireArbitre gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("Martine","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("Martine","secondaire"));`
	`cout << gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`Martine\tprincipal`
		`Le conteneur Map`
		`principal\tMartine\t`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`Martine\tprincipal`
			`Le conteneur Map`
			`principal\tMartine\t`
				✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 1,00/1,00.

Commentaire : Question getArbitreParNom. 1.0/1.0

Question 14

Terminé

Note de 2,00 sur 2,00

Écrire la fonction membre **ajouterArbitre** qui ajoute le pointeur d'Arbitre dans les deux attributs conteneurs de la classe **GestionnaireArbitre**. L'ajout doit se faire uniquement si le nom de l'arbitre n'est pas déjà présent. La fonction retourne faux dans le cas où il est déjà présent.

Cette fonction membre doit faire appel à la fonction membre **getArbitreParNom**.

Par exemple:

Test	Résultat

`GestionnaireArbitre gestionArbitre;`
`gestionArbitre.ajouterArbitre(new Arbitre("Martine","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("Damien","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("Raphael","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("Francois","secondaire"));`
`cout << gestionArbitre;`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`Martine principal`
	`Damien secondaire`
	`Raphael secondaire`
	`Francois secondaire`
	`Le conteneur Map`
	`secondaire Damien Raphael Francois`
	`principal Martine`

`GestionnaireArbitre gestionArbitre;`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("charles","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));`
`cout << gestionArbitre;`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`albert principal`
	`marc secondaire`
	`william secondaire`
	`charles principal`
	`Le conteneur Map`
	`secondaire marc william`
	`principal albert charles`

`GestionnaireArbitre gestionArbitre;`
`cout`
`<< gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"))`
`<< gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire"))`
`<< gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire"))`
`<< gestionArbitre.ajouterArbitre(new Arbitre("marc","principal"))`
`<< gestionArbitre.ajouterArbitre(new Arbitre("charles","principal"))`
`<< gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"))`
`<< endl;`
`cout << gestionArbitre;`
	`111010`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`albert\tprincipal`
	`marc\tsecondaire`
	`william\tsecondaire`
	`charles\tprincipal`
	`Le conteneur Map`
	`secondaire\tmarc\twilliam\t`
	`principal\talbert\tcharles\t`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  bool GestionnaireArbitre::ajouterArbitre(Arbitre* a) {
2      if (!getArbitreParNom(a->getNom())) {
3          arbitres_.push_back(a);
4          filtreArbitreRole_[a->getRole()].push_back(a);
5          return true;
6      }
7      else
8          return false;
9  }
10 // à compléter
11
```

	Test	Résultat attendu	Résultat obtenu	

✔	GestionnaireArbitre gestionArbitre;
	gestionArbitre.ajouterArbitre(new Arbitre("Martine","principal"));
	gestionArbitre.ajouterArbitre(new Arbitre("Damien","secondaire"));
	gestionArbitre.ajouterArbitre(new Arbitre("Raphael","secondaire"));
	gestionArbitre.ajouterArbitre(new Arbitre("Francois","secondaire"));
	cout << gestionArbitre;
		le contenu de gestionnaire des arbitres
		Le conteneur Vector
		Martine\tprincipal
		Damien\tsecondaire
		Raphael\tsecondaire
		Francois\tsecondaire
		Le conteneur Map
		secondaire\tDamien\tRaphael\tFrancois\t
		principal\tMartine\t
			le contenu de gestionnaire des arbitres
			Le conteneur Vector
			Martine\tprincipal
			Damien\tsecondaire
			Raphael\tsecondaire
			Francois\tsecondaire
			Le conteneur Map
			secondaire\tDamien\tRaphael\tFrancois\t
			principal\tMartine\t
				✔

✔	GestionnaireArbitre gestionArbitre;
	gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));
	gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire"));
	gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire"));
	gestionArbitre.ajouterArbitre(new Arbitre("charles","principal"));
	gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire"));
	cout << gestionArbitre;
		le contenu de gestionnaire des arbitres
		Le conteneur Vector
		albert\tprincipal
		marc\tsecondaire
		william\tsecondaire
		charles\tprincipal
		Le conteneur Map
		secondaire\tmarc\twilliam\t
		principal\talbert\tcharles\t
			le contenu de gestionnaire des arbitres
			Le conteneur Vector
			albert\tprincipal
			marc\tsecondaire
			william\tsecondaire
			charles\tprincipal
			Le conteneur Map
			secondaire\tmarc\twilliam\t
			principal\talbert\tcharles\t
				✔

✔	GestionnaireArbitre gestionArbitre; cout << gestionArbitre.ajouterArbitre(new Arbitre("albert","principal")) << gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire")) << gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire")) << gestionArbitre.ajouterArbitre(new Arbitre("marc","principal")) << gestionArbitre.ajouterArbitre(new Arbitre("charles","principal")) << gestionArbitre.ajouterArbitre(new Arbitre("albert","secondaire")) << endl; cout << gestionArbitre;	111010 le contenu de gestionnaire des arbitres Le conteneur Vector albert\tprincipal marc\tsecondaire william\tsecondaire charles\tprincipal Le conteneur Map secondaire\tmarc\twilliam\t principal\talbert\tcharles\t	111010 le contenu de gestionnaire des arbitres Le conteneur Vector albert\tprincipal marc\tsecondaire william\tsecondaire charles\tprincipal Le conteneur Map secondaire\tmarc\twilliam\t principal\talbert\tcharles\t	✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 2,00/2,00.

Commentaire : Question ajouterArbitre. 2.0/2.0

Question 15

Terminé

Note de 2,00 sur 2,00

La fonction membre **supprimerArbitre**, de la classeGestionnaireArbitre, doit supprimer un arbitre des deux conteneurs attributs selon son nom si cet arbitre existe. La fonction retourne vrai si l'arbitre existait et a donc été supprimé. Compléter le code proprement, en évitant la duplication, sans for/while.

Par exemple:

Test	Résultat

`gestionArbitre.supprimerArbitre("albert");`
`cout << "suppression" << endl;`
`cout << gestionArbitre;`
	`suppression`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`marc  secondaire`
	`william secondaire`
	`charles principal`
	`Le conteneur Map`
	`secondaire  marc  william`
	`principal  charles`

`gestionArbitre.supprimerArbitre("albert");`
`gestionArbitre.supprimerArbitre("sophie");`
`cout << "suppression" << endl;`
`cout << gestionArbitre;`
	`suppression`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`marc  secondaire`
	`william secondaire`
	`charles principal`
	`Le conteneur Map`
	`secondaire  marc  william`
	`principal  charles`

`cout`
`  << gestionArbitre.supprimerArbitre("albert")`
`  << gestionArbitre.supprimerArbitre("sophie");`
`cout << endl << "suppression" << endl;`
`cout<< gestionArbitre;`
	`10`
	`suppression`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`marc\tsecondaire`
	`william\tsecondaire`
	`charles\tprincipal`
	`Le conteneur Map`
	`secondaire\tmarc\twilliam\t`
	`principal\tcharles\t`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
bool GestionnaireArbitre::supprimerArbitre(string nom) {
    auto a = getArbitreParNom(nom);
    if (a) {
        auto v = &filtreArbitreRole_[a->getRole()];
        auto it = remove_if(v->begin(), v->end(), [&](Arbitre* a){return (a->getNom() == nom);});
        v->erase(it, v->end());
        // ^ Il y a des répétitions de getIteratorArbitreParNom(nom) mais
        // si on fait v->erase(getIteratorArbitreParNom(nom)) il y a une erreur de segmentation
        // (invalidition de l'itérateur car retrait)

        arbitres_.erase(getIteratorArbitreParNom(nom));
        return true;
    }
    return false;
}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`gestionArbitre.supprimerArbitre("albert");`
	`cout << "suppression" << endl;`
	`cout << gestionArbitre;`
		`suppression`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`marc\tsecondaire`
		`william\tsecondaire`
		`charles\tprincipal`
		`Le conteneur Map`
		`secondaire\tmarc\twilliam\t principal\tcharles\t`
			`suppression`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`marc\tsecondaire`
			`william\tsecondaire`
			`charles\tprincipal`
			`Le conteneur Map`
			`secondaire\tmarc\twilliam\t principal\tcharles\t`
				✔

✔	`gestionArbitre.supprimerArbitre("albert");`
	`gestionArbitre.supprimerArbitre("sophie");`
	`cout << "suppression" << endl;`
	`cout << gestionArbitre;`
		`suppression`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`marc\tsecondaire`
		`william\tsecondaire`
		`charles\tprincipal`
		`Le conteneur Map`
		`secondaire\tmarc\twilliam\t principal\tcharles\t`
			`suppression`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`marc\tsecondaire`
			`william\tsecondaire`
			`charles\tprincipal`
			`Le conteneur Map`
			`secondaire\tmarc\twilliam\t principal\tcharles\t`
				✔

✔	`cout`
	`  <<`
	`gestionArbitre.supprimerArbitre("albert")`
	`  <<`
	`gestionArbitre.supprimerArbitre("sophie");`
	`cout << endl << "suppression" << endl;`
	`cout<< gestionArbitre;`
		`10`
		`suppression`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`marc\tsecondaire`
		`william\tsecondaire`
		`charles\tprincipal`
		`Le conteneur Map`
		`secondaire\tmarc\twilliam\t principal\tcharles\t`
			`10`
			`suppression`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`marc\tsecondaire`
			`william\tsecondaire`
			`charles\tprincipal`
			`Le conteneur Map`
			`secondaire\tmarc\twilliam\t principal\tcharles\t`
				✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 2,00/2,00.

Commentaire : Question supprimerArbitre. 2.0/2.0

Question 16

Terminé

Note de 2,00 sur 2,00

Compléter proprement la surcharge de l'opérateur <<, il y a le caractère "\t" entre les informations, mais pas à la fin des lignes. Préférer les boucles sur intervalles (celles avec :) aux anciens **for** (avec ; ;). Note: le compilateur supporte bien C++17.

Par exemple:

Test	Résultat

`GestionnaireArbitre  gestionArbitre;`
`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
`gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire"));`
`gestionArbitre.ajouterArbitre(new Arbitre("charles","principal"));`
`cout<< gestionArbitre;`
	`le contenu de gestionnaire des arbitres`
	`Le conteneur Vector`
	`albert\tprincipal`
	`marc\tsecondaire`
	`william\tsecondaire`
	`charles\tprincipal`
	`Le conteneur Map`
	`secondaire\tmarc\twilliam`
	`principal\talbert\tcharles`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```cpp
ostream& operator<< (ostream& sortie, const GestionnaireArbitre & g)
{
    sortie << "le contenu de gestionnaire des arbitres" << endl;
    sortie << "Le conteneur Vector" << endl;
    for(auto&& a : g.arbitres_) {
        sortie << a->getNom() << "\t" << a->getRole() << endl;
    }
    sortie << "Le conteneur Map" << endl;
    for(auto&& [r,v] : g.filtreArbitreRole_) {
        sortie << r;
        for(auto&& a : v) {
            sortie << "\t" << a->getNom();
        }
        sortie << endl;
    }
    sortie << endl;
    return sortie;
}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`GestionnaireArbitre  gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("albert","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("marc","secondaire"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("william","secondaire"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("charles","principal"));`
	`cout<< gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`albert\tprincipal`
		`marc\tsecondaire`
		`william\tsecondaire`
		`charles\tprincipal`
		`Le conteneur Map`
		`secondaire\tmarc\twilliam`
		`principal\talbert\tcharles`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`albert\tprincipal`
			`marc\tsecondaire`
			`william\tsecondaire`
			`charles\tprincipal`
			`Le conteneur Map`
			`secondaire\tmarc\twilliam`
			`principal\talbert\tcharles`
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`GestionnaireArbitre  gestionArbitre;`
	`gestionArbitre.ajouterArbitre(new Arbitre("A","principal"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("B","secondaire"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("C","secondaire"));`
	`gestionArbitre.ajouterArbitre(new Arbitre("D","principal"));`
	`cout << gestionArbitre;`
		`le contenu de gestionnaire des arbitres`
		`Le conteneur Vector`
		`A\tprincipal`
		`B\tsecondaire`
		`C\tsecondaire`
		`D\tprincipal`
		`Le conteneur Map`
		`secondaire\tB\tC`
		`principal\tA\tD`
			`le contenu de gestionnaire des arbitres`
			`Le conteneur Vector`
			`A\tprincipal`
			`B\tsecondaire`
			`C\tsecondaire`
			`D\tprincipal`
			`Le conteneur Map`
			`secondaire\tB\tC`
			`principal\tA\tD`
				✔

Tous les tests ont été réussis ! ✔

Correct

Note pour cet envoi : 2,00/2,00.

Commentaire : Question operateur affichage. 2.0/2.0

Question 17

Terminé

Non noté

Sur mon honneur, je (écrire votre nom) Alice Vergeau, affirme que j'ai fait cet examen par moi-même, sans communication avec personne (autre que les enseignants indiqués en première page en cas de problème), et selon les directives identifiées dans cet énoncé d'examen.

Question 18

Terminé

Non noté

Si nécessaire, inscrivez vos suppositions ici, en précisant pour chaque supposition le numéro de la question concernée.

Q8.

Pour faire du polymorphisme sur getRole() il faut :
5. class Arbitre : public Personne // Dans la déclaration d'Arbitre
6. virtual string getRole() { return "aucun"; } // dans la méthode de Personne

mais il faut aussi :
string getRole() **override** { return role_; }
Dans la classe Arbitre.

Q15.
Comme expliqué en commentaire, pour le filtreArbitreRole_, on ne peut pas utiliser la fonction prédéfinie pour avoir l'itérateur d'un arbitre selon son nom, sinon il y a une erreur de segmentation.

Q4. JUSTIFICATION SUPPLÉMENTAIRE // facultatif //
a. Le modèle doit connaitre les contrôleurs. // FAUX. est écrit dans le cours "Le Modèle ne connait pas les contrôleurs" (Cours 16, page 19)
b. Le modèle informe la vue lorsqu'il y a un changement dans son état. // VRAI. est écrit dans le cours "Le Modèle sert à : [...] Signaler aux vues que l'état de ses attributs a été modifié" (Cours 16. page 16)
c. Le modèle doit fournir les méthodes nécessaires pour manipuler ses données. // VRAI. est écrit dans le cours "Maintenir l'état des données et **fournir des méthodes d'accès et de modification** pour celles-ci" (Cours 16. page 16)
d. Le modèle doit adapter ses données à la sortie. // FAUX. est écrit dans le cours "Le Modèle ne formate pas les données pour la sortie: c'est le rôle de la Vue." (Cours 16. page 16). synonyme de formater = modifier, or si le Modèle ne modifie pas les données pour la sortie, alors il ne les adapte pas