Description

Marquer la question

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**

Réponse enregistrée

Noté sur 1,00

Marquer la question

```
unordered_map<int,int> v;
// ... des valeurs dans v
int n = v.size();
for (int i : range(42))
    v[f(i)] = i;
```

Est $O($ 1 $)$

Question **2**

Réponse enregistrée

Noté sur 1,00

Marquer la question

```
int somme = 0;
for (int i : range(n))
    for (int j : range(i))
        somme += f(i,j);
```

Est $O($ n^2 $)$

Question **3**

Réponse enregistrée

Noté sur 1,00

Marquer la question

```
vector<int> v;
int cnt = 0;
for (auto&& e1 : v) {
    cnt += count_if(
        v.begin(),
        v.end(),
        [&](auto e) { return f(e1, e); }
    );
    cout << cnt;
}
```

Est $O($ n^2 $)$

On désir 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:
   A() { }
};

class B {
public:
   B() { }
};

class C {
public:
   C() { }
};

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

Soit le programme suivant:

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

On veut savoir l'ordre de construction lorsqu'on exécute ce programme. Indice: il y a 5 objets/sous-objets construits.

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

Réponse : (régime de pénalités : 0 %)

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

D B A A C

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

B A A D C

Vous décidez d'implémenter un planificateur de tâches en utilisant les nouvelles connaissances que vous avez acquises sur la STL.

Les définitions de classes suivantes (se trouvant dans leur .h respectif) représentent la base de votre planificateur, mais nous allons en implémenter seulement une petite partie:

```
class Tache {
public:
	Tache(enums::Priorite priorite, const std::string& nom, const std::tm& echeance = tm());

	bool operator==(const Tache& tache);

	bool operator<(const enums::Priorite& priorite);
	friend bool operator<(const enums::Priorite& priorite, const Tache& tache);
	bool operator<(const tm& echeance);
	friend bool operator<(const tm& echeance, const Tache& tache);

	std::string getNom() const;
	tm  getEcheance() const;
	enums::Priorite getPriorite() const;

	friend std::ostream& operator<<(std::ostream& out, const Tache& tache);

private:
	std::string nom_;
	enums::Priorite priorite_;
	tm  echeance_;
};
```

```
class Projet {
public:
	Projet(std::string nom);

	std::string getNom() const;

	void ajouterTache(const Tache& tache);
	void supprimerTache(const Tache& tache);

	void trierTaches(std::function<bool(const Tache&, const Tache&)> f);
	std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getTaches(tm& echeance);
	std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getTaches(enums::Priorite prio);

	friend std::ostream& operator<<(std::ostream& out, const Projet& projet);

private:
	std::string nom_;
	std::vector<Tache> taches_;

	template<typename Comparateur, typename T>
	std::pair<std::vector<Tache>::iterator, std::vector<Tache>::iterator> getRange(T element);
};
```

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

	template<typename Comparateur, typename T>
	std::multimap<Tache, std::string, Comparateur> getTaches(T element);

	std::multimap<Tache, std::string, ComparateurEcheance> getTachesParPriorite(enums::Priorite prio);
	std::multimap<Tache, std::string, ComparateurPrioriteDecroissant> getTachesParEcheance(tm& echeance);

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

Les priorités sont définies sous forme d'énumération:

```
namespace enums {
	enum Priorite {
		BASSE,
		MOYENNE,
		HAUTE,
	};
}
```

ComparateurEcheance, ComparateurPrioriteCroissant et ComparateurPrioriteDecroissant sont des foncteurs qui permettent la comparaison entre deux tâches de la manière énoncée dans leur nom.

On vous demande de faire l'implémentation des méthodes **ajouterTache** et **supprimerTache** de la classe **Projet**. La tâche est ajoutée au vecteur seulement si elle n'existe pas déjà dans celui-ci.

Par exemple:

Test	Résultat

`projet.ajouterTache(tache1);`
`projet.ajouterTache(tache1);`
`cout << projet.getNbTaches() << endl;`
	`1`

`projet.ajouterTache(tache1);`
`projet.ajouterTache(tache2);`
`cout << projet.getNbTaches() << endl;`
	`2`

`projet.ajouterTache(tache1);`
`projet.ajouterTache(tache2);`
`projet.supprimerTache(tache1);`
`projet.supprimerTache(tache2);`
`cout << projet.getNbTaches() << endl;`
	`0`

Réponse : (régime de pénalités : 10, 20, ... %)

Réinitialiser la réponse

```
void Projet::ajouterTache(const Tache& tache)
{
    if (auto it = find(taches_.begin(), taches_.end(), tache); it == taches_.end())
        taches_.push_back(tache);
}
/*If std::find does not find a match for the tache object within the range,
it returns an iterator to the end of the vector taches_. Therefore, if it equals taches_.end(),
it means that tache is not already present in the vector.
*/
void Projet::supprimerTache(const Tache& tache)
{
    auto it = remove(taches_.begin(), taches_.end(), tache);
    taches_.erase(it, taches_.end());
}
/*The line auto it = remove(taches_.begin(), taches_.end(), tache); uses the remove()
algorithm from the <algorithm> header to remove all occurrences of the tache object from the taches_ vector.
The remove() algorithm takes three arguments: the beginning and end iterators of
the range to operate on, and the value to remove. It returns an iterator pointing to the element past the last element not removed.
In the code snippet you provided, the remove() algorithm returns an iterator it pointing to
the element past the last element not removed, which is the first element in the vector that was not equal to tache.
The auto keyword allows the type of it to be automatically deduced by the compiler based on the return type of the remove() function.
In this case, it will be of type std::vector<Tache>::iterator, which is the iterator type for the taches_ vector.
The it iterator is then used as the starting point to erase all the elements that were removed by remove(), by calling the erase()
member function of the taches_ vector. The second argument to erase() is the end iterator, which is the end() iterator of the taches_ vector.
This removes all the elements from the it iterator to the end of the vector.
*/
// Ou, puisque ce n'est pas dit s'il faut enlever plusieurs tâches identique (et n'est pas dans les tests):
void Projet::supprimerTache(const Tache& tache) {
    if (auto it = find(taches_.begin(), taches_.end(), tache); it != taches_.end())
        taches_.erase(it);
}
```

On vous demande de faire l'implémentation de la méthode **trierTaches** de la classe **Projet**. La méthode **trierTaches** permet de trier les tâches du projet en utilisant la fonction qui lui est passée en paramètre.

Par exemple:

Test	Résultat

`projet.trierTaches(ComparateurEcheance());`
`cout << projet << endl;`
	`Nom: INF1010`
	`Taches:`
	`Nom: Composition & agregation`
	`Priorite: 0`
	`Echeance: 2020/02/12`
	
	`Nom: Revision Intra`
	`Priorite: 2`
	`Echeance: 2020/03/14`
	
	`Nom: Pointeurs intelligents`
	`Priorite: 1`
	`Echeance: 2020/04/20`
	
	`Nom: Nouvelle tache`
	`Priorite: 2`
	`Echeance: 2020/04/20`

Réponse : (régime de pénalités : 10, 20, ... %)

Réinitialiser la réponse

```
1  void Projet::trierTaches(function<bool(const Tache&, const Tache&)> f)
2  {
3      // Attention que le compilateur sur Moodle n'a pas les ranges:: de C++20.
4      sort(taches_.begin(), taches_.end(), f);
5  }
6  /*n this implementation, the sort() function is called with the beginning and end iterators of the taches_ vector,
7  along with the provided comparison function f.
8  The comparison function f takes two Tache objects as input and returns a bool.
9  The function returns true if the first argument is less than the second argument according to the sorting order,
10 and false otherwise. The sorting order is determined by the implementation of f.
11 */
```

Soit le code suivant:

```
class Cmp {
public:
        template<typename T>
        bool operator()(const T& gauche, const T& droite) const {
                return gauche > droite;
        }
};

template<typename Ta, typename Tb>
class Noeud {
public:
        Noeud(Ta a, Tb b) : a_(a), b_(b) {}

        bool operator<(const Noeud& n) const {
                return (a_ < n.a_) || (a_ == n.a_ && b_ < n.b_); }

        bool operator>(const Noeud& n) const {
                return (a_ > n.a_) || (a_ == n.a_ && b_ > n.b_); }

private:
        Ta a_;
        Tb b_;
};

int main() {
        // Le numéro en commentaire à la fin de chaque ligne indique
        // l'ordre d'entrée des éléments dans la map
        map<Noeud<string, string>, int> johiver = {
                {{"Etats-Unis", "Lake Placid"}, 1980},          // 1
                {{"Yougoslavie", "Sarajevo"}, 1984},            // 2
                {{"Canada", "Calgary"}, 1988},                  // 3
                {{"France", "Albertville"}, 1992},              // 4
                {{"Norvege", "Lillehammer"}, 1994},             // 5
                {{"Japon", "Nagano"}, 1998},                    // 6
                {{"Etats-Unis", "Salt Lake City"}, 2002},       // 7
                {{"Italie", "Turin"}, 2006},                    // 8
                {{"Canada", "Vancouver"}, 2010}                 // 9
        };

        map<Noeud<string, string>, int, Cmp> johiver2;
        copy(johiver.begin(), johiver.end(), inserter(johiver2, johiver2.begin()));

        return 0;
}
```

Indiquez, en utilisant les numéros (1 à 9) des entrées dans le conteneur **johiver**, dans quel ordre sont stockés les éléments dans **johiver**.

Dans la réponse, indiquez la série de chiffres sans espaces entre eux.

Réponse : 391748652

Indiquez, en utilisant les numéros (1 à 9) des entrées dans la map **johiver**, dans quel ordre sont stockés les éléments dans **johiver2**.

Dans la réponse, indiquez la série de chiffres sans espaces entre eux.

Réponse : 256847193

Dans le programme principal précédent, quelle version de l'opérateur générique () de la classe Cmp est appelée (en d'autres termes, à quel type correspond le T):

Veuillez choisir une réponse.

- a. pair<Noeud<string, string>, int>
- b. Noeud<string, string>
- c. string
- d. int

Effacer mon choix

Cochez les affirmations qui sont justes.

Veuillez choisir au moins une réponse.

- [x] a. Il est possible de lever une exception dans un bloc de capture d'exception ("catch").
- [x] b. Il est possible d'utiliser plusieurs blocs "catch" à la suite d'un bloc "try".
- [ ] c. Après la capture et la gestion d'une exception, le programme reprend son exécution à l'instruction ayant causé l'exception.
- [ ] d. Lever une exception dans un destructeur sans la gérer permet d'éviter les fuites de mémoire.
- [ ] e. Lors de la levée d'une exception, toute la mémoire dynamiquement allouée par la fonction est automatiquement libérée.

L'utilisation d'un espace de noms (namespace) permet de réduire la portée (scope) de ce qu'elle contient.

Il est possible de définir un seul symbole(s) ayant le même nom dans un même espace de nom.

Cochez les affirmations qui sont justes.

Veuillez choisir au moins une réponse.

- [x] a. L'utilisation d'attributs statiques de classe permet de les partager entre toutes les instances de la classe.
- [x] b. L'accès à une méthode statique peut se faire à l'aide de l'opérateur de résolution de portée ("::").
- [ ] c. Utiliser le mot clef static permet de réduire la portée (scope) d'une méthode.
- [x] d. Une méthode statique peut être appelée sans instancier d'objet de la classe dont elle fait partie.
- [ ] e. Un attribut statique de classe est forcément public.
- [ ] f. Un attribut non statique de classe peut être directement accédé dans une méthode statique de la même classe, comme pour toute autre méthode de cette classe.
- [x] g. Une variable globale statique est accessible dans d'autres fichiers que celui dans lequel elle est déclarée.

Lors de la définition d'une nouvelle exception par l'utilisateur, celle-ci doit absolument hériter de std::exception.

Veuillez choisir une réponse.

- ○ Vrai
- ◉ Faux

Soit le code suivant (des commentaires sont présents à la place de certaines lignes de code pour alléger la lecture) :

```
class ErreurCovid19 : public runtime_error {
public:
    using runtime_error::runtime_error;
};

class ErreurQuarantaine : public ErreurCovid19 {
public:
    using ErreurCovid19::ErreurCovid19;
};

class ErreurDistanciationSociale : public ErreurCovid19 {
public:
    using ErreurCovid19::ErreurCovid19;
};

class Citoyen {
public:
    void faireDesTartelettesPortugaises() {
        if (not vérifierGardeManger())
            faireDesCourses();

        // Faire la recette...

        cout << "Miam!" << "\n";
    }

    void faireDesCourses() {
        if (estSymptomatique())
            throw ErreurQuarantaine("Peut pas faire l'épicerie :(");

        bool fini = false;
        while (not fini) {
            if (mesurerDistancePlusProche() < 2.0)
                throw ErreurDistanciationSociale("Stranger danger!");

            // Se promener et acheter ce qu'il faut...
            // Mettre à jour 'fini', quand on a ce qu'il faut
        }

        // Après appel sans erreur de la méthode, vérifierGardeManger() retourne true.
    }

    bool   vérifierGardeManger() const;
    double mesurerDistancePlusProche() const;
    bool   estSymptomatique() const;
    void   attendre14Jours();
    void   faireAttentionAu2m();
};


int main() {
    Citoyen drArruda;
    // Faire de la distanciation sociale...

    try {
        drArruda.faireDesTartelettesPortugaises();
    } catch (ErreurDistanciationSociale&) {
        drArruda.faireAttentionAu2m();
    } catch (ErreurQuarantaine&) {
        drArruda.attendre14Jours();
    } catch (ErreurCovid19& e) {
        cout << "Erreur de COVID-19 : " << e.what() << endl;
    } catch (exception& e) {
        cout << "Erreur générique : " << e.what() << endl;
    } catch (...) {
        cout << "Erreur inconnue" << endl;
    }
}

```

Quel est le comportement du code ci-dessus si le citoyen n'a pas tous les ingrédients nécessaires (la méthode `vérifierGardeManger()` retourne `false`) et qu'il est malade (si la méthode `estSymptomatique()` retourne `true`)? On suppose que les bouts de code remplacés par des commentaires et les méthodes non implémentées ne lancent pas d'exception.

Veuillez choisir une réponse.

- a. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`.. Étant donné qu'on a mis un `catch(...)`, c'est celui-ci qui attrape toutes les erreurs. On a donc un message d'erreur inconnue qui s'affiche et le programme se termine correctement (atteint la fin du `main()`).
- b. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le message `Erreur de COVID-19` est affiché (troisième `catch`).
- c. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et à travers le `main()`. Le programme plante à cause d'une exception non gérée.
- d. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite attendre 14 jours (deuxième `catch`) et le programme se termine correctement (atteint la fin du `main()`).

Effacer mon choix

Quel est le comportement du code ci-dessus si le citoyen n'a pas tous les ingrédients et que durant ses courses il est mal distancé des autres citoyens (`mesurerDistancePlusProche()` retourne une valeur < 2)?

Veuillez choisir une réponse.

- a. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite garder ses distances et le programme se termine correctement (atteint la fin du `main()`).
- b. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`. Le citoyen va ensuite attendre 14 jours et le programme se termine correctement (atteint la fin du `main()`).
- c. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()`.. Étant donné qu'on a mis un `catch(...)`, c'est celui-ci qui attrape toutes les erreurs. On a donc un message d'erreur inconnue qui s'affiche et le programme se termine correctement (atteint la fin du `main()`).
- d. Une exception est lancée dans `Citoyen::faireDesCourses()`, se propage à travers `Citoyen::faireDesTartelettesPortugaises()` et est attrapée dans le `main()` par le troisième catch. Une erreur COVID-19 est affichée et le programme se termine correctement (atteint la fin du `main()`).

On considère l'implémentation suivante du `main()` en conservant le reste du code inchangé :

```
int main() {
    Citoyen drArruda;
    // Faire de la distanciation sociale...

    while (true) {
        try {
            drArruda.faireDesTartelettesPortugaises();
            break;
        } catch (ErreurDistanciationSociale&) {
            drArruda.faireAttentionAu2m();
        } catch (ErreurQuarantaine&) {
            drArruda.attendre14Jours();
        } catch (ErreurCovid19& e) {
            cout << "Erreur de COVID-19 : " << e.what() << endl;
        } catch (exception& e) {
            cout << "Erreur générique : " << e.what() << endl;
        } catch (...) {
            cout << "Erreur inconnue" << endl;
        }
    }
}
```

Supposons que la méthode `estSymptomatique()` retourne vrai une première fois, puis retourne faux par la suite, et que la méthode `vérifierGardeManger()` retourne faux au départ, puis retourne vrai après un appel complet à `faireDesCourses()`. Qu'arrive-t-il?

Veuillez choisir une réponse.

- a. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le deuxième `catch`. Le `try-catch` brise tout de suite la boucle et le programme se termine normalement.
- b. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le deuxième `catch` (donc on attend 14 jours). La boucle se répète une deuxième fois, la méthode complète son appel et affiche «Miam!», puis la boucle se brise. Le programme se termine ensuite normalement.
- c. Au premier appel de `drArruda.faireDesTartelettesPortugaises()`, une exception est levée et attrapée par le `catch(...)` qui attrape toutes les erreurs. Toutefois, puisque le `try-catch` est dans un `while`, on obtiendra nécessairement une boucle infinie dans la majorité des cas.