Commencé le	vendredi 15 mars 2024, 18:31
État	Terminé
Terminé le	vendredi 15 mars 2024, 20:31
Temps mis	2 heures
Note	**15,47** sur 20,00 (**77,37**%)

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.
- En cas de réel problème technique, vous pouvez demander au surveillant.
- 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

Partiellement correct

Note de 1,50 sur 2,25

Cette question devrait être faite sans utiliser un compilateur.

Soit le programme suivant. Dans chaque case écrivez une seule valeur de manière à ce que l'exécution du programme corresponde à l'affichage. Si la valeur ne peut pas être déterminée, utilisez les valeurs suivantes:

- -1 si une donnée n'a pas d'influence sur le résultat de l'affichage et pourrait donc avoir n'importe quelle valeur;
- -2 si la valeur est indéterminée par "undefined behavior";
- 100 pour une valeur dans le programme qui influence l'affichage mais qui n'est pas contrainte par les valeurs données dans l'énoncé.

Réponse : (régime de pénalités : 0 %)

```
void f(int a, int& b, int c[])
{
    cout << b-- << " " << (c[1])++ << " " << ++a << endl;
    a += 10;
    b += 10;
}
int main() {
    int b = 35;
    int a = 26;
    int v[] = { 25, 14, 26 };
    f(a, b, v);
    cout << v[1] << " " << b << " " << a << endl;
}

Affichage:
35 14 27
```

14 43 26

Votre réponse	Devrait être	
35	35	✓
26	26	✓
14	14	✓
14	15	✗
43	44	✗
26	26	✓

Réponse bien enregistrée: 35, 26, 14, 14, 43, 26

Partiellement correct

Note pour cet envoi : 1,50/2,25.

Question 2

Partiellement correct

Note de 1,80 sur 2,25

Déterminer le type d'agrégation ou de composition des classes ci dessous pour le texte **nom_**. Une même réponse peut être utilisée plus d'une fois.

```
class Chose {
  public:
    Chose(const string& nom) : nom_(make_unique<string>(nom)) {}
  private:
    unique_ptr<string> nom_;
};
```

composition par pointeur ✓

```
class Chose {
  public:
     Chose(const string& nom) : nom_(nom) {}
  private:
     string nom_;
};
```

composition directe (par valeur) ✓

```
class Chose {
  public:
     Chose(string& nom) : nom_(nom) {}
  private:
     string& nom_;
};
```

agrégation par référence ✓

```
class Chose {
  public:
    Chose(const shared_ptr<string>& nom) : nom_(nom) {}
  private:
    shared_ptr<string> nom_;
};
```

composition par référence ✗

```
class Chose {
  public:
    Chose(string& nom) : nom_(&nom) {}
  private:
    string* nom_;
};
```

agrégation par pointeur ✓

Question 3

Partiellement correct

Note de 1,33 sur 2,50

Soit la déclaration de la variable: **unique_ptr<A> a**;

Pour chaque déclaration de fonction suivante (indépendantes les unes des autres), indiquez:

- comment appeler la fonction pour lui passer l'objet de type **A** pointé par **a** ;
- s'il y a prise de possession par la fonction de la mémoire pour l'objet de type **A** ;
- et si ce type de passage de paramètre est recommandé.

Pour l'appel, n'utilisez pas inutilement d'appel de fonctions ou méthodes, et n'ajoutez pas d'espacements. Par exemple, si on a une fonction void f(int x); et un int a; on peut faire f(move(a)) mais le move n'est pas nécessaire (est inutile) donc la bonne réponse serait f(a) . Notez que le f( ) est déjà là, il faut uniquement indiquer ce qu'il faut dans les parenthèses.

- void f(unique_ptr<A>& x); appel: f(a ✓)
  - prend toujours possession ✗
  - généralement non recommandé ✓
- void f(const unique_ptr<A>& x); appel: f(a ✓)
  - prend toujours possession ✗
  - recommandé si la fonction doit prendre possession de l'objet ✗
- void f(unique_ptr<A> x); appel: f(move(a) ✓)
  - peut prendre possession mais uniquement quand l'appel à f utilise move ✗
  - recommandé si la fonction doit prendre possession de l'objet ✓
- void f(A& x); appel: f(*a ✓) prend toujours possession ✗
  - recommandé si la fonction doit prendre possession de l'objet ✗
- void f(A* x); appel: f(a ✗) ne peut pas prendre possession ✓
  - recommandé si la fonction ne doit pas prendre possession de l'objet ✓

Question 4

Correct

Note de 2,75 sur 2,75

Cette question devrait être faite sans utiliser un compilateur.

Pour chaque case de cette question, indiquez le type qu'il faut mettre devant le nom de variable dans la déclaration pour que l'affectation soit correcte et qu'elle ne change pas le type de la valeur (ex. : bool x = 2; est une affectation correcte mais change le 2 d'un entier à un booléen, donc pas accepté comme réponse). Aucune des réponses n'est "const" ni une référence à quelque chose qui pouvait être affecté par copie.

Indiquez 'X' comme type si l'expression ne peut pas être correcte.

Vous n'obtenez aucun point pour une réponse auto.

Réponse : (régime de pénalités : 0 %)

```
// Soient les déclarations:
struct C { char** a; int b; double c; };
struct A { short a; C** b; C c; A* d; };
struct B { float a; A** b; string* c; };
A* c(int x);
A a;
B b;

// Indiquez les bons types devant les variables suivantes:

X        d = a.b->a;

C*       e = a.d->b[0];

C        f = a.d->b[0][0];

A        g = *(c(3));

A        h = **(b.b);

A*       i = c(5);

C        j = c(1)->d->c;

double*  k = &(a.b[0]->c);

double*  m = new double;

string   n = b.c[0];
```

Votre réponse	Devrait être	
X	X	✓
C*	C*	✓
C	C	✓
A	A	✓
A	A	✓
A*	A*	✓
C	C	✓

Votre réponse	Devrait être	
`double*`	`double*`	✓
`double*`	`double*`	✓
`string`	`string`	✓

Réponse bien enregistrée: X, C*, C, A, A, A*, C, double*, double*, string
Tous les tests ont été réussis ! ✓

Correct
Note pour cet envoi : 2,75/2,75.

Question 5

Partiellement correct

Note de 3,84 sur 4,25

Pour les questions de cette section, vous pouvez utiliser un compilateur, mais ce n'est pas nécessaire. Vous pouvez utiliser le bouton «Vérifier» sans pénalité.

Athletes

On veut écrire un programme permettant de gérer les **Athlete**s dans une **Competition.**

Les identifiants des classes, attributs et méthodes doivent être les mots en gras dans le texte, et les paramètres doivent être dans le même ordre que dans le texte, pour passer les tests.

Dans cette première partie, on veut écrire une classe de données (aucun invariant, les attributs sont publiques) **Athlete** qui permet de décrire un athlète par un **nom** et un **pays**, textuels, ainsi qu'un d'identification **id** (entier positif). Cette classe doit avoir:

- Un constructeur par défaut qui fait un athlète sans nom ni pays, et qui a l'id zéro.
- Un constructeur avec paramètre « input stream » (**istream**) qui initialise les attributs avec les valeurs lues du stream. Les valeurs sont dans l'ordre « nom pays id », sans espace dans le nom et le pays, un espace entre chaque champ, et une fin de ligne après l'id.
- Que la **copie** d'un athlète ait '_copie' ajouté à la fin de son nom, mais que le **move** d'un athlète déplace correctement l'athlète et n'ait pas cet ajout à son nom. Ex:
  - Athlete athleteB = athleteA: // athleteB aura un nom où _copie a été ajouté à la fin du nom de l'athleteA.
  - Athlete athleteC = move(athleteA); // athleteC aura le nom de l'athleteA, et l'athleteA n'a plus à exister, qui permet optimisation.
- Ce qu'il faut pour qu'on puisse faire la comparaison **athleteA < athleteB**, qui est vraie si l'**id** d'**athleteA** est plus petit.
- Ce qu'il faut pour qu'on puisse faire **cout << athlete << endl;** et que ça affiche « nom, pays, id », soit les valeurs des attributs séparées par une virgule et un espace.

Réponse : (régime de pénalités : 0 %)

```
class Athlete {
public:
    string nom;
    string pays;
    unsigned id;

    Athlete() : nom(""), pays(""), id(0){}
    Athlete(istream& cin) {
        cin >> nom >> pays >> id;
    }
    Athlete(const Athlete& autre) {
        if (this != &autre) {
            nom = autre.nom+"_copie";
            pays = autre.pays;
            id = autre.id;
        }
    }
     Athlete& operator=(Athlete&& autre) noexcept{
        if (this == &autre){
         nom = autre.nom;
         pays = autre.pays;
         id = autre.id;
         delete &autre;

        }
        return *this;
     }

     Athlete(Athlete&& autre) noexcept {
         *this = move(autre);
     }

     bool operator<(const Athlete& autre) {
         return id < autre.id;
     }

     friend ostream& operator<< (ostream& o, const Athlete& f);

```

```
39 };
40 ostream& operator<< (ostream& o, const Athlete& a) {
41     return o << a.nom << ", " << a.pays << ", " << a.id;
42 }
```

	Test	Résultat attendu	Résultat obtenu	
✓	_			✓
✓	`// Athlete existe et est constructible par défaut`	`true`	`true`	✓
✓	`// Le nom et pays de Athlete sont des textes` `// vides par défaut`	`truetruetruetrue`	`truetruetruetrue`	✓
✓	`// L'id de Athlete est un entier` `// zéro par défaut`	`truetrue`	`truetrue`	✓

✓	`for (int i : {0,1}) {`
	`  Athlete a(fichier);`
	`  afficherPourTest(a);`
	`}`
		`unNom,Canada,1337`
		`autreNom,France,9876`
			`unNom,Canada,1337`
			`autreNom,France,9876`
				✓

✓	`cout << (aA < aB) << (aB < aC)` `  << (aC < aD) << (aD < aC);`	`truefalsefalsefalse`	`truefalsefalsefalse`	✓
✓	`cout << aA << endl << aB << endl;`	`unNom, Canada, 1337` `autreNom, France, 9876`	`unNom, Canada, 1337` `autreNom, France, 9876`	✓

✓	`Athlete aCopy = a;`
	`Athlete bCopy = b;`
	`cout << a.nom << " " << aCopy.nom << " " << bCopy.nom;`
		`Alice Alice_copie Bob_copie`	`Alice Alice_copie Bob_copie`	✓

✗	`Athlete aCopy = a;`
	`Athlete aMove = move(a);`
	`Athlete bMove = move(b);`
	`cout << "'" << a.nom << "' " << aCopy.nom`
	`  << " " << aMove.nom << " " << bMove.nom;`
		`'' Alice_copie Alice Bob`	`'Alice' Alice_copie`	✗

Montrer les différences

Partiellement correct

Note pour cet envoi : 3,54/4,25.

Question 6

Partiellement correct

Note de 2,50 sur 4,25

En suite à la question précédente, on veut écrire la classe **Competition**, qui a en agrégation des **Athletes** . Cette classe doit permettre:

- La désallocation automatique du pointeur voyageurs_ en utilisant plutôt un unique_ptr (non testé par les tests automatiques).
- La copie **Competition copie = original;** où la copie doit avoir les mêmes athlètes que l'original, pas des copies des athlètes, mais on doit pouvoir enlever ou ajouter des athlètes de manière indépendante dans les deux compétitions après la copie.
- De faire une **sousCompetition** , une méthode qui prend en paramètre une lambda pour spécifier quels athlètes sont dans la sous-compétition et qui retourne une nouvelle **Competition** avec uniquement ces athlètes. La lambda est un prédicat unaire qui prend en paramètre un **Athlete** par référence constante et retourne un booléen vrai si l'athlète doit être dans la compétition.

Vous devez minimiser la duplication de code et la complexité des algorithmes n'est pas importante en autant que ça ne dépasse pas une complexité quadratique ($O(N^2)$). Préférez aussi span pour les boucles, si possible. (non testés par les tests automatiques)

Les tests utilisent une version fonctionnelle de **Athlete**, et non votre réponse précédente.
Ne changez pas les noms des méthodes/attributs et laissez la ligne qui contient "POUR_LES_TESTS" qui permet de faire les tests.

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
class Competition {
public:
    Competition(int capacite) :
        capacite_(capacite),
        nAthletes_(0)
    {
        athletes_ = make_unique<shared_ptr<Athlete>[]>(capacite_);
    }
    ~Competition() { }
    void ajouterAthlete(shared_ptr<Athlete> v) {
        athletes_[nAthletes_++] = v;
    }

    Competition& operator=(const Competition& autre) {
        capacite_ = autre.capacite_;
        nAthletes_ = autre.nAthletes_;
        unique_ptr<shared_ptr<Athlete>[]> nouvelle = make_unique<shared_ptr<Athlete>[]>(aut
        for (int i = 0; i < nAthletes_; i++) {
            nouvelle[i] = (autre.athletes_[i]);
        }
        athletes_ = move(nouvelle);
        return *this;
    }
    Competition(const Competition& autre) {
        *this = autre;
    }
    auto sousCompetition(Athlete& athlete) {
        return [&](auto sous) {return athlete;};
    }
private:
    int capacite_;
    unique_ptr<shared_ptr<Athlete>[]> athletes_;
    int nAthletes_;
    POUR_LES_TESTS
};
```

	Test	Résultat attendu	Résultat obtenu	
✓	_			✓

	Test	Résultat attendu	Résultat obtenu	

✓	`Competition copie = original;`
	`cout << (Tests::getAthletes(copie) !=`
	`Tests::getAthletes(original))`
	`<< (Tests::getNAthletes(copie) ==`
	`Tests::getNAthletes(original))`
	`<< (Tests::getAthletes(copie)[0]->id ==`
	`Tests::getAthletes(original)[0]->id)`
	`<< (Tests::getAthletes(copie)[1]->id ==`
	`Tests::getAthletes(original)[1]->id);`
		`truetruetruetrue`	`truetruetruetrue`	✓

✓	`Competition copie = original;`
	`cout << (Tests::getAthletes(copie) !=`
	`Tests::getAthletes(original))`
	`<< (Tests::getNAthletes(copie) ==`
	`Tests::getNAthletes(original))`
	`<< (Tests::getAthletes(copie)[0] ==`
	`Tests::getAthletes(original)[0])`
	`<< (Tests::getAthletes(copie)[1] ==`
	`Tests::getAthletes(original)[1]);`
		`truetruetruetrue`	`truetruetruetrue`	✓

✗	`Competition sous = original.sousCompetition(estFrance);`
	`afficherPourTests(sous);`
		`2`
		`b, France, 234`
		`d, France, 456`
			`tests.cpp: In function 'int main(int, char**)':`
			`tests.cpp:146:54: error: no matching function for call to 'Competition::sousCompetition(main(int, char**)::<lambda(auto:2&)>&)'`
			`__tester__.cpp:27:10: note: candidate: auto Competition::sousCompetition(Athlete&)`
			`auto sousCompetition(Athlete& athlete) {`
			`^~~~~~~~~~~~~~`
			`__tester__.cpp:27:10: note: no known conversion for argument 1 from 'main(int, char**)::<lambda(auto:2&)>' to 'Athlete&'`
				✗

Montrer les différences

Partiellement correct

Note pour cet envoi : 2,13/4,25.

Question 7

Correct

Note de 1,75 sur 1,75

Nous voulons une fonction **estPareil** qui fonctionne sur différents types de données et qui retourne vrai uniquement si les deux données sont de même type et qu'elles comparent comme étant égales avec l'opérateur ==. Pour vérifier que deux types sont identiques, il existe le trait de type **is_same_v<A, B>** , mais ce n'est pas l'unique manière de répondre à cette question.

On veut donc, par exemple, que

- **estPareil(4, 4)** soit **vrai** car c'est le même type et même valeur
- **estPareil(4, 3)** soit **faux** car 4 n'est pas == 3
- **estPareil(4, 4.0)** soit **faux** car 4 est un **int** et 4.0 est un **double**, les types sont donc différents.

Le but est d'avoir une fonction générique, et non de surcharger pour chaque cas des tests.

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  template <typename T, typename U>
2  bool estPareil(T a, U b)
3  {
4      return (a == b) && is_same_v<T,U>;
5  }
```

	Test	Résultat attendu	Résultat obtenu	
✓	`cout << estPareil(3, 4) << estPareil(4, 4);`	`falsetrue`	`falsetrue`	✓
✓	`cout << estPareil(4.5, 4.5) << estPareil(4.5, 4.0) << estPareil(4, 4.0);`	`truefalsefalse`	`truefalsefalse`	✓
✓	`cout << estPareil(4U, 4U) << estPareil(4, 4U);`	`truefalse`	`truefalse`	✓
✓	`cout << estPareil(4L, 4L) << estPareil(4L, 4);`	`truefalse`	`truefalse`	✓

Tous les tests ont été réussis ! ✓

Correct

Note pour cet envoi : 1,75/1,75.