Commencé le	mardi 24 mai 2022, 09:30
État	Terminé
Terminé le	mardi 24 mai 2022, 11:30
Temps mis	1 heure 59 min
Note	**12,75** sur 20,00 (**64**%)

Description

- Pour un examen, le **plagiat** inclut le fait de demander de l'aide à quelqu'un pour répondre aux questions et le copier-coller venant de sources dont vous ne possédez pas tous les droits (i.e. TP fait en équipe, notes de cours, site Internet), sans en citer la source. Toute personne qui aide un autre étudiant pendant l'examen commet une **fraude**.
- En cas de doute sur le sens d'une question, énoncez clairement toutes suppositions que vous faites, soit en commentaires dans le code, soit dans la dernière question de cet examen (qui est un espace pour écrire vos commentaires). Nous ne répondrons pas aux questions.
- Vous avez droit à toutes les notes de cours et, pour les questions de «programmation» (où vous devez écrire un programme) à un compilateur, mais CodeRunner est suffisant pour répondre et les questions n'ont pas été particulièrement faites pour l'utilisation d'un autre compilateur. Vos programmes doivent passer les tests CodeRunner.
- La sauvegarde se fait automatiquement lorsque vous changez de page, elle est déclenchée quand le bouton « Suivant » dans le bas de la page est utilisé ou lorsque vous changez de page en utilisant le bloc «Navigation du test ». Cette sauvegarde permet de limiter la perte d'information en cas de coupure avec Internet ou autre problème technique et de garder la session active. Après deux heures d'inactivité (c.-à-d. aucune interaction avec le serveur), vous êtes automatiquement déconnecté(e) de Moodle.
- En cas de perte de connexion à la fin de l'examen, la tentative est envoyée automatiquement.
- L'examen est sur 20 points, le barème de chaque exercice est indiqué dans le bloc «Navigation du test ».

Bon travail !

Question **1**

Terminer

Non noté

Sur mon honneur, je (écrire votre nom)

, 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 **2**

Non répondue

Non noté

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

Question **3**

Terminer

Note de 1,50 sur 1,50

Parmi les choix suivants, déterminez le prototype de la fonction **echange** pour que le programme principal suivant

```
#include <iostream>

int main()
{
    int a(10), b(20);
    std::cout << "Avant : a = " << a << " et b = " << b << std::endl;
    echange(a, b);
    std::cout << "Apres : a = " << a << " et b = " << b << std::endl;
    return 0;
}
```

affiche le texte suivant :

Avant : a = 10 et b = 20
Apres : a = 20 et b = 10

Veuillez choisir une réponse :

- a. void echange(const int& a, const int& b)
- b. void echange(int a, int b)
- c. void echange(int& a, int& b)
- d. int echange(int a, int b)

Votre réponse est correcte.

La réponse correcte est : void echange(int& a, int& b)

Question **4**

Terminer

Note de 1,00 sur 1,50

Soit la fonction suivante:

```
int maFonction(int a, double b)
```

Sélectionnez parmi les choix suivants toutes les fonctions qui surchargent **maFonction()**. Par exemple, on veut savoir si le choix *a* tout seul permet de surcharger **maFonction()**, indépendamment de son incompatibilité avec les autres choix.

Veuillez choisir au moins une réponse :

- a. `string maFonction(double a, double b)`
- b. `int maFonction(double a, double b)`
- c. `int maFonction(int p1, double p2, string p3)`
- d. `string maFonction(int a, double b)`

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 2.

Les réponses correctes sont :

```
int maFonction(int p1, double p2, string p3)
```

,

```
string maFonction(double a, double b)
```

,

```
int maFonction(double a, double b)
```

Question **5**

Terminer

Note de 1,50 sur 1,50

Quelles affirmations sont **vraies**?

Veuillez choisir au moins une réponse :

- [x] a. À l'intérieur d'une méthode, le mot-clé **this** est un pointeur vers l'objet courant.
- [ ] b. La présence d'un membre pointeur dans une classe indique automatiquement une relation d'agrégation.
- [ ] c. Le mot clé **const** suivant le prototype d'une méthode d'une classe signifie que les variables locales de la méthode sont constantes.
- [ ] d. Toutes les méthodes d'une classe doivent être définies publiques.
- [x] e. La surcharge de l'opérateur = (affectation par copie) de la classe **vector** fait une copie profonde (deep copy).

Votre réponse est correcte.

Les réponses correctes sont : À l'intérieur d'une méthode, le mot-clé **this** est un pointeur vers l'objet courant., La surcharge de l'opérateur = (affectation par copie) de la classe **vector** fait une copie profonde (deep copy).

Description

Ce bloc est répété sur plusieurs pages pour éviter la navigation inutile.

Vous décidez de programmer un planificateur de tâches.

Dans les questions CodeRunner de cette section, vous avez accès à toutes les méthodes nécessaires pour répondre aux questions. Elles utilisent la version du corrigé et non pas vos réponses précédentes ou suivantes. Les implémentations sont faites à l'extérieur de la classe.

Les déclarations suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur :

```
class Priorite {};

class Tache {
public:
        Tache(Priorite& priorite, const string& nom, const string& echeance = "sans echeance");
        Tache(const Tache& tache);

        //Permet d'ajouter une sous-tâche au vecteur de Tache sousTaches_ en la copiant.
        void ajouterSousTache(const Tache& tache);
private:
        string nom_;
        Priorite& priorite_;
        string echeance_;
        vector<unique_ptr<Tache>> sousTaches_;
};

class Projet {
public:
        Projet(string nom);

        //Permet de déplacer la Tache sousTache vers la Tache tacheParente
        void deplacerTache(const Tache& tacheParente, const Tache& sousTache);
        //Permet d'ajouter une tâche au vecteur de Tache taches_
        void ajouterTache(const Tache& tache);

        //Permet de trouver l'index dans le vecteur taches_ d'une tache donnée
        int trouverIndexTache(const Tache& tache);

private:
        string nom_;
        vector<unique_ptr<Tache>> taches_;
};

class Utilisateur {
public:
        void ajouterProjet(const shared_ptr<Projet>& projet);

private:
        vector<shared_ptr<Projet>> projets_;
};
```

Question **6**

Incorrect

Note de 0,00 sur 2,50

Écrire l'implémentation du constructeur par copie de la classe **Tache** en évitant toutes redondances. N'oubliez pas que vous avez accès à la méthode **ajouterSousTache()**.

Par exemple:

Test	Résultat

`Priorite priorite1;`
`Tache tache1(priorite1, "Revision Intra", "1 janvier 1970");`
`Tache tache2 = tache1;`
`cout << (tache2.getNom() == tache1.getNom()) << "\n"`
`<< (tache2.getEcheance() == tache1.getEcheance()) << "\n"`
`<< (&tache2.getPriorite() == &tache1.getPriorite()) << "\n";`
	`true`
	`true`
	`true`

Réponse : (régime de pénalités : 0 %)

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Solution de l'auteur de la question (Cpp):

```cpp
Tache::Tache(const Tache& tache) :
    Tache(tache.priorite_, tache.nom_, tache.echeance_)
{
    for (auto& sousTache : tache.sousTaches_)
        ajouterSousTache(*sousTache);
}
```

Incorrect

Note pour cet envoi : 0,00/2,50.

Description

Ce bloc est répété sur plusieurs pages pour éviter la navigation inutile.

Vous décidez de programmer un planificateur de tâches.

Dans les questions CodeRunner de cette section, vous avez accès à toutes les méthodes nécessaires pour répondre aux questions. Elles utilisent la version du corrigé et non pas vos réponses précédentes ou suivantes. Les implémentations sont faites à l'extérieur de la classe.

Les déclarations suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur :

```
class Priorite {};

class Tache {
public:
        Tache(Priorite& priorite, const string& nom, const string& echeance = "sans echeance");
        Tache(const Tache& tache);

        //Permet d'ajouter une sous-tâche au vecteur de Tache sousTaches_ en la copiant.
        void ajouterSousTache(const Tache& tache);
private:
        string nom_;
        Priorite& priorite_;
        string echeance_;
        vector<unique_ptr<Tache>> sousTaches_;
};

class Projet {
public:
        Projet(string nom);

        //Permet de déplacer la Tache sousTache vers la Tache tacheParente
        void deplacerTache(const Tache& tacheParente, const Tache& sousTache);
        //Permet d'ajouter une tâche au vecteur de Tache taches_
        void ajouterTache(const Tache& tache);

        //Permet de trouver l'index dans le vecteur taches_ d'une tache donnée
        int trouverIndexTache(const Tache& tache);

private:
        string nom_;
        vector<unique_ptr<Tache>> taches_;
};

class Utilisateur {
public:
        void ajouterProjet(const shared_ptr<Projet>& projet);

private:
        vector<shared_ptr<Projet>> projets_;
};
```

Question **7**

Terminer

Note de 1,50 sur 1,50

Complétez l'implémentation de la fonction ajouterSousTache de la classe **Tache**.

```
void Tache::ajouterSousTache(const Tache& tache) {
    sousTaches_.push_back(make_unique<Tache>(tache));
}
```

Question **8**

Terminer

Note de 0,00 sur 1,50

Complétez l'implémentation de la fonction ajouterProjet de la classe **Utilisateur**. Notez que les projets sont partagés entre les utilisateurs.

```
void ajouterProjet(const shared_ptr<Projet>& projet) {
    projets_.push_back(shared_ptr<Projet>(projet.get());
}
```

Description

Ce bloc est répété sur plusieurs pages pour éviter la navigation inutile.

Vous décidez de programmer un planificateur de tâches.

Dans les questions CodeRunner de cette section, vous avez accès à toutes les méthodes nécessaires pour répondre aux questions. Elles utilisent la version du corrigé et non pas vos réponses précédentes ou suivantes. Les implémentations sont faites à l'extérieur de la classe.

Les déclarations suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur :

```
class Priorite {};

class Tache {
public:
        Tache(Priorite& priorite, const string& nom, const string& echeance = "sans echeance");
        Tache(const Tache& tache);

        //Permet d'ajouter une sous-tâche au vecteur de Tache sousTaches_ en la copiant.
        void ajouterSousTache(const Tache& tache);
private:
        string nom_;
        Priorite& priorite_;
        string echeance_;
        vector<unique_ptr<Tache>> sousTaches_;
};

class Projet {
public:
        Projet(string nom);

        //Permet de déplacer la Tache sousTache vers la Tache tacheParente
        void deplacerTache(const Tache& tacheParente, const Tache& sousTache);
        //Permet d'ajouter une tâche au vecteur de Tache taches_
        void ajouterTache(const Tache& tache);

        //Permet de trouver l'index dans le vecteur taches_ d'une tache donnée
        int trouverIndexTache(const Tache& tache);

private:
        string nom_;
        vector<unique_ptr<Tache>> taches_;
};

class Utilisateur {
public:
        void ajouterProjet(const shared_ptr<Projet>& projet);

private:
        vector<shared_ptr<Projet>> projets_;
};
```

Question **9**

Correct

Note de 3,50 sur 3,50

Complétez l'implémentation de la méthode deplacerTache de la classe **Projet**. Cette fonction prend en paramètres deux tâches d'un même projet et déplace le paramètre **sousTache** du vecteur **taches_** du projet courant vers le paramètre **tacheParente** (l'ajouter comme sous-tâche). Il n'est pas nécessaire de conserver l'ordre des éléments dans le vecteur **taches_** après avoir enlevé la tâche. Assumez que les tâches en paramètre sont dans le vecteur (ne traitez pas le cas de tâches invalides).

Par exemple:

Test	Résultat

`Priorite priorite1;`
`Tache tache1(priorite1, "Revision Intra", "1 janvier 1970");`
`Tache tache2(priorite1, "Composition & agregation", "1 janvier 1970");`
`Projet projet("INF1015");`
`projet.ajouterTache(tache1);`
`projet.ajouterTache(tache2);`
`projet.deplacerTache(tache1, tache2);`
`cout << projet.getNbTaches() << endl;`
`cout << projet[0].getNom() << endl;`
`cout << projet[0].getNbSousTaches() << endl;`
`cout << projet[0][0].getNom() << endl;`
	`1`
	`Revision Intra`
	`1`
	`Composition & agregation`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
void Projet::deplacerTache(const Tache& tacheParente, const Tache& sousTache) {
    unsigned int indexTacheParente = trouverIndexTache(tacheParente);
    unsigned int indexSousTache = trouverIndexTache(sousTache);

    taches_[indexTacheParente]->ajouterSousTache(sousTache);
    taches_.erase(taches_.begin() + indexSousTache);


}
```

	Test	Résultat attendu	Résultat obtenu	

✔	`Priorite priorite1;`
	`Tache tache1(priorite1, "Revision Intra", "1 janvier 1970");`
	`Tache tache2(priorite1, "Composition & agregation", "1 janvier 1970");`
	`Projet projet("INF1015");`
	`projet.ajouterTache(tache1);`
	`projet.ajouterTache(tache2);`
	`projet.deplacerTache(tache1, tache2);`
	`cout << projet.getNbTaches() << endl;`
	`cout << projet[0].getNom() << endl;`
	`cout << projet[0].getNbSousTaches() << endl;`
	`cout << projet[0][0].getNom() << endl;`
		`1`
		`Revision Intra`
		`1`
		`Composition & agregation`
			`1`
			`Revision Intra`
			`1`
			`Composition & agregation`
				✔

	Test	Résultat attendu	Résultat obtenu	

✔	`Priorite priorite1;`
	`Tache tache1(priorite1, "Revision Intra", "1 janvier 1970");`
	`Tache tache2(priorite1, "Composition & agregation", "1 janvier 1970");`
	`Projet projet("INF1015");`
	`projet.ajouterTache(tache1);`
	`projet.ajouterTache(tache2);`
	`projet.deplacerTache(tache1, tache2);`
	`try{`
	`  projet.trouverIndexTache(tache2);`
	`} catch(logic_error& e){`
	`  cout << e.what();`
	`}`
		`Tache inextistante`	`Tache inextistante`	✔

Tous les tests ont été réussis ! ✔

Solution de l'auteur de la question (Cpp):

```cpp
void Projet::deplacerTache(const Tache& tacheParente, const Tache& sousTache) {
    unsigned int indexTacheParente = trouverIndexTache(tacheParente);
    unsigned int indexSousTache = trouverIndexTache(sousTache);
    taches_[indexTacheParente]->ajouterSousTache(*taches_[indexSousTache]);
    if(indexSousTache != taches_.size() - 1)
        taches_[indexSousTache] = move(taches_.back());
    taches_.pop_back();
}
```

Correct

Note pour cet envoi : 3,50/3,50.

Description

Ce bloc est répété sur plusieurs pages pour éviter la navigation inutile.

Vous décidez de programmer un planificateur de tâches.

Dans les questions CodeRunner de cette section, vous avez accès à toutes les méthodes nécessaires pour répondre aux questions. Elles utilisent la version du corrigé et non pas vos réponses précédentes ou suivantes. Les implémentations sont faites à l'extérieur de la classe.

Les déclarations suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur :

```
class Priorite {};

class Tache {
public:
        Tache(Priorite& priorite, const string& nom, const string& echeance = "sans echeance");
        Tache(const Tache& tache);

        //Permet d'ajouter une sous-tâche au vecteur de Tache sousTaches_ en la copiant.
        void ajouterSousTache(const Tache& tache);
private:
        string nom_;
        Priorite& priorite_;
        string echeance_;
        vector<unique_ptr<Tache>> sousTaches_;
};

class Projet {
public:
        Projet(string nom);

        //Permet de déplacer la Tache sousTache vers la Tache tacheParente
        void deplacerTache(const Tache& tacheParente, const Tache& sousTache);
        //Permet d'ajouter une tâche au vecteur de Tache taches_
        void ajouterTache(const Tache& tache);

        //Permet de trouver l'index dans le vecteur taches_ d'une tache donnée
        int trouverIndexTache(const Tache& tache);

private:
        string nom_;
        vector<unique_ptr<Tache>> taches_;
};

class Utilisateur {
public:
        void ajouterProjet(const shared_ptr<Projet>& projet);

private:
        vector<shared_ptr<Projet>> projets_;
};
```

Question **10**

Terminer

Note de 0,75 sur 1,50

Identifiez les relations entre les différentes classes du planificateur.

1) La classe **string** (membres **echeance_** et **nom_**) est en composition par références dans la classe **Tache**.

2) La classe **Tache** est en composition par références dans la classe **Projet**.

3) La classe **Priorite** est en agrégation par valeurs dans la classe **Tache** .

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 3.
La réponse correcte est :
Identifiez les relations entre les différentes classes du planificateur.

1) La classe **string** (membres **echeance_** et **nom_**) est en [composition] [par valeurs] dans la classe **Tache**.

2) La classe **Tache** est en [composition] [par pointeurs (incluant intelligents)] dans la classe **Projet**.

3) La classe **Priorite** est en [agrégation] [par références] dans la classe **Tache** .

Question **11**

Correct

Note de 3,00 sur 3,00

Nous voulons une classe générique Vecteur, pour faire des vecteurs au sens mathématique, qui peut supporter tout type d'éléments pour lesquels les opérations arithmétiques ont du sens, i.e. int, double, complexe, fraction, etc.

On veut l'utilisation suivante :

```
Vecteur<int> a(3);
Vecteur<double> b(3);
// Avec déduction de type en C++17:
Vecteur c = { 1, 2, 3 };
Vecteur d = { 1.2, 2.2, 3.2 };
```

Pour la dernière déclaration, le constructeur a en paramètre un initializer_list<double> (où double est le type des éléments entre accolades).

On utilise les crochets **[ ]** pour accéder aux éléments du vecteur, et la méthode **size()** pour retourner sa taille.

On vous fournit une classe **Vecteur** fonctionnelle pour "int" que vous devez rendre générique. Ne changez pas les noms des méthodes/attributs et laissez la classe "friend" qui permet de faire les tests.

Le #define ACTIVER_TEST_ACCOLADES vous permet de désactiver/activer le dernier test, en commentant/décommentant la ligne. Ceci permet de vérifier sans ce test. Initialement, il est désactivé. Vous n'avez bien sûr pas tous vos points si vous le laissez désactivé.

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
#define ACTIVER_TEST_ACCOLADES
template<typename T>
class Vecteur {
    friend class TestsClass;

public:
    Vecteur(T taille) : v_(taille) {}
    Vecteur(initializer_list<T> valeurs) : v_(valeurs) {}  // Permet d'initialiser le vecteur avec des vale
    T& operator[](T i) { return v_[i]; }
    const T& operator[](T i) const { return v_[i]; }
    T size() const { return v_.size(); }
    Vecteur operator+(const Vecteur& rhs) const {
        if (size() != rhs.size())
            return {};
        Vecteur res(size());
        for (auto i : range(size()))
            res[i] = (*this)[i] + rhs[i];
        return res;
    }

private:
```

	Test	Résultat attendu	Résultat obtenu	

✔	`// Ça compile avec paramètre générique et size fonctionne encore:`
	`Vecteur<int> a(3);`
	`cout << a.size();`
		3	3	✔

✔	`// Les types des éléments sont bons:`
	`Vecteur<int> a(3);`
	`Vecteur<double> b(3);`
	`cout << is_same_v< type_des_elements_de<decltype(a)>, int >;`
	`cout << is_same_v< type_des_elements_de<decltype(b)>, double >;`
		11	11	✔

✔	`// Les crochets donnent ce qu'il faut:`
	`Vecteur<int> a(3);  const Vecteur<int> const_a(3);`
	`Vecteur<double> b(3);  const Vecteur<double> const_b(3);`
	`cout << is_same_v< decltype(a[0]), int& >;`
	`cout << is_same_v< decltype(b[0]), double& >;`
	`cout << is_same_v< decltype(const_a[0]), const int& >;`
	`cout << is_same_v< decltype(const_b[0]), const double& >;`
		1111	1111	✔

	Test	Résultat attendu	Résultat obtenu	

✔	`// L'addition fonctionne`
	`Vecteur<int> a(3);`
	`Vecteur<int> b(3);`
	`a[0] = 10; a[1] = 20; a[2] = 30;`
	`b[0] = 1; b[1] = 2; b[2] = 3;`
	`Vecteur<int> c = a + b;`
	`cout << c[0] << " " << c[1] << " " << c[2];`
		11 22 33	11 22 33	✔

✔	`// Les types des éléments sont bons avec les accolades:`
	`#ifdef ACTIVER_TEST_ACCOLADES`
	`Vecteur c = { 1, 2, 3 };`
	`Vecteur d = { 1.2, 2.2, 3.2 };`
	`cout << is_same_v< type_des_elements_de<decltype(c)>, int >;`
	`cout << is_same_v< type_des_elements_de<decltype(d)>, double >;`
	`#else`
	`cout << "Decommentez le #define";`
	`#endif`
		11	11	✔

Tous les tests ont été réussis ! ✔

Solution de l'auteur de la question (Cpp):

```cpp
#define ACTIVER_TEST_ACCOLADES

template <typename T>
class Vecteur {
    friend class TestsClass;

public:
    Vecteur(size_t taille) : v_(taille) {}
    Vecteur(initializer_list<T> valeurs) : v_(valeurs) {}  // Permet d'initialiser le vecteur avec des vale
    T&       operator[](size_t i)       { return v_[i]; }
    const T& operator[](size_t i) const { return v_[i]; }
    size_t size() const { return v_.size(); }
    Vecteur operator+(const Vecteur& rhs) const {
        if (size() != rhs.size())
            return {};
        Vecteur res(size());
        for (auto i : range(size()))
            res[i] = (*this)[i] + rhs[i];
        return res;
    }

```

Correct

Note pour cet envoi : 3,00/3,00.

Question **12**

Incorrect

Note de 0,00 sur 2,00

Écrivez la fonction d'ordre supérieur genererOndeSin() qui prend en paramètre l'amplitude A, la fréquence f en Hz et la phase φ (phi) en radian d'une fonction d'onde sinusoïdale $y(t) = A \sin(2\pi f t + \varphi)$ et retourne la fonction $y(t)$, où t est en secondes. Toutes les valeurs manipulées sont de type **double**. Les tests fournis vous montrent l'utilisation attendue.

Par exemple:

Test	Résultat

`auto f1 = genererOndeSin(10, 1, 0);`
`auto f2 = genererOndeSin(100, 0.1, pi/2);`
`cout << round(f1(0.25)) << "\n"`
`<< round(f2(2.5)) << "\n"`
`<< round(f1(-0.25)) << "\n"`
`<< round(f2(-2.5)) << "\n";`
	`10`
	`0`
	`-10`
	`0`

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  static const double pi = 3.141592653589793;
2
3  /
4
5
6
7
8
9
10
11
12
13
14
```

Solution de l'auteur de la question (Cpp):

```cpp
static const double pi = 3.141592653589793;

function<double(double)> genererOndeSin(double amplitude, double frequence, double phase) {
    return [=](double t) {
        return amplitude * sin(2*pi * frequence * t + phase);
    };
}
```

Incorrect

Note pour cet envoi : 0,00/2,00.

◄ Réponses aux questions de pratique

Aller à...

Quiz 2- Été-2022 ►