

[Tableau de bord](#) / [Mes cours](#) / [INF1010 - Programmation orientée objet](#) / [Contrôle Périodique](#) / [Contrôle Périodique - INF1010 - A2018](#)
/ [Prévisualisation](#)

Commencé le	mardi 10 mars 2020, 12:13
État	Terminé
Terminé le	jeudi 2 avril 2020, 17:12
Temps mis	23 jours 4 heures
En retard	23 jours 3 heures
Points	0,00/76,00
Note	0,00 sur 20,00 (0%)

Question **1**

Non répondue

Noté sur 2,00

Pour respecter le principe d'encapsulation, il faut que les attributs soient privés dans la classe

Sélectionnez une réponse :

- ☐ Vrai
- ☐ Faux

La réponse correcte est « Vrai ».

Question **2**

Non répondue

Noté sur 2,00

Le mot `this` est un mot réservé qui représente un attribut spécifique d'une classe

Sélectionnez une réponse :

- ☐ Vrai
- ☐ Faux

La réponse correcte est « Faux ».

Question **3**
Non répondue
Non noté

La fonction générique trouverSup qui prend comme paramètres un vecteur de type générique et une valeur du même type (le seuil). La fonction doit retourner un vecteur contenant tous les éléments du vecteur d'entrée qui sont supérieurs ou égaux au seuil. Identifier la signature de la fonction

Veuillez choisir une réponse :

☐ a.

```
template <typename T>
```

```
void trouverSup(const vector<T>& vec, const T& val)
```

☐ b.

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

☐ c.

```
template <typename T>
```

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

☐ d.

```
template <typename T>
```

```
vector trouverSup(const vector & vec, const T& val)
```

☐ e.

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

Votre réponse est incorrecte.

Les réponses correctes sont :

```
template <typename T>
```

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

```
,
```

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

```
,
```

```
template <typename T>
```

```
vector trouverSup(const vector & vec, const T& val)
```

```
,
```

```
vector<T> trouverSup(const vector<T>& vec, const T& val)
```

```
,
```

```
template <typename T>
```

```
void trouverSup(const vector<T>& vec, const T& val)
```

Question **4**

Non répondue

Noté sur 4,00

En C++, on transmet les objets par référence constante dans les paramètres d'une fonction, car

Veuillez choisir au moins une réponse :

- ☐ a. on veut modifier l'objet transmis par paramètre
- ☐ b. la transmission par valeur n'existe pas en c++
- ☐ c. on sauve du temps d'exécution
- ☐ d. la transmission par pointeur n'existe pas pour les objets.
- ☐ e. on sauve de l'espace mémoire
- ☐ f. on ne veut pas modifier l'objet

Votre réponse est incorrecte.

Les réponses correctes sont : on sauve du temps d'exécution, on sauve de l'espace mémoire, on ne veut pas modifier l'objet

Question **5**

Non répondue

Noté sur 2,00

On peut déclarer un objet x de ces deux façons:

MaClasse x;

ou

MaClasse x();

Sélectionnez une réponse :

- ☐ Vrai
- ☐ Faux

La réponse correcte est « Faux ».

Question **6**
Non répondue
Noté sur 6,00

Le Cookie Monster vous demande de l'aider à préparer sa sortie pour Halloween.
Pour cela, il voudrait que vous le représentiez en tant que classe C++.

Soit la classe **SacDeFriandises** qui possède les éléments suivants:

- Un constructeur par défaut
- Une méthode **ajouter** qui prend en paramètre une chaîne de caractères et l'ajoute à un vecteur interne
- Une méthode **attraperFriandise** qui va chercher une chaîne de caractères dans un vecteur interne, l'en retire, et la retourne

Vous devez écrire la classe suivante:

- La classe s'appelle **CookieMonster**
- Elle a un attribut **nombreDeCookiesManges_** de type entier
- Elle a un attribut **sacDeFriandises_** qui est un objet de type **SacDeFriandises** alloué dynamiquement
- Elle a un attribut **sacDeCookies_** qui est un tableau de chaînes de caractères alloué automatiquement de taille 5 (de toutes façons, le Cookie Monster les mangera toujours avant qu'ils arrivent dans le sac...)
- Elle a un constructeur par défaut qui initialise le nombre de cookies mangés à 10 (parce que bon, c'est quand même le minimum!)
- Elle a un constructeur par paramètres pour initialiser le nombre de cookies mangés
- Elle a, **seulement si nécessaire**, un destructeur
- Elle a une méthode **getNombreDeCookiesManges()** pour l'attribut **nombreDeCookiesManges_**
- Elle a une méthode **ajouterFriandise** qui ne retourne rien, prend en paramètre une chaîne de caractères et fait appel à la méthode **void ajouter(string friandise)** de l'objet **sacDeFriandises_**
- Elle a une méthode **mangerCookie** qui ne retourne rien, prend une friandise du **sacDeFriandises_** (méthode **string attraperFriandise()**) et, s'il s'agit d'un cookie (chaîne de caractères "**Cookie**"), on incrémente le nombre de cookies mangés.

Écrire la **définition seulement** de la classe **CookieMonster**. Les attributs ne doivent pas être initialisés.

```
1 class CookieMonster {
2 public:
3     CookieMonster();
4     CookieMonster(int nombreDeCookiesManges);
5     ~CookieMonster();
6     void ajouterFriandise(string friandise);
7     void mangerCookie();
8     int getNombreDeCookiesManges() const;
9 private:
10    int nombreDeCookiesManges_;
11    SacDeFriandises *sacDeFriandises_;
12    string sacDeCookies_[5];
13};
```

Question **7**

Non répondue

Noté sur 2,00

Écrire l'implémentation du constructeur par défaut tel que demandé dans l'énoncé ci-dessus.

1

```
CookieMonster::CookieMonster() : nombreDeCookiesManges_(10), sacDeFriandises_(nullptr) { sacDeFriandises = new SacDeFriandises(); }
```

Pour la notation, considérant les points moodle (4pts moodle = 1pt réel, 1pt moodle = 0.25pts réel), on retirera 1pt moodle par erreur (quelle qu'elle soit) jusqu'à concurrence de 0 à la question

Question **8**

Non répondue

Noté sur 2,00

Écrire l'implémentation du constructeur par paramètres tel que demandé dans l'énoncé ci-dessus.

1

2

```
CookieMonster::CookieMonster(int nombreDeCookiesManges) : nombreDeCookiesManges_(nombreDeCookiesManges), sacDeFriandises_(nullptr) { sacDeFriandises = new SacDeFriandises(); }
```

Question **9**

Non répondue

Noté sur 2,00

Écrire l'implémentation du destructeur tel que demandé dans l'énoncé ci-dessus.
Si le destructeur n'était pas nécessaire, écrivez exactement: "Non"

Réponse :

✖

La réponse correcte est : `CookieMonster::~CookieMonster() { delete sacDeFriandises_; }`

Question **10**

Non répondue

Noté sur 2,00

Écrire l'implémentation de la méthode `getNombreDeCookiesManges()` telle que demandée dans l'énoncé ci-dessus.

1

2

3

```
int CookieMonster::getNombreDeCookiesManges() const { return nombreDeCookiesManges_; }
```

Question **11**
Non répondue
Noté sur 2,00

Écrire l'implémentation de la méthode **ajouterFriandise(string friandise)** telle que demandée dans l'énoncé ci-dessus.

```
1 void CookieMonster::ajouterFriandise(string friandise) {  
2     sacDeFriandises_>ajouter(friandise);  
3 }
```

Question **12**
Non répondue
Noté sur 2,00

Écrire l'implémentation de la méthode **mangerCookie()** telle que demandée dans l'énoncé ci-dessus.

```
1 void CookieMonster::mangerCookie() {  
2     if (sacDeFriandises_>attraperFriandise() == "Cookie") {  
3         nombreDeCookiesManges_++;  
4     }  
5 }
```

Question **13**

Non répondue

Noté sur 4,00

Soient les classes

```
1 class PolyBook {
2     public:
3         // 2
4     private:
5         // 1
6 }
7
8 class Etudiant {
9     // ...
10 }
```

Identifier le constructeur et l'attribut de la classe PolyBook.

Si **PolyBook** est une classe composite par pointeur de **Etudiant**,

Choisir...

Si **PolyBook** est une classe agrégée par pointeur de **Etudiant**,

Choisir...

Si **PolyBook** est une classe agrégée par référence de **Etudiant**,

Choisir...

Si **PolyBook** est une classe composite par référence de **Etudiant**,

Choisir...

Si **PolyBook** est une classe composite de **Etudiant**,

Choisir...

Votre réponse est incorrecte.

La réponse correcte est :

Si **PolyBook** est une classe composite par pointeur de **Etudiant**,

→ 1. Etudiant *etud_; 2. PolyBook(const Etudiant& etud) : etud_(nullptr) { etud_ = new Etudiant(etud); },

Si **PolyBook** est une classe agrégée par pointeur de **Etudiant**,

→ 1. Etudiant *etud_; 2. PolyBook(Etudiant* etud) : etud_(etud) {},

Si **PolyBook** est une classe agrégée par référence de **Etudiant**,

→ 1. Etudiant &etud_; 2. PolyBook(Etudiant& etud) : etud_(etud) {},

Si **PolyBook** est une classe composite par référence de **Etudiant**,

→ Aucune solution,

Si **PolyBook** est une classe composite de **Etudiant**,

→ 1. Etudiant etud_; 2. PolyBook(const Etudiant& etud) : etud_(etud) {}

Question 14

Non répondue

Noté sur 4,00

Soit la classe **Etudiant** qui représente un étudiant identifié par son matricule et son nom.

```
1 class Etudiant {
2 public:
3     Etudiant(int matricule, string nom)
4         : matricule_(matricule), nom_(nom) {}
5     int getMatricule() const { return matricule_; }
6     string getNom() const { return nom_; }
7     bool operator==(const Etudiant& etudiant) const {
8         return matricule_ == etudiant.matricule_;
9     }
10 private:
11     int matricule_;
12     string nom_;
13 };
```

Soit la classe **Base**, composite d'**Etudiant**, et servant à garder une base de données d'étudiants.

Toutes les allocations de mémoire d'un objet **Etudiant** sont dynamiques et allouées par des méthodes de la classe.

```
1 class Base {
2 public:
3     Base();
4     Base(const Base& base);
5     ~Base();
6     Base& operator=(const Base& base);
7     bool ajouterEtudiant(int matricule, string nom);
8     string getNomEtudiant(int matricule) const;
9     bool retirerEtudiant(int matricule);
10 private:
11     vector<Etudiant*> etudiants_; // attribut composite
12 };
```

Pourquoi faut-il implémenter le constructeur par copie et surcharger l'opérateur d'affectation?

Veuillez choisir au moins une réponse :

- ☐ a. Pour éviter le partage des objets **Etudiant** lorsqu'une transmission par valeur d'un objet **Base**
- ☐ b. Pour partager les objets **Etudiant** entre deux objets de classe **Base**.
- ☐ c. Pour éviter le partage des objets **Etudiant** lors qu'une transmission par référence d'un objet **Base**
- ☐ d. Pour créer des nouveaux espaces mémoire d'objets **Etudiant** lors d'une affectation de deux objets de classe **Base**
- ☐ e. il est inutile d'implémenter le constructeur de copie et de surcharger l'opérateur d'affectation, car ceux-ci existent dans le conteneur **vector** de la STL

Votre réponse est incorrecte.

Les réponses correctes sont : Pour éviter le partage des objets **Etudiant** lorsqu'une transmission par valeur d'un objet **Base**, Pour créer des nouveaux espaces mémoire d'objets **Etudiant** lors d'une affectation de deux objets de classe **Base**

Question **15**

Non répondue

Noté sur 4,00

Soit la classe **Etudiant** qui représente un étudiant identifié par son matricule et son nom.

```
1 class Etudiant {
2 public:
3     Etudiant(int matricule, string nom)
4         : matricule_(matricule), nom_(nom) {}
5     int getMatricule() const { return matricule_; }
6     string getNom() const { return nom_; }
7     bool operator==(const Etudiant& etudiant) const {
8         return matricule_ == etudiant.matricule_;
9     }
10 private:
11     int matricule_;
12     string nom_;
13 };
```

Soit la classe **Base**, composite d'**Etudiant**, et servant à garder une base de données d'étudiants.

Toutes les allocations de mémoire d'un objet **Etudiant** sont dynamiques et allouées par des méthodes de la classe.

```
1 class Base {
2 public:
3     Base();
4     Base(const Base& base);
5     ~Base();
6     Base& operator=(const Base& base);
7     bool ajouterEtudiant(int matricule, string nom);
8     string getNomEtudiant(int matricule) const;
9     bool retirerEtudiant(int matricule);
10 private:
11     vector<Etudiant*> etudiants_; // attribut composite
12 };
```

Écrire la méthode **ajouterEtudiant**

Cette méthode doit retourner **false** si le matricule de l'étudiant existe déjà, **true** sinon. De plus, si le matricule de l'étudiant existe déjà, celui-ci ne sera **PAS** ajouté en double.

```
1 bool Base::ajouterEtudiant(int matricule, string nom) {
2     for (size_t i = 0; i < etudiants_.size(); i++) {
3         if (etudiants_[i]->getMatricule() == matricule) {
4             return false;
5         }
6     }
7     etudiants_.push_back(new Etudiant(matricule, nom));
8     return true;
9 }
```

Question 16

Non répondue

Noté sur 4,00

Soit la classe **Etudiant** qui représente un étudiant identifié par son matricule et son nom.

```

1 class Etudiant {
2 public:
3     Etudiant(int matricule, string nom)
4         : matricule_(matricule), nom_(nom) {}
5     int getMatricule() const { return matricule_; }
6     string getNom() const { return nom_; }
7     bool operator==(const Etudiant& etudiant) const{
8         return matricule_ == etudiant.matricule_;
9     }
10 private:
11     int matricule_;
12     string nom_;
13 };

```

Soit la classe **Base**, composite d'**Etudiant**, et servant à garder une base de données d'étudiants.

Toutes les allocations de mémoire d'un objet **Etudiant** sont dynamiques et allouées par des méthodes de la classe.

```

1 class Base {
2 public:
3     Base();
4     Base(const Base& base);
5     ~Base();
6     Base& operator=(const Base& base);
7     bool ajouterEtudiant(int matricule, string nom);
8     string getNomEtudiant(int matricule) const;
9     bool retirerEtudiant(int matricule);
10 private:
11     vector<Etudiant*> etudiants_; // attribut composite
12 };

```

Dans le code suivant, identifier les instructions manquantes:

```

Base& Base::operator=(const Base& base) {
    if (  ) {
        for (size_t i = 0; i < etudiants_.size(); i++) {
             ;
        }
        etudiants_.clear();
        for (size_t i = 0; i <  ; i++) {
            int matricule = base.etudiants_[i]->getMatricule();
            string nom = base.etudiants_[i]->getNom();
            Etudiant* e =  ;
            etudiants_. ;
        }
    }
    return  ;
}

```

Votre réponse est incorrecte.

La réponse correcte est : Soit la classe **Etudiant** qui représente un étudiant identifié par son matricule et son nom.

```

1 class Etudiant {
2 public:
3     Etudiant(int matricule, string nom)
4         : matricule_(matricule), nom_(nom) {}
5     int getMatricule() const { return matricule_; }
6     string getNom() const { return nom_; }
7     bool operator==(const Etudiant& etudiant) const{
8         return matricule_ == etudiant.matricule_;
9     }
10 private:
11     int matricule_;
12     string nom_;
13 };

```

Soit la classe **Base**, composite d'**Etudiant**, et servant à garder une base de données d'étudiants.

Toutes les allocations de mémoire d'un objet **Etudiant** sont dynamiques et allouées par des méthodes de la classe.

```
1 class Base {
2 public:
3     Base();
4     Base(const Base& base);
5     ~Base();
6     Base& operator=(const Base& base);
7     bool ajouterEtudiant(int matricule, string nom);
8     string getNomEtudiant(int matricule) const;
9     bool retirerEtudiant(int matricule);
10 private:
11     vector<Etudiant*> etudiants_; // attribut composite
12 };
```

Dans le code suivant, identifier les instructions manquantes:

```
Base& Base::operator=(const Base& base) {
    if ([this != &base]) {
        for (size_t i = 0; i < etudiants_.size(); i++) {
            [delete etudiants_[i]];
        }
        etudiants_.clear();
        for (size_t i = 0; i < [base.etudiants_.size()]; i++) {
            int matricule = base.etudiants_[i]->getMatricule();
            string nom = base.etudiants_[i]->getNom();
            Etudiant* e = [new Etudiant(matricule, nom)];
            etudiants_.push_back(e);
        }
    }
    return [*this];
}
```

Question **17**

Non répondue

Noté sur 4,00

Soit le code suivant:

```
1 class Point{
2 public:
3     Point(int x, int y);
4     void afficher() const;
5     ?? incrementer(??);
6 private:
7     int x_, y_;
8 };
9
10 Point::Point(int x, int y) {
11     x_ = x;
12     y_ = y;
13 }
14
15 void Point::afficher() const {
16     cout << "(" << x_ << "," << y_ << ")" << endl;
17 }
18
19 int main() {
20     Point p(3,4);
21     p.incrementer().incrementer().incrementer();
22     p.afficher(); // La sortie du programme est: (6,7)
23 }
```

Écrivez l'implémentation de la fonction **incrementer**:

- qui augmente d'une unité les coordonnées (en x et en y) d'une instance de la classe Point, et
- qui peut être appelée en cascade

```
1 Point& Point::incrementer() {
2     x_++;
3     y_++;
4     return *this ;
5 }
```

Question 18

Non répondue

Noté sur 20,00

Votre ami Jacquot, surnommé "*Le lent terne*" pour son humour sombre et désabusé, écrit un jeu vidéo permettant de fabriquer des lanternes à base de citrouilles. Jacquot vous montre les classes suivantes qu'il a commencé à écrire:

```

1  class Lanterne {
2  public:
3      Lanterne(string couleur, int lumens)
4          : couleur_(couleur), lumens_(lumens) {};
5  private:
6      string couleur_;
7      int lumens_;
8  };
9
10 class Lumiere {
11 public:
12     Lumiere() : lumens_(42) {};
13     int getLumens() const { return lumens_; }
14     void setLumens(int lumens) { lumens_ = lumens; }
15 private:
16     int lumens_;
17 };
18
19 class Citrouille {
20 public:
21     Citrouille() : couleur_("orange") {};
22     string getCouleur() const { return couleur_; }
23     void setCouleur(string couleur) { couleur_ =
24 couleur; }
25 private:
26     string couleur_;
27 };
28
29 int main () {
30     Citrouille citrouille;
31     Lumiere lumiere;
32     Lanterne lanterne = citrouille + lumiere;
33
34     lanterne--;
35     ++lanterne;
36
37     cout << lanterne << endl;
38
39     return 0;
40 }

```

Malheureusement, si Jacquot apprécie beaucoup les lanternes du jardin des lumières, il n'en est lui-même pas une (mais vous l'aimez bien quand même, il est gentil) et requiert donc votre aide pour compléter les bouts de code suivant. Il vous demande, tant que possible, de privilégier l'utilisation de **méthodes globales**.

Attention!

Pour les blocs où vous devez sélectionner plusieurs lignes de code à la suite (dans les implémentations), n'utilisez les mentions **[rien]** que pour les dernières lignes si elles sont inusitées. Utiliser **[rien]** au début ou au milieu (avec d'autres lignes que **[rien]** par la suite) vous fera perdre des points.

Par exemple, si le contenu de la fonction est `int i = 0; i++; return i;` mais que 5 lignes sont disponibles, alors je remplirais ligne 1 = `int i = 0;`, ligne 2 = `i++;`, ligne 3 = `return i;`, ligne 4 = `[rien]`, ligne 5 = `[rien]`

En dehors des blocs de lignes de code, **[rien]** reste un choix possible quand les autres ne s'appliquent pas. Par exemple, si ma méthode n'est pas censée être **const**, alors je sélectionne **[rien]** à l'endroit où **const** se trouverait (**const** et **[rien]** apparaîtront tous les deux dans la liste déroulante)

1. Additionner une Citrouille et une Lumiere pour obtenir une Lanterne (Lanterne lanterne = citrouille + lumiere)

- Définition: ()

```
operator+(
```

)		;
---	--	---

- Implémentation:

			operator+(
)		{
// Attention: s'il y a trop de lignes, utilisez les blocs			
// [rien] pour les dernières cases seulement!			

2. Utiliser l'opérateur ++ préfixe sur une Lanterne pour qu'elle éclaire plus fort (++lanterne)

- Définition: ()

```
operator++(  
    )  
;
```

- Implémentation:

[illegible]

3. Utiliser l'opérateur -- suffixe sur une Lanterne pour qu'elle éclaire moins fort (lanterne--)

- Définition: ()

```
operator--(
```

- Implémentation:

			operator--(
)		{
// Attention: s'il y a trop de lignes, utilisez les blocs // [rien] pour les dernières cases seulement!			

}

4. Permettre d'afficher une Lanterne via un flux de sortie et l'opérateur <<

- Définition: ()

operator<<(
) ;

- Implémentation:

operator<<(
) {
// Attention: s'il y a trop de lignes, utilisez les blocs
// [rien] pour les dernières cases seulement!
os << "Lanterne de couleur " <<

 << " de " <<
lumens" << endl;
}

Votre réponse est incorrecte.

La réponse correcte est :

Votre ami Jacquot, surnommé "*Le lent terne*" pour son humour sombre et désabusé, écrit un jeu vidéo permettant de fabriquer des lanternes à base de citrouilles. Jacquot vous montre les classes suivantes qu'il a commencé à écrire:

```
1 class Lanterne {  
2 public:  
3     Lanterne(string couleur, int lumens)  
4         : couleur_(couleur), lumens_(lumens) {};  
5 private:  
6     string couleur_;  
7     int lumens_;  
8 };  
9  
10 class Lumiere {  
11 public:  
12     Lumiere() : lumens_(42) {};  
13     int getLumens() const { return lumens_; }  
14     void setLumens(int lumens) { lumens_ = lumens; }  
15 private:  
16     int lumens_;  
17 };  
18  
19 class Citrouille {  
20 public:  
21     Citrouille() : couleur_("orange") {};  
22     string getCouleur() const { return couleur_; }  
23     void setCouleur(string couleur) { couleur_ =  
24 couleur; }  
25 private:  
26     string couleur_;  
27 };  
28  
29 int main () {  
30     Citrouille citrouille;  
31     Lumiere lumiere;  
32     Lanterne lanterne = citrouille + lumiere;  
33  
34     lanterne--;  
35     ++lanterne;  
36  
37     cout << lanterne << endl;  
38     return 0;  
39 }
```

Malheureusement, si Jacquot apprécie beaucoup les lanternes du jardin des lumières, il n'en est lui-même pas une (mais vous l'aimez bien quand même, il est gentil) et requiert donc votre aide pour compléter les bouts de code suivant. Il vous demande, tant que possible, de privilégier l'utilisation de **méthodes globales**.

Attention!

Pour les blocs où vous devez sélectionner plusieurs lignes de code à la suite (dans les implémentations), n'utilisez les mentions **[rien]** que pour les dernières lignes si elles sont inusitées. Utiliser **[rien]** au début ou au milieu (avec d'autres lignes que **[rien]** par la suite) vous fera perdre des points.

Par exemple, si le contenu de la fonction est "int i = 0; i++; return i;" mais que 5 lignes sont disponibles, alors je remplirais ligne 1 = int i = 0;, ligne 2 = i++;, ligne 3 = return i;, ligne 4 = [rien], ligne 5 = [rien]

En dehors des blocs de lignes de code, **[rien]** reste un choix possible quand les autres ne s'appliquent pas. Par exemple, si ma méthode n'est pas censée être **const**, alors je sélectionne **[rien]** à l'endroit où **const** se trouverait (**const** et **[rien]** apparaîtront tous les deux dans la liste déroulante)

1. Additionner une Citrouille et une Lumière pour obtenir une Lanterne (Lanterne lanterne = citrouille + lumière)

- Définition: ([Dans la partie "public" de la classe Citrouille])

```
[[rien]] [Lanterne] [[rien]]operator+([const Lumiere& obj]) [const];
```

- Implémentation:

```
[[rien]] [Lanterne] [Citrouille::]operator+([const Lumiere& obj]) [const] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    [Lanterne lanterne(couleur_, lumiere.getLumens());]
    [return lanterne;]
    [[rien]]
}
```

2. Utiliser l'opérateur ++ préfixe sur une Lanterne pour qu'elle éclaire plus fort (++lanterne)

- Définition: ([Dans la partie "public" de la classe Lanterne])

```
[[rien]] [Lanterne&] [[rien]]operator++([[rien]]) [[rien]];
```

- Implémentation:

```
[[rien]] [Lanterne&] [Lanterne::]operator++([[rien]]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    [lumens_++;]
    [return *this;]
    [[rien]]
    [[rien]]
    [[rien]]
}
```

3. Utiliser l'opérateur -- suffixe sur une Lanterne pour qu'elle éclaire moins fort (lanterne--)

- Définition: ([Dans la partie "public" de la classe Lanterne])

```
[[rien]] [Lanterne] [[rien]]operator--([int]) [[rien]];
```

- Implémentation:

```
[[rien]] [Lanterne] [Lanterne::]operator--([int]) [[rien]] {
    // Attention: s'il y a trop de lignes, utilisez les blocs
    // [rien] pour les dernières cases seulement!
    [Lanterne lanterne = *this;]
    [lumens_--;]
    [return lanterne;]
    [[rien]]
    [[rien]]
}
```

4. Permettre d'afficher une Lanterne via un flux de sortie et l'opérateur <<

- Définition: ([Dans la partie "public" de la classe Lanterne])


```
[friend] [ostream&] [[rien]]operator<<([ostream& os, const Lanterne& obj]) [[rien]];
```

- Implémentation:

```
[[rien]] [ostream&] [[rien]]operator<<([ostream& os, const Lanterne& obj]) [[rien]] {  
    // Attention: s'il y a trop de lignes, utilisez les blocs  
    // [rien] pour les dernières cases seulement!  
    os << "Lanterne de couleur " << [obj.couleur_]  
    << " de " << [obj.lumens_] << " lumens" << endl;  
    [return os;]  
    [[rien]]  
}
```

Question **19**

Non répondue

Noté sur 8,00

Soit le code suivant:

```

1  class Fee {
2  public:
3      Fee() { cout << "Fee()" << endl; }
4      Fee(string name) { cout << "Fee(" << name << ")" << endl; }
5      ~Fee() { cout << "~Fee()" << endl; }
6  };
7
8  class Lutin {
9  public:
10     Lutin() { cout << "Lutin()" << endl; }
11     Lutin(string name) { cout << "Lutin(" << name << ")" << endl; }
12     Lutin(string name, string fee) : fee_(fee) {
13         cout << "Lutin(" << name << ", " << fee << ")" << endl; }
14     ~Lutin() { cout << "~Lutin()" << endl; }
15 private:
16     Fee fee_;
17 };
18
19 class Licorne {
20 public:
21     Licorne() { cout << "Licorne()" << endl; }
22     Licorne(string name) { cout << "Licorne(" << name << ")" << endl; }
23     Licorne(string name, string lutin) : lutin_(lutin) {
24         cout << "Licorne(" << name << ", " << lutin << ")" << endl; }
25     Licorne(string name, string lutin, string fee) : lutin_(lutin, fee) {
26         cout << "Licorne(" << name << ", " << lutin << ", " << fee << ")" <<
27 endl; }
28     ~Licorne() { cout << "~Licorne()" << endl; }
29 private:
30     Lutin lutin_;
31 };
32
33 class PaysMagiqueDesLicornes {
34 public:
35     PaysMagiqueDesLicornes() {
36         cout << "PaysMagiqueDesLicornes()" << endl;
37         licornes_ = new Licorne("Alice Korn", "Luca Thym", "Fee Passa");
38     }
39     ~PaysMagiqueDesLicornes() { cout << "~PaysMagiqueDesLicornes()" << endl; }
40 private:
41     Lutin lutins_[2];
42     Licorne* licornes_;
43 };
44
45 int main () {
46     PaysMagiqueDesLicornes pays;
47
48     return 0;
49 }

```

Que va-t-il s'afficher sur la sortie standard ?

Si trop de lignes sont disponibles, sélectionnez "Rien" pour les dernières.

Par exemple, si le code devait afficher **Jambon()** **Pate()** **Moutarde()** et que 5 lignes sont disponibles, je sélectionnerais
 ligne 1 = **Jambon()**, ligne 2 = **Pate()**, ligne 3 = **Moutarde()**, ligne 4 = **Rien**, ligne 5 = **Rien**.

- Ligne 1: **Fee()**

- Ligne 2:

- Ligne 3:

- Ligne 4: **Lutin()**

- Ligne 5:

- Ligne 6:

- Ligne 7:

- Ligne 8:

- Ligne 9:

- Ligne 10: **~Lutin()**

- Ligne 11: **~Fee()**

- Ligne 12:

- Ligne 13:

• Ligne 14:

• Ligne 15:

• Ligne 16:

Votre réponse est incorrecte.

La réponse correcte est :

Soit le code suivant:

```

1 class Fee {
2 public:
3     Fee() { cout << "Fee()" << endl; }
4     Fee(string name) { cout << "Fee(" << name << ")" << endl; }
5     ~Fee() { cout << "~Fee()" << endl; }
6 };
7
8 class Lutin {
9 public:
10     Lutin() { cout << "Lutin()" << endl; }
11     Lutin(string name) { cout << "Lutin(" << name << ")" << endl; }
12     Lutin(string name, string fee) : fee_(fee) {
13         cout << "Lutin(" << name << ", " << fee << ")" << endl; }
14     ~Lutin() { cout << "~Lutin()" << endl; }
15 private:
16     Fee fee_;
17 };
18
19 class Licorne {
20 public:
21     Licorne() { cout << "Licorne()" << endl; }
22     Licorne(string name) { cout << "Licorne(" << name << ")" << endl; }
23     Licorne(string name, string lutin) : lutin_(lutin) {
24         cout << "Licorne(" << name << ", " << lutin << ")" << endl; }
25     Licorne(string name, string lutin, string fee) : lutin_(lutin, fee) {
26         cout << "Licorne(" << name << ", " << lutin << ", " << fee << ")" <<
27 endl; }
28     ~Licorne() { cout << "~Licorne()" << endl; }
29 private:
30     Lutin lutin_;
31 };
32
33 class PaysMagiqueDesLicornes {
34 public:
35     PaysMagiqueDesLicornes() {
36         cout << "PaysMagiqueDesLicornes()" << endl;
37         licornes_ = new Licorne("Alice Korn", "Luca Thym", "Fee Passa");
38     }
39     ~PaysMagiqueDesLicornes() { cout << "~PaysMagiqueDesLicornes()" << endl; }
40 private:
41     Lutin lutins_[2];
42     Licorne* licornes_;
43 };
44
45 int main () {
46     PaysMagiqueDesLicornes pays;
47
48     return 0;
49 }

```

Que va-t-il s'afficher sur la sortie standard ?

Si trop de lignes sont disponibles, sélectionnez "**Rien**" pour les dernières.

Par exemple, si le code devait afficher **Jambon()** **Pate()** **Moutarde()** et que 5 lignes sont disponibles, je sélectionnerais ligne 1 = **Jambon()**, ligne 2 = **Pate()**, ligne 3 = **Moutarde()**, ligne 4 = **Rien**, ligne 5 = **Rien**.

- Ligne 1: **Fee()**
- Ligne 2: **[Lutin()]**
- Ligne 3: **[Fee()]**
- Ligne 4: **Lutin()**
- Ligne 5: **[PaysMagiqueDesLicornes()]**
- Ligne 6: **[Fee(Fee Passa)]**
- Ligne 7: **[Lutin(Luca Thym, Fee Passa)]**
- Ligne 8: **[Licorne(Alice Korn, Luca Thym, Fee Passa)]**
- Ligne 9: **[~PaysMagiqueDesLicornes()]**
- Ligne 10: **~Lutin()**
- Ligne 11: **~Fee()**
- Ligne 12: **[~Lutin()]**

- Ligne 13: [-Fee()]
- Ligne 14: [Rien]
- Ligne 15: [Rien]
- Ligne 16: [Rien]

◀ [Recommandations pour les examens en ligne \(caché\)](#)

Aller à...

[Contrôle périodique - INF1010 - H2019 \(caché\)](#) ▶