INF1010
Programmation orientée-objet

Travail pratique 5
Librairie standard, expressions lambdas et foncteurs

Département de génie informatique et logiciel

Polytechnique Montréal
31 mars 2020

POLYTECHNIQUE MONTRÉAL

Objectifs

Permettre à l'étudiant de se familiariser avec les conteneurs, les itérateurs et les algorithmes de la libraire standard, l'utilisation de lambdas et de foncteurs, et enfin de revoir en profondeur les notions de possession et de non-possession pour les données gérées par pointeurs intelligents.

Remise du travail

- Date : **17 avril 2020 à 23h55**
- **Une note de 0 sera attribuée aux équipes qui remettent leur travail en retard.**
- Remettre **tous** les fichiers `.cpp` et `.h` directement sous une archive **.zip** (sans dossier à l'intérieur et sans autres fichiers). **Tout archive autre que zip ou remise ne respectant pas ces consignes sera refusée.**
- Nom de l'archive zip : **`matricule1_matricule2_groupe.zip`** où matricule1 < matricule2. Exemple : 1234566_1234567_1.zip

Références

- Notes de cours sur Moodle
- https://en.cppreference.com/w/cpp
- https://en.cppreference.com/w/cpp/container
- https://en.cppreference.com/w/cpp/algorithm
- https://en.cppreference.com/w/cpp/iterator
- https://en.cppreference.com/w/cpp/language/lambda
- https://en.cppreference.com/w/cpp/utility/pair

Directives

- Les travaux s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.
- Les entêtes de fichiers sont obligatoires.
- Les fonctions doivent être documentées.
- **Le guide de codage sur Moodle doit être suivi.**

Conseils

- Lisez les conseils, l'aperçu et les spécifications avant de commencer!
- Ayez lu vos notes de cours!

Tout cas de plagiat sera automatiquement reporté au Comité d'examen de fraude (CEF)

Si, après recherche, vous êtes confus par rapport à un élément du TP, vous pouvez venir poser vos questions sur Slack dans le channel `#td5`. Vous devez utiliser votre adresse @polymtl.ca.

Lien pour faire son compte : https://join.slack.com/t/inf1010-h20/signup

Lien du workspace : https://inf1010-h20.slack.com

Mise en contexte (lecture frivole et optionnelle)

Après des mois de dur labeur non payé de développeurs sous-contractés, l'entreprise CinéPoly remporte un succès mitigé. Malgré une plateforme média à la fine pointe de la technologie, fruit des efforts combinés d'une populace énergisée, CinéPoly ne rentre pas dans ses marges de profit et risque la faillite. Ne voulant pas remettre en question la qualité du produit technologique au centre de son modèle d'affaires, l'entreprise considère maintenant s'emparer de nouvelles sources de revenu en se tournant vers des marchés moins honnêtes : la vente des données personnelles. Sachant que les programmeurs en apprentissage de Polytechnique sont une source inépuisable de main-d'œuvre gratuite, CinéPoly fait de nouveau recours à vous et vos talents exceptionnels de programmeur·se. Vous devrez donc écrire un nouveau programme pouvant extraire des journaux (logs) d'utilisation les tendances relatives aux préférences des utilisateurs afin de vendre ces informations aux dieux des réseaux informatiques sans scrupule aucun.

Conseils

- **Compilateur** : Vous êtes encouragés (mais aucunement obligés) à utiliser GCC (ou Clang) pour ce TP. Pour ceux travaillant sur un OS basé sur Unix (Linux et Mac), vous pouvez utiliser le Makefile comme d'habitude. Pour ceux sur Windows, si vous le souhaitez, vous pouvez utiliser GCC en installant MinGW-W64 et en utilisant le Makefile à partir de **Git Bash** (et non cmd.exe) et en substituant la commande `make` pour `mingw32-make`. Les instructions pour l'installation de MinGW sur Windows se retrouvent ici (n'oubliez pas d'ajouter `mingw/bin` à votre path!) : https://code.visualstudio.com/docs/cpp/config-mingw. Le lien explique aussi comment configurer le `launch.json` pour l'intégration du débogueur (GDB) dans VS Code. Enfin, vous pouvez utiliser l'extension VS Code « C/C++ » pour activer l'autocomplétion et IntelliSense.
- **Utilisation du Makefile** : Placez tous les fichiers `*.cpp` dans `src`, les `*.h` dans `include` et laissez le Makefile à la racine du projet. Utilisez la commande `make` pour compiler, et `make run` pour exécuter le programme. N'oubliez pas de copier les fichiers `*.txt` dans le répertoire d'exécution (`bin/[linux|macos|windows32|windows64]/debug`) pour que le programme les trouve correctement.
- **Conteneurs associatifs** : Ce TP utilise les conteneurs associatifs `std::unordered_map` et `std::unordered_set` plutôt que `std::map` et `std::set`, respectivement. Les versions non ordonnées sont hashées et donc beaucoup plus performantes et devraient être favorisées. Puisque **leur interface reste sensiblement la même**, les notes de cours portant sur `std::map` et `std::set` peuvent quand même être consultées pour comprendre les versions non ordonnées.
- **Méthodes des conteneurs** : Une grande partie du TP consiste à trouver la façon la plus élégante de satisfaire le comportement demandé, ce qui importe d'utiliser le moins de lignes de code pour utiliser la librairie standard à son plein potentiel. En programmant,

ayez toujours la documentation de cppreference sur les conteneurs et algorithmes avec vous pour trouver la meilleure façon de résoudre le problème! N'oubliez pas de regarder si certaines méthodes des conteneurs retournent des valeurs pertinentes pour simplifier encore plus votre code!

- **Difficulté** : Si vous avez des difficultés au cours du TP, rappelez-vous que de nombreux problèmes demandés sont assez couramment rencontrés en C++. Consulter la documentation sur cppreference, les notes de cours, et les forums sur Google peut vous donner de bonnes pistes de résolution! Prenez également le temps de vous familiariser avec la hiérarchie du projet avant de commencer.
- **Tests** : Le TP a été conçu pour vous permettre de compiler votre code graduellement, afin que vous soyez en mesure de voir et corriger vos erreurs au fur et à mesure. Il y a 4 macros dans le fichier `Tests.h`, initialement mis à la valeur de false. Aussitôt que vous avez terminé d'implémenter une classe, vous pouvez activer ses tests, et ceux-ci fonctionneront même si vous n'avez pas encore implémenté les fonctionnalités plus loin dans le TP. Puisque certaines méthodes vers la fin du TP dépendent du bon fonctionnement des classes implémentées au début du TP, **vous êtes très fortement encouragés à suivre l'ordre suggéré par l'énoncé pour pouvoir profiter des tests.**
- **Débogage** : Si vos tests ne passent pas, allez jeter un coup d'œil à leur code, et pensez à utiliser votre débogueur et des breakpoints dans les tests! Les tests sont très détaillés et souvent unitaires; vous trouverez sans doute le problème en inspectant les variables.
- **Documentation** : Ayez les pages https://en.cppreference.com/w/cpp/container et https://en.cppreference.com/w/cpp/algorithm ouvertes avec vous lorsque vous travaillez, elles vous seront indispensables!

Aperçu de la hiérarchie du projet

- **`GestionnaireUtilisateurs`** : Gère la possession exclusive des `Utilisateurs` chargés du fichier `utilisateurs.txt`. Les utilisateurs sont sauvegardés dans une map pour les retrouver à partir de leur ID, soit leur courriel.
    - **`Utilisateur`** : Struct POD déjà fournie contenant les informations pour un utilisateur. Tous les champs sont donc publics (et n'ont pas de suffixe _). L'opérateur d'insertion de stream (`operator<<`) est déjà fourni pour que vous puissiez afficher un `Utilisateur` facilement.
- **`GestionnaireFilms`** : Gère la possession exclusive des `Films` chargés du fichier `films.txt`. Les films sont sauvegardés dans un vecteur de pointeurs uniques, et la classe possède trois maps agissant comme **filtres** pour rechercher rapidement des films en fonction d'un critère. Les trois maps **contiennent donc les films par agrégation, ce qui explique pourquoi ils ont un pointeur brut vers les films.** (Les pointeur bruts sont vers

`const` puisqu'il n'y a aucune raison de vouloir permettre une modification d'un film à travers un filtre.)

- **Filtres :**
  - **`filtreNomFilms_`** : Permet de retrouver le pointeur vers un film à partir de son nom.
  - **`filtreGenreFilms_`** : Permet d'obtenir la liste de films (un vecteur) ayant un certain genre.
  - **`filtrePaysFilms_`** : Permet d'obtenir la liste de films (un vecteur) ayant un certain pays.
- **`Film`** : Struct POD déjà fournie contenant les informations pour un film. Même principe que pour `Utilisateur`.

- **`AnalyseurLogs`** : Gère la possession exclusive des `LigneLogs` chargées du fichier `logs.txt` et permet d'en extraire des statistiques. Les lignes de log sont conservées dans un vecteur ordonné, c'est-à-dire que chaque nouvelle insertion se fait en respectant l'ordre préexistant. Dans ce TP, les lignes de log sont ordonnées selon leur timestamp.
  - **`LigneLog`** : Struct POD déjà fournie contenant les informations tirées d'une ligne des logs. Chacune contient un timestamp et des références non possédantes vers un `Utilisateur` et un `Film`, ce qui explique pourquoi des pointeurs bruts sont utilisés.
- **Enums fournis :**
  - **`Pays`** : Représente un pays. La fonction `getPaysString` permet de retrouver la représentation du pays comme string.
  - **`Film::Genre`** : Représente un genre de film. La fonction `getGenreString` permet de retrouver la représentation du genre comme string.
- **`Foncteurs.h`** : C'est dans ce fichier que vous écrirez trois foncteurs dont vous aurez besoin.
- **`RawPointerBackInserter.h`** : Fichier déjà écrit pour vous. Il vous permet d'imiter le comportement d'un `std::back_inserter` tout en insérant un pointeur brut obtenu à partir d'un pointeur unique. Vous n'avez pas à comprendre son implémentation.
- **`Tests.h`** et **`Tests.cpp`** : Fichiers déjà écrits pour vous. Contiennent les tests automatisés. Pour pouvoir tester au fur et à mesure sans qu'il y ait d'erreur de compilation pour des fonctions pas encore implémentées, ce TP implémente des tests graduels. Il vous de changer les 4 defines à true un à un lorsque vous êtes prêts à tester une composante.
- **`main.cpp`** : La fonction main vous est laissée si vous voulez expérimenter librement avec les classes que vous avez implémentées, tant qu'elle appelle d'abord les tests. Il vous est également possible d'obtenir un point bonus (voir la fin de l'énoncé).

Spécifications générales

- Toutes les classes, structs, enums, fonctions et entêtes de fichiers doivent être documentées en utilisant la convention de Doxygen. Suivez l'exemple de la documentation déjà présente dans le TP.
- Ne modifiez pas la signature des méthodes. Celles-ci sont déjà const-correct, et vous ne devez en aucun cas enlever un const.
- Toutes les méthodes doivent être définies dans le fichier d'implémentation (`.cpp`) **dans le même ordre** que leur déclaration dans le fichier d'en-tête (`.h`). Le non-respect de cette règle entraînera une pénalité au niveau du style.
- Tout warning à la compilation sera pénalisé. Si vous utilisez Visual Studio, vous devez activer l'option **/W4**.
- Utilisez le plus possible la liste d'initialisation des constructeurs.
- N'utilisez aucun `new` ou `delete`.
- N'utilisez aucune boucle `for`, `while` ou `do while` sauf indication contraire.
- L'utilisation la plus efficace possible de la libraire standard est exigée pour toutes les méthodes. **Assurez-vous donc de bien respecter les contraintes quant au nombre maximal de lignes de code lorsque demandé**. Notez que le nombre de lignes de code indiqué ne compte pas les lignes vides, les commentaires et les accolades seules sur leur ligne. Si un appel de fonction est trop long et que ses arguments se retrouvent sur des lignes individuelles, ceci compte toujours comme une seule ligne.
- Suivez le guide de codage sur Moodle.
- Modifications/ajouts au guide de codage à respecter :
    - Vous pouvez utiliser le style d'accolades que vous désirez, **tant que vous êtes uniformes**. Sachez cependant qu'il est généralement attendu d'un·e développeur·se de suivre la convention établie par le projet, ou la convention établie par la langue (si elle existe) en créant un nouveau projet. Dans le cas du TP fourni, les accolades ouvrantes sont sur leur propre ligne (style Allman).
    - Mettez un espace avant la parenthèse ouvrante des énoncés de contrôle comme if, for, while et switch, mais pas avant les parenthèses d'un appel de fonction. Ne mettez jamais d'espace tout juste après une parenthèse ouvrante ou tout juste avant une parenthèse fermante.
    - Le deux-points (`:`) et le signe égal (`=`) devraient être entourés d'espaces, sauf dans le cas des étiquettes de `case` et les spécificateurs d'accès (`private`, `protected` et `public`) qui ne devraient pas avoir d'espace avant le deux-points. Un espace devrait toujours suivre les éléments de ponctuation comme la virgule (`,`), le deux-points (`:`) et le point-virgule (`;`).
    - N'utilisez pas `NULL` ou `0` pour les pointeurs, mais bien `nullptr`. Dans le cas de test de `nullptr` avec un pointeur, soyez explicites : préférez `if (p != nullptr)` à `if (p)`.

- N’utilisez pas de `else` après un `return`.
- Les listes d’initialisation devraient suivre l’un des deux formats suivants :

```
Class::Class(int a, int b)
    : a_(a)
    , b_(b)
{
}
```

```
Class::Class(int a, int b)
    : a_(a),
      b_(b)
{
}
```

- Le corps de **tous** les énoncés de contrôle comme if, for, while et switch doit **toujours** être borné par des accolades, même s’il ne fait qu’une ligne.
- N’utilisez pas `using namespace std` dans les fichiers d’en-tête.

- Pour que vous puissiez compiler au fur et à mesure, le code vous est fourni avec certaines lignes en commentaire pour ne pas causer d’erreurs avant que vous n’ayez écrit les fonctions. N’oubliez pas de lignes en commentaire! Elles sont toutes accompagnées d’un « TODO ».

Travail à réaliser

On vous demande de compléter les fichiers qui vous sont fournis pour pouvoir implémenter le programme décrit ci-dessous. Vous ne devriez pas avoir à toucher aux fichiers d'en-tête, à l'exception du fichier `Foncteurs.h` et au fichier `Tests.h` pour y activer les tests.

Classe `GestionnaireUtilisateurs`

Gère la possession exclusive des `Utilisateurs` chargés du fichier `utilisateurs.txt`. Les utilisateurs sont sauvegardés dans une map pour les retrouver à partir de leur ID, soit leur courriel. Il ne peut donc pas exister deux utilisateurs avec le même courriel en même temps dans la map, puisque les clés d'une map doivent être uniques.

Variables membres :

- `std::unordered_map<std::string, Utilisateur> utilisateurs_`
  - Map de string vers l'utilisateur au nom associé.

Méthodes à implémenter :

- `bool ajouterUtilisateur(const Utilisateur& utilisateur)`
  - Nombre maximal de lignes : 1
  - Ajoute un utilisateur au gestionnaire, en l'insérant dans la map avec son ID comme clé et l'utilisateur comme valeur.
  - Retourne true si l'utilisateur a été ajouté avec succès, false si l'utilisateur n'a pas pu être ajouté puisque son ID était déjà présent dans la map.
  - Indice : regardez la valeur de retour de la fonction `std::unordered_map::emplace` sur cppreference!
  - Guide de codage : pour ajouter des éléments aux maps, utilisez `emplace` plutôt que `insert`, puisqu'en utilisant `emplace`, vous n'avez pas besoin de construire une paire avant de l'ajouter, comme ce serait le cas avec insert (il faudrait alors utiliser `std::make_pair` ou le constructeur de `std::pair`).
- `bool supprimerUtilisateur(const std::string& idUtilisateur)`
  - Nombre maximal de lignes : 1
  - Supprime un utilisateur du gestionnaire à partir de son ID.
  - Retourne true si l'utilisateur a été trouvé et supprimé avec succès, false si l'utilisateur n'a pas pu être supprimé puisque son ID n'était pas déjà présent dans la map.
  - Indice : regardez la valeur de retour de la fonction que vous utiliserez!
- `std::size_t getNombreUtilisateurs() const`
  - Nombre maximal de lignes : 1
  - Retourne le nombre d'utilisateurs présentement dans le gestionnaire.

- `const Utilisateur* getUtilisateurParId(const std::string& id) const`
  - Nombre maximal de lignes : 4
  - Trouve et retourne un utilisateur en le cherchant à partir de son ID.
  - Retourne un pointeur vers l'utilisateur ou `nullptr` si aucun utilisateur avec le ID ne peut être trouvé.
  - Note : vous ne pouvez pas utiliser `std::unordered_map::operator[]` puisque la méthode `getUtilisateurParId` est constante mais l'opérateur de la variable membre ne l'est pas. `operator[]` sur une map n'est pas constant puisque son comportement est de créer un nouvel élément initialisé par défaut s'il ne trouve aucun élément avec la clé donnée, ce qui est une opération modifiante. De toute façon, dans notre cas, on ne veut pas créer un nouvel `Utilisateur` initialisé par défaut si le ID donné ne se trouve pas dans la map, donc le `const` sur la méthode nous empêche d'écrire du code qui aurait été accidentellement incorrect.
  - Indice : conservez l'itérateur retourné par la fonction `find` de la map dans une variable locale pour le tester et pour retourner l'utilisateur facilement si jamais l'itérateur indique qu'il a été trouvé.
  - Guide de codage : si une fonction existe à la fois comme fonction membre (méthode) d'un conteneur et comme une fonction libre (fonction globale) du header `algorithm` de la librairie standard, utilisez la fonction membre. Si une version en fonction membre existe, c'est qu'elle est soit beaucoup plus efficace (dans le cas de `std::unordered_map::find`, la recherche est en temps constant), ou que la version en fonction libre ne fonctionnerait pas à cause de restrictions sur le type d'itérateurs (par exemple, `std::list::sort` doit être utilisée parce que `std::sort(std::list)` ne fonctionnerait pas puisque les itérateurs de `std::list` ne sont pas *random-access*).

Méthodes à modifier :

- `std::ostream& operator<<(std::ostream& outputStream, const GestionnaireUtilisateurs& gestionnaireUtilisateurs)`
  - Vous devez réécrire la boucle de la fonction en utilisant une *range-based for loop* et un *structured binding declaration*, plutôt que d'utiliser une boucle à itérateurs.
  - Les range-based for loops sont de la forme `for (const auto& element : container)` {…}. On utilise une référence constante lorsqu'on ne veut pas modifier les éléments, et on enlève le const si on veut les modifier. Référence recommandée : https://en.cppreference.com/w/cpp/language/range-for
  - Les structured binding declarations permettent d'assigner des sous-objets d'un élément à des variables. Ils ont la syntaxe suivante : `auto [x, y, …] = {array, tuple, paire ou une struct}`. Par exemple, `auto [a, b, c] = {4, 5, 6};` initialisera trois variables, `a` et `b` et `c` chacune de type `int`, et `a = 4`, `b = 5` et `c = 6`. On peut notamment utiliser les structured bindings avec une paire pour

en extraire les éléments. Ceci est très pratique puisque les maps retournent des itérateurs vers des paires `{key, value}` et l'utilisation de structured bindings nous permet d'avoir accès à ces deux éléments très facilement. On peut alors écrire, pour une `std::pair item(key, value)`, le structured binding suivant : `auto [key, value] = item`. Référence recommandée : https://en.cppreference.com/w/cpp/language/structured_binding

- Pour combiner ces deux concepts, il faut alors que l'élément dans le range-based for loop soit `const auto& [key, value]` puisqu'on veut avoir une référence constante à la clé et la valeur à chaque itération plutôt que d'en faire une copie potentiellement lourde à chaque fois (surtout pour ce qui est de copier la valeur, qui peut être un type complexe). La syntaxe finale pour itérer à travers une map de la façon la plus élégante possible devient donc la suivante : `for (const auto& [key, value] : map) {…}`.

Foncteurs

Il faut créer trois foncteurs dans le fichier `Foncteurs.h`. Vous remarquerez qu'il est vide : n'oubliez donc pas d'écrire vos gardes d'inclusion (*header guards*)! Les foncteurs sont les suivants :

`EstDansIntervalleDatesFilm` :

- Foncteur prédicat unaire servant à déterminer si un film est sorti entre deux années (inclusivement).
- Il doit être construit à partir de deux paramètres, soit les bornes inférieure et supérieure sur l'année de sortie du film qui lui sera éventuellement envoyé en appelant `operator()`.
- Son `operator()` doit prendre en paramètre une référence **constante** vers un `std::unique_ptr` tenant un film (on ne pourrait pas copier un `std::unique_ptr` et on ne veut pas le modifier), puisque le foncteur sera utilisé dans la classe `GestionnaireFilms` sur un conteneur de `std::unique_ptr<Film>`. Il retourne true si l'année de sortie est à l'intérieur de l'intervalle d'années (inclusivement), false sinon.

`ComparateurLog` :

- Foncteur prédicat binaire comparant les dates des lignes de log pour indiquer si elles sont en ordre chronologique.
- Son `operator()` doit prendre en paramètre deux références **constantes** (on ne veut pas copier ni modifier les paramètres) vers des `LigneLogs`. Il retourne true si la date de la première `LigneLog` envoyée en paramètre est strictement plus ancienne que la date de la deuxième `LigneLog`, false sinon. Puisque les dates sont en format ISO 8601, elles peuvent

être conservées dans une string et une date sera plus ancienne qu'une autre si sa string est inférieure selon une comparaison lexicographique (utilisez < sur les strings).

`ComparateurSecondElementPaire` :

- Template de foncteur prédicat binaire comparant les seconds éléments de paires pour déterminer si elles sont en ordre. On veut que le foncteur fonctionne avec des paires contenant n'importe quels éléments, et c'est pourquoi on en a fait un template.
- Le template de classe prend deux paramètre de template, soit les types `T1` et `T2` tenus par une paire hypothétique `std::pair<T1, T2>`.
- Son `operator()` doit prendre en paramètre deux références **constantes** (on ne veut pas copier ni modifier les paramètres) vers des paires dont les éléments sont de type `T1` et `T2`. Il retourne true si le second élément de la première paire est strictement inférieur au second élément de la deuxième paire, false sinon. Référence recommandée : https://en.cppreference.com/w/cpp/utility/pair.

Classe `GestionnaireFilms`

Gère la possession exclusive des `Films` chargés du fichier `films.txt`. Les films sont sauvegardés dans un vecteur de pointeurs uniques, et la classe possède trois maps agissant comme **filtres** pour rechercher rapidement des films en fonction d'un critère. Les trois maps **contiennent donc les films par agrégation, ce qui explique pourquoi ils ont un pointeur brut vers les films.** (Les pointeur bruts sont const puisqu'il n'y a aucune raison de vouloir permettre une modification d'un film à travers un filtre.)

Variables membres :

- `std::vector<std::unique_ptr<Film>> films_`
  - Vecteur de pointeurs contenant les films par composition. On utilise un vecteur de pointeurs vers des films plutôt qu'un vecteur de films pour que les éléments des trois filtres ne deviennent pas invalidés lorsque les adresses des éléments du vecteur changent quand il est redimensionné. Les trois filtres ont donc des pointeurs bruts pointant vers l'adresse qui est tenue par chacun des pointeurs intelligents dans le vecteur, plutôt que des pointeurs pointant vers l'adresse à un index du vecteur, qui serait susceptible de changer en ajoutant des éléments au vecteur et en causant un doublage de sa capacité en mémoire.
- `std::unordered_map<std::string, const Film*> filtreNomFilms_`
  - Filtre associant le nom d'un film au film en question. La conception de la classe a été faite de telle sorte qu'un seul film ne peut avoir le même nom, ce qui permet de relier un nom directement à un seul film. Vous devrez garder cette contrainte en tête dans l'implémentation de vos méthodes.

- `std::unordered_map<Film::Genre, std::vector<const Film*>> filtreGenreFilms_`
    - Filtre associant un genre de film à la liste de films ayant ce genre. Contrairement au filtre par nom, plus d'un film peut bien évidemment appartenir à un genre. C'est donc pourquoi cette map relie un genre à une liste de films (un vecteur de pointeurs non possédants vers des films).
- `std::unordered_map<Pays, std::vector<const Film*>> filtrePaysFilms_`
    - Filtre associant le pays d'origine d'un film à la liste de films provenant de ce pays. Même principe que pour le filtre par genre.

Méthodes à implémenter :

- `const Film* getFilmParNom(const std::string& nom) const`
    - Nombre maximal de lignes : 4
    - Trouve et retourne un film en le cherchant à partir de son nom.
    - Retourne un pointeur vers le film ou `nullptr` si aucun film avec le nom ne peut être trouvé.
    - Note : vous devez effectuer cette recherche de la façon la plus performante possible. Pensez à utiliser un des filtres!
    - Regardez les indices et conseils donnés plus haut pour `GestionnaireUtilisateurs::getUtilisateurParId`.
- `bool ajouterFilm(const Film& film)`
    - Ajoute un film au gestionnaire et met à jour les filtres en conséquence.
    - Retourne true si le film a été ajouté avec succès, false si le film n'a pas pu être ajouté puisque son nom était déjà présent dans le filtre par nom (il ne peut y avoir qu'un film avec le même nom à la fois).
    - Pensez à réutiliser une fonction membre pour voir si le film a déjà été ajouté!
    - Après avoir ajouté le film au vecteur, n'oubliez pas d'ajouter le film aux trois filtres. Réfléchissez bien à quel pointeur il faut utiliser!
    - Dans le cas des filtres ayant un vecteur comme valeur de la map, pensez à utiliser `operator[]` sur la map. Ceci créera automatiquement un vecteur vide à la clé donnée, et vous pourrez tout simplement y ajouter un élément avec `std::vector::push_back`, si aucune entrée pour le genre ou le pays n'avait été ajoutée jusqu'ici.
    - Regardez les indices et conseils donnés plus haut pour `GestionnaireUtilisateurs::ajouterUtilisateur`.
- `bool supprimerFilm(const std::string& nomFilm)`
    - Supprime un film du gestionnaire à partir de son nom.
    - Retourne true si le film a été trouvé et supprimé avec succès, false si le film n'a pas pu être supprimé puisqu'aucun film avec le nom donné n'a pu être trouvé.

- Note : vous devez effectuer cette recherche de la façon la plus performante possible. Évitez d'itérer plus d'une fois (avec des algorithmes) à travers le vecteur de films.
- Conseil : assurez-vous de ne pas effacer le film du vecteur avant de l'avoir supprimé des filtres, pour pouvoir accéder à ses attributs lorsque pertinent. Si vous le supprimez du vecteur avant, vous ne pourrez pas accéder à son genre pour l'effacer de la liste en question dans le filtre par genre, par exemple.
- Indice : vous aurez besoin d'un itérateur pour effacer le film du vecteur s'il est trouvé parce que la méthode pertinente du vecteur demande un itérateur. Commencez donc par trouver une façon d'obtenir un itérateur vers le film ayant le nom donné en paramètre. Puisqu'on n'a qu'une caractéristique du film et non l'objet complet qu'on recherche, un simple `std::find` ne suffit pas : il vous faudra utiliser un algorithme de la librairie standard ainsi qu'une expression lambda.
- Indice : une fois l'itérateur obtenu, vous pouvez tester sa valeur pour savoir si un film ayant ce nom existe dans le vecteur. Si c'est le cas, vous pouvez déréférencer l'itérateur pour obtenir le film.
- Indice : la méthode de suppression d'un élément par clé peut être utilisée directement pour le filtre par nom du film.
- Conseil : pour les deux autres filtres ayant comme valeur un vecteur, vous devez supprimer le film des listes par genre ou par pays qui y font référence. Commencez d'abord par initialiser une référence au vecteur associé à la clé pertinente (le genre ou le pays). Cette référence vous sera utile à la prochaine étape, pour raccourcir la longueur de la ligne de code pour effacer le film de la liste.
- Pour les filtres par genre et par pays, vous devez trouver une façon de supprimer un élément d'un vecteur en fonction de sa valeur, et non en fonction de son index (puisqu'on ne sait pas quel index est associé à l'élément qu'on cherche). **Prenez le temps de bien comprendre** le *erase-remove idiom*, qui permet de supprimer tous les éléments satisfaisant un critère de façon très performante : https://en.wikipedia.org/wiki/Erase%E2%80%93remove_idiom.
- Guide de codage : aucune boucle for ne devrait être utilisée! La fonctionnalité demandée s'implémente très bien avec les algorithmes et les méthodes des conteneurs de la librairie standard.
- Note : contrairement aux TPs précédents, n'utilisez pas d'échange de valeurs suivi d'un `pop_back`! On cherche à conserver l'ordre d'insertion des films dans les listes des filtres grâce au patron *erase-remove*.

- `std::size_t getNombreFilms() const`
  - Nombre maximal de lignes : 1
  - Retourne le nombre de films présentement dans le gestionnaire.
- `std::vector<const Film*> GestionnaireFilms::getFilmsParGenre(Film::Genre genre) const`

  - Nombre maximal de lignes : 4
  - Retourne une copie de la liste des films appartenant à un genre donné.
  - Attention au cas où aucun film n'a été ajouté avec ce genre, faisant en sorte qu'aucune liste associée à ce genre n'existe dans le filtre! Vous devrez retourner un vecteur vide.
- `std::vector<const Film*> GestionnaireFilms::getFilmsParPays(Pays pays) const`
  - Nombre maximal de lignes : 4
  - Retourne une copie de la liste des films appartenant à un genre donné.
  - Attention au cas où aucun film n'a été ajouté avec ce pays, faisant en sorte qu'aucune liste associée à ce pays n'existe dans le filtre! Vous devrez retourner un vecteur vide.
- `std::vector<const Film*> GestionnaireFilms::getFilmsEntreAnnees(int anneeDebut, int anneeFin)`
  - Nombre maximal de lignes : 3
  - Retourne une liste des films produits entre deux années passées en paramètre en utilisant un algorithme de la librairie standard et un des foncteurs que vous avez écrits plus haut.
  - Conseil : puisqu'on veut ici créer et retourner un vecteur de pointeurs bruts (qui ne possèdent pas les films, mais qui ne font que les observer) à partir du vecteur de `std::unique_ptr` gardé comme variable membre, il faut que l'itérateur inséreur appelle automatiquement un `get()` sur chaque pointeur intelligent en copiant les éléments. L'itérateur inséreur arrière custom `RawPointerBackInserter` vous est donc fourni pour cette raison : utilisez-le à la place de `std::back_inserter`, avec la même syntaxe que vous utiliseriez normalement, soit `RawPointerBackInserter(conteneur)`, où `conteneur` a une méthode `push_back`.

Méthodes à modifier :

- `std::ostream& operator<<(std::ostream& outputStream, const GestionnaireFilms& gestionnaireFilms)`
  - Vous devez réécrire la boucle de la fonction en utilisant deux *range-based for loops* et un *structured binding declaration*, plutôt que d'utiliser une boucle à itérateurs et une boucle à index.
  - Regardez les indices et conseils donnés plus haut pour `operator<<` avec la classe `GestionnaireUtilisateurs`.

Classe `AnalyseurLogs`

Gère la possession exclusive des `LigneLogs` chargées du fichier `logs.txt` et permet d'en extraire des statistiques.

Variables membres :

- `std::vector<LigneLog> logs_`
  - Vecteur ordonné contenant les lignes de log par composition.
  - Puisqu'on veut que les lignes de log soient en tout temps gardées en ordre chronologique dans le vecteur, chaque nouvelle insertion se fait en respectant à la position respectant l'ordre préexistant.
- `std::unordered_map<const Film*, int> vuesFilms_`
  - Filtre associant un film à son nombre de vues par tous les utilisateurs sur toute la période chargée du fichier `logs.txt`. Cette variable membre sera utile pour trouver quels films sont les plus populaires de la façon la plus performante. Elle doit cependant être gardée à jour : chaque ajout d'une ligne log doit incrémenter le nombre de vue du film de la ligne de log.

Méthodes à implémenter :

- `bool creerLigneLog(const std::string& timestamp, const std::string& idUtilisateur, const std::string& nomFilm, GestionnaireUtilisateurs& gestionnaireUtilisateurs, GestionnaireFilms& gestionnaireFilms)`
  - Nombre maximal de lignes : 6
  - Crée et ajoute une ligne de log dans le vecteur de logs. Après avoir créé une instance de la struct LigneLog, appelle la fonction `ajouterLigneLog`.
  - Retourne true si le film et l'utilisateur existaient et le log a été ajouté avec succès, false sinon.
- `void ajouterLigneLog(const LigneLog& ligneLog)`
  - Nombre maximal de lignes : 3
  - Ajoute une ligne de log en ordre chronologique dans le vecteur de logs tout en mettant à jour le nombre de vues.
  - Vous devez utiliser un algorithme de la librairie standard pour insérer la ligne de log directement à sa position ordonnée à l'intérieur du vecteur, sachant que le vecteur est toujours ordonné. L'opération de recherche est efficace puisqu'elle utilise un *binary search* pour trouver la position d'insertion le plus rapidement possible. Cependant, il faut noter que l'insertion ne sera pas particulièrement performante, puisque pour insérer un objet au milieu d'un vecteur, il faut déplacer tous les éléments suivant la position d'insertion d'une position vers la fin du vecteur pour accommoder le nouvel élément.

  - Pensez à utiliser un des foncteurs que vous avez créés!
- `int getNombreVuesFilm(const Film* film) const`
  - Nombre maximal de lignes : 4
  - Retourne le nombre de vues pour un film passé en paramètre. Cette opération doit se faire de la façon la plus performante possible (en temps constant).
- `const Film* getFilmPlusPopulaire() const`
  - Nombre maximal de lignes : 4
  - Retourne le film le plus regardé parmi les données chargées dans l'analyseur de logs. Si les logs sont vides (aucun film n'est trouvé), retourne nullptr.
  - Cette opération doit se faire de la façon la plus performante possible (en temps constant). N'oubliez pas que la map `vuesFilms_` existe!
  - Utilisez un algorithme de la librarie standard avec le foncteur `ComparateurSecondElementPaire` que vous avez créé plus tôt pour accomplir ceci.
- `std::vector<std::pair<const Film*, int>> getNFilmsPlusPopulaires(std::size_t nombre) const`
  - Nombre maximal de lignes : faisable en 4, mais un peu plus est acceptable.
  - Retourne une liste des films les plus regardés et leur nombre de vues parmi les données chargées dans l'analyseur de logs.
  - Prend en paramètre le nombre de films les plus populaires à retourner. Si le nombre envoyé est supérieur au nombre de films distincts, alors le nombre de films retournées est égal au nombre de films distincts contenus dans l'analyseur de logs.
  - Indice : pensez à précalculer la taille du vecteur qui sera retourné par la fonction en utilisant la fonction `std::min` de la librairie standard.
  - La méthode retourne un vecteur contenant la liste des n paires de pointeurs vers les films les plus populaires et leur nombre de vues.
  - Conseil : vous aurez besoin d'initialiser un `std::vector` avec une certaine taille pour y ajouter les résultats. Vous pouvez initialiser un vecteur avec une certaine taille dès sa construction grâce à son constructeur.
  - Il existe un algorithme qui copie des éléments dans un intervalle de destination de façon ordonnée selon un certain prédicat. Utilisez-le avec une expression lambda, dont le comportement sera très semblable au foncteur `ComparateurSecondElementPaire` que vous avez utilisé pour `AnalyseurLogs::getFilmPlusPopulaire`, sauf pour le fait qu'il retournera true si le second élément de la première paire est **plus grand que** le second élément de la deuxième paire (soit le comportement inverse).
- `int getNombreVuesPourUtilisateur(const Utilisateur* utilisateur) const`
  - Nombre maximal de lignes : 1 (mais le corps de l'expression lambda sera sur sa propre ligne).

  - Prend en paramètre l'utilisateur pour lequel on veut compter le nombre de vues, pour le bombarder à son insu de plus d'annonces pour CinéPoly Premium.
  - Retourne le nombre de films vus par un utilisateur en utilisant un algorithme de la librairie standard et une expression lambda comme prédicat. Notez que vous aurez besoin de capturer une variable dans votre lambda (une variable entre crochets).
  - Indice : puisque chaque ligne de log associe un visionnement d'un film à un utilisateur, le nombre de lignes de log faisant référence à un utilisateur représente logiquement le nombre de vues pour cet utilisateur. Sachant cela, il existe un algorithme de la librairie standard qui compte le nombre d'éléments qui satisfont à un prédicat (dans notre cas, le prédicat sera une expression lambda).
- `std::vector<const Film*> getFilmsVusParUtilisateur(const Utilisateur* utilisateur) const`
  - Retourne un vecteur des films **uniques** regardés par un utilisateur parmi les données chargées dans l'analyseur de logs.
  - Prend en paramètre l'utilisateur pour lequel on cherche à savoir la liste de films distincts regardés. Retourne un vecteur contenant la liste des pointeurs vers les films regardés par l'utilisateur.
  - Indice : le défi principal de cette méthode consiste à construire une liste de films sans qu'il y ait de film en double. Pour ce faire, vous pouvez utilisez un `std::set` (ou encore mieux, un `std::unordered_set`, qui sera plus performant). Puisqu'un set ne peut contenir que des éléments uniques (et une insertion d'un élément déjà existant sera ignorée), on peut ajouter au set tous les films, et ensuite extraire les éléments du set dans un vecteur pour obtenir un vecteur d'éléments uniques.
  - Pour faire l'insertion des éléments dans le set, le plus facile est d'utiliser une boucle for, tant que celle-ci est une range-based for loop.
  - Enfin, pour retourner un vecteur à partir des élments une fois insérés dans le set, vous pouvez utiliser le constructeur de `std::vector` avec deux itérateurs.

Bonus de 1/20

Vous pouvez obtenir un bonus d'un point si vous utilisez les classes que vous avez complétées pour afficher des informations à l'écran (dans le même ordre). Notez que vous devrez charger les données à partir des fichiers `*.txt` fournis. Écrivez le code pour l'affichage directement dans la fonction `main`, juste après les tests. Vous êtes encouragés à utiliser les range-based for loops pour cette partie, vous n'avez pas besoin d'utiliser les itérateurs de streams. L'affichage attendu se situe en annexe, à la toute fin de cet énoncé.

Correction

La correction du TP se fait sur 20 points :

- [3 pt] Compilation du programme (la compilation ne doit pas émettre d'avertissements).
- [2 pt] Exécution du programme.
- [6 pt] Comportement exact des méthodes du programme (passage des tests).
- [5 pt] Respect des consignes d'implémentation (nombre de lignes max, utilisation de la librairie standard, pas de boucles for, utilisation des lambdas et des foncteurs, etc.)
- [4 pt] Qualité et documentation du code (respect des normes et des conventions de style).
- [+1 pt bonus]

Relisez bien les spécifications générales et les consignes de remise au début du TP avant de remettre votre travail!

Remarque : si vous avez des suggestions/commentaires constructifs/commentaires désobligeants, n'hésitez pas à donner votre feedback!

```
❯ make run | lolcat
Starting program: bin/linux/debug/project

Tests pour GestionnaireUtilisateurs:
--------
Test  1: GestionnaireUtilisateurs::ajouterUtilisateur      : OK
Test  2: GestionnaireUtilisateurs::supprimerUtilisateur    : OK
Test  3: GestionnaireUtilisateurs::getNombreUtilisateurs   : OK
Test  4: GestionnaireUtilisateurs::getUtilisateurParId     : OK
Test  5: GestionnaireUtilisateurs::operator<<              : OK
--------
Total pour la section: 1/1

Tests pour Foncteurs:
--------
Test  1: Foncteur EstDansIntervalleDatesFilm               : OK
Test  2: Foncteur ComparateurLog                           : OK
Test  3: Foncteur ComparateurSecondElementPaire            : OK
--------
Total pour la section: 1/1

Tests pour GestionnaireFilms:
--------
Test  1: GestionnaireFilms::ajouterFilm                    : OK
Test  2: GestionnaireFilms::supprimerFilm                  : OK
Test  3: GestionnaireFilms::getNombreFilms                 : OK
Test  4: GestionnaireFilms::getFilmParNom                  : OK
Test  5: GestionnaireFilms::getFilmsParGenre               : OK
Test  6: GestionnaireFilms::getFilmsParPays                : OK
Test  7: GestionnaireFilms::getFilmsEntreAnnees            : OK
Test  8: GestionnaireFilms::operator<<                     : OK
Test  9: Chargement et copy ctor toujours fonctionnels     : OK
--------
Total pour la section: 2/2

Tests pour AnalyseurLogs:
--------
Test  1: AnalyseurLogs::creerLigneLog                      : OK
Test  2: AnalyseurLogs::ajouterLigneLog                    : OK
Test  3: AnalyseurLogs::getNombreVuesFilm                  : OK
Test  4: AnalyseurLogs::getFilmPlusPopulaire               : OK
Test  5: AnalyseurLogs::getNFilmsPlusPopulaires            : OK
Test  6: AnalyseurLogs::getNombreVuesPourUtilisateur       : OK
Test  7: AnalyseurLogs::getFilmsVusParUtilisateur          : OK
--------
Total pour la section: 2/2

Total pour tous les tests: 6/6

Le gestionnaire d'utilisateurs contient 100 utilisateurs:
       Identifiant: bebing@mac.com | Nom: Rupert Irvine | Âge: 9 | Pays: Mexique
       Identifiant: dsowsy@yahoo.com | Nom: Angla Corrigan | Âge: 37 | Pays: Mexique
       Identifiant: tlinden@outlook.com | Nom: Thu Doughty | Âge: 22 | Pays: Canada
       Identifiant: animats@sbcglobal.net | Nom: Nickole Montero | Âge: 91 | Pays: Mexique
       Identifiant: cosimo@live.com | Nom: Ilana Cervantes | Âge: 71 | Pays: Canada
       Identifiant: codex@mac.com | Nom: Stan Durand | Âge: 3 | Pays: Royaume-Uni
       Identifiant: mfburgo@live.com | Nom: Jaimee Chester | Âge: 36 | Pays: Royaume-Uni
       Identifiant: philen@msn.com | Nom: Gaston Hundley | Âge: 2 | Pays: Chine
       Identifiant: dogdude@icloud.com | Nom: Renetta Camarillo | Âge: 8 | Pays: Japon
       Identifiant: rddesign@me.com | Nom: Chanel Arevalo | Âge: 30 | Pays: Mexique
       Identifiant: fatelk@comcast.net | Nom: Gay Shuler | Âge: 71 | Pays: États-Unis
       Identifiant: notaprguy@verizon.net | Nom: Kacie Heflin | Âge: 80 | Pays: Canada
       Identifiant: zyghom@yahoo.com | Nom: Easter Findley | Âge: 74 | Pays: Japon
       Identifiant: kwilliams@msn.com | Nom: Emmanuel Mcwhorter | Âge: 31 | Pays: Japon
       Identifiant: philb@yahoo.ca | Nom: Hildred Fernandes | Âge: 89 | Pays: Chine
       Identifiant: danzigism@att.net | Nom: Masako Shipp | Âge: 44 | Pays: Chine
       Identifiant: melnik@aol.com | Nom: Isiah Peralta | Âge: 50 | Pays: Canada
       Identifiant: kalpol@icloud.com | Nom: Chaya Massie | Âge: 21 | Pays: Mexique
       Identifiant: facet@verizon.net | Nom: Charisse Beckman | Âge: 73 | Pays: France
       Identifiant: rgarton@att.net | Nom: Christiana Alves | Âge: 82 | Pays: France
       Identifiant: gward@sbcglobal.net | Nom: Rocco Lacroix | Âge: 76 | Pays: Russie
       Identifiant: singh@me.com | Nom: Tanika Weston | Âge: 76 | Pays: Canada
       Identifiant: ivoibs@yahoo.ca | Nom: Louanne Bellamy | Âge: 83 | Pays: Russie
       Identifiant: plover@outlook.com | Nom: Julianne Sasser | Âge: 87 | Pays: Brésil
       Identifiant: pakaste@gmail.com | Nom: Corazon Toliver | Âge: 2 | Pays: France
       Identifiant: rasca@msn.com | Nom: Donnette Crawley | Âge: 76 | Pays: Canada
       Identifiant: scottlee@att.net | Nom: Kiley Tang | Âge: 1 | Pays: Japon
       Identifiant: jespley@me.com | Nom: Marylouise Otero | Âge: 26 | Pays: France
       Identifiant: moonlapse@msn.com | Nom: Verdell Reichert | Âge: 25 | Pays: Chine
       Identifiant: sharon@icloud.com | Nom: Karine Herrington | Âge: 97 | Pays: Canada
       Identifiant: bartak@outlook.com | Nom: Eli Burdette | Âge: 86 | Pays: Russie
       Identifiant: leakin@yahoo.ca | Nom: Gita Stitt | Âge: 72 | Pays: Russie
       Identifiant: wainwrig@aol.com | Nom: Danyelle Craddock | Âge: 16 | Pays: Chine
       Identifiant: dinther@me.com | Nom: Deetta Edmond | Âge: 94 | Pays: Chine
       Identifiant: mlewan@hotmail.com | Nom: Tambra Wilbanks | Âge: 59 | Pays: Russie
       Identifiant: thrymm@aol.com | Nom: Olin Staton | Âge: 58 | Pays: Chine
       Identifiant: wikinerd@icloud.com | Nom: Georgetta Aquino | Âge: 10 | Pays: Japon
       Identifiant: ivoibs@yahoo.com | Nom: Jenise Oates | Âge: 45 | Pays: Russie
       Identifiant: dmouse@mac.com | Nom: Marlin Mcgregor | Âge: 51 | Pays: Chine
       Identifiant: gravyface@live.com | Nom: Euna Spinks | Âge: 49 | Pays: Canada
       Identifiant: dawnsong@icloud.com | Nom: Stephine Jamison | Âge: 16 | Pays: Canada
       Identifiant: mxiao@verizon.net | Nom: Marla Dempsey | Âge: 39 | Pays: États-Unis
       Identifiant: karasik@hotmail.com | Nom: Luis Noe | Âge: 18 | Pays: États-Unis
       Identifiant: mcsporran@yahoo.ca | Nom: Anya Baez | Âge: 73 | Pays: Canada
       Identifiant: klaudon@hotmail.com | Nom: Salena Christy | Âge: 39 | Pays: Royaume-Uni
       Identifiant: mglee@outlook.com | Nom: Sharron Parson | Âge: 67 | Pays: Royaume-Uni
       Identifiant: mastinfo@live.com | Nom: Echo Shumate | Âge: 41 | Pays: États-Unis
```

Identifiant: jkegl@verizon.net | Nom: Retta Wheatley | Âge: 91 | Pays: Brésil
Identifiant: tubajon@live.com | Nom: Irish Gilmore | Âge: 21 | Pays: États-Unis
Identifiant: noticias@optonline.net | Nom: Adelaide Blank | Âge: 10 | Pays: Canada
Identifiant: chaffar@live.com | Nom: Babara Guerin | Âge: 52 | Pays: Russie
Identifiant: pdbaby@msn.com | Nom: Chas Spain | Âge: 27 | Pays: Royaume-Uni
Identifiant: grdschl@icloud.com | Nom: Lavenia Beach | Âge: 10 | Pays: Mexique
Identifiant: mbswan@optonline.net | Nom: Pattie Delgadillo | Âge: 84 | Pays: Canada
Identifiant: parksh@msn.com | Nom: Alden Tyner | Âge: 21 | Pays: Chine
Identifiant: denton@me.com | Nom: Vashti Snipes | Âge: 80 | Pays: Canada
Identifiant: karasik@msn.com | Nom: Carman Gipson | Âge: 97 | Pays: Russie
Identifiant: falcao@verizon.net | Nom: Breanne Hawley | Âge: 86 | Pays: Brésil
Identifiant: ijackson@gmail.com | Nom: Seymour Knudson | Âge: 28 | Pays: Japon
Identifiant: dvdotnet@yahoo.ca | Nom: Shirlee Keating | Âge: 82 | Pays: Canada
Identifiant: pajas@comcast.net | Nom: Eldora Schaffer | Âge: 66 | Pays: Royaume-Uni
Identifiant: wetter@yahoo.com | Nom: Mignon Haag | Âge: 69 | Pays: Canada
Identifiant: kourai@verizon.net | Nom: Kendall Dias | Âge: 84 | Pays: Chine
Identifiant: barjam@sbcglobal.net | Nom: Kennith Doan | Âge: 69 | Pays: États-Unis
Identifiant: cremonini@verizon.net | Nom: Truman Sawyer | Âge: 20 | Pays: Mexique
Identifiant: policies@yahoo.com | Nom: Fatimah Teal | Âge: 2 | Pays: Brésil
Identifiant: bester@optonline.net | Nom: Margherita Noland | Âge: 50 | Pays: États-Unis
Identifiant: akoblin@optonline.net | Nom: Marnie Melvin | Âge: 37 | Pays: Chine
Identifiant: augusto@verizon.net | Nom: Rhett Sargent | Âge: 15 | Pays: Mexique
Identifiant: srour@verizon.net | Nom: Cyndi Marino | Âge: 42 | Pays: Russie
Identifiant: srour@me.com | Nom: Johnetta Adler | Âge: 72 | Pays: Chine
Identifiant: mrobshaw@outlook.com | Nom: Joella Sparkman | Âge: 43 | Pays: Brésil
Identifiant: bsikdar@gmail.com | Nom: Emerald Stubbs | Âge: 5 | Pays: France
Identifiant: andrei@hotmail.com | Nom: Armanda Hooper | Âge: 11 | Pays: Royaume-Uni
Identifiant: frode@sbcglobal.net | Nom: Lane Britton | Âge: 25 | Pays: Brésil
Identifiant: mcraigw@verizon.net | Nom: Anjanette Seitz | Âge: 58 | Pays: France
Identifiant: claypool@aol.com | Nom: Hilton Coles | Âge: 90 | Pays: Mexique
Identifiant: froodian@mac.com | Nom: Florrie Carmichael | Âge: 50 | Pays: Brésil
Identifiant: unreal@aol.com | Nom: Necole Johansen | Âge: 80 | Pays: Royaume-Uni
Identifiant: linuxhack@yahoo.com | Nom: Lennie Hatley | Âge: 43 | Pays: Royaume-Uni
Identifiant: alhajj@mac.com | Nom: Toccara Patino | Âge: 65 | Pays: Canada
Identifiant: carreras@mac.com | Nom: Roma Greco | Âge: 29 | Pays: Mexique
Identifiant: trieuvan@att.net | Nom: Mariana Pfeiffer | Âge: 85 | Pays: Chine
Identifiant: wortmanj@me.com | Nom: Bernie Napier | Âge: 1 | Pays: France
Identifiant: kobayasi@yahoo.com | Nom: Lindsay Lamar | Âge: 44 | Pays: France
Identifiant: snunez@optonline.net | Nom: Mitchel Brandt | Âge: 2 | Pays: États-Unis
Identifiant: staikos@optonline.net | Nom: Christinia Rousseau | Âge: 14 | Pays: Mexique
Identifiant: gomor@optonline.net | Nom: Earlie Portillo | Âge: 57 | Pays: États-Unis
Identifiant: jfinke@comcast.net | Nom: Mara Yarbrough | Âge: 68 | Pays: Royaume-Uni
Identifiant: paley@msn.com | Nom: Magan Messer | Âge: 57 | Pays: France
Identifiant: fwitness@gmail.com | Nom: Arlinda Murillo | Âge: 62 | Pays: Japon
Identifiant: jdhedden@mac.com | Nom: Meghann Cope | Âge: 80 | Pays: Royaume-Uni
Identifiant: sopwith@hotmail.com | Nom: Russ Humphries | Âge: 98 | Pays: France
Identifiant: dkrishna@att.net | Nom: Shirely Kenney | Âge: 100 | Pays: Canada
Identifiant: rgiersig@live.com | Nom: Pierre Mcnabb | Âge: 28 | Pays: Chine
Identifiant: mschwartz@verizon.net | Nom: Lilly Valles | Âge: 84 | Pays: Brésil
Identifiant: temmink@att.net | Nom: Tracee Boyer | Âge: 97 | Pays: États-Unis
Identifiant: cantu@optonline.net | Nom: Amberly Bohannon | Âge: 99 | Pays: Brésil
Identifiant: kramulous@aol.com | Nom: Shizuko Beltran | Âge: 48 | Pays: Brésil
Identifiant: quinn@comcast.net | Nom: Lashell Bower | Âge: 54 | Pays: Russie

Films d'aventure:
Nom: A Simple Moment | Genre: Aventure | Pays: Mexique | Réalisateur: Denise Harvey | Année: 1968
Nom: Beyond the Surface | Genre: Aventure | Pays: France | Réalisateur: Kelley Stanley | Année: 1950
Nom: Cathode Mary | Genre: Aventure | Pays: France | Réalisateur: Emmett Schultz | Année: 1960
Nom: Constance Paine | Genre: Aventure | Pays: Chine | Réalisateur: Marguerite Carr | Année: 1943
Nom: Darwinter of my Discontent | Genre: Aventure | Pays: Canada | Réalisateur: Wilma Miles | Année: 1978
Nom: Dead Rockers | Genre: Aventure | Pays: Japon | Réalisateur: Kathleen Pierce | Année: 1987
Nom: Disorder in the Court | Genre: Aventure | Pays: Russie | Réalisateur: Marc Mendoza | Année: 1996
Nom: Drifting Off | Genre: Aventure | Pays: Chine | Réalisateur: Roosevelt Ward | Année: 2008
Nom: Fair Weather Father | Genre: Aventure | Pays: Mexique | Réalisateur: Marguerite Carr | Année: 1998
Nom: Father Earth | Genre: Aventure | Pays: États-Unis | Réalisateur: Laura Garcia | Année: 1952
Nom: Faustus Frieze | Genre: Aventure | Pays: Brésil | Réalisateur: Albert Webb | Année: 1973
Nom: Gumby on a Hot Tin Roof | Genre: Aventure | Pays: Russie | Réalisateur: William Cunningham | Année: 1963
Nom: Gumby Sleeps with Fishes | Genre: Aventure | Pays: Mexique | Réalisateur: Gilberto Dennis | Année: 1946
Nom: Gumby's Choice | Genre: Aventure | Pays: Russie | Réalisateur: Victoria Mckenzie | Année: 2001
Nom: Guru Smile | Genre: Aventure | Pays: Royaume-Uni | Réalisateur: Roy Thompson | Année: 1996
Nom: Keister's Kiss | Genre: Aventure | Pays: France | Réalisateur: Sherman Hubbard | Année: 1991
Nom: Landmark of My Youth | Genre: Aventure | Pays: Chine | Réalisateur: Susie Nichols | Année: 1994
Nom: Lone Licorice | Genre: Aventure | Pays: Chine | Réalisateur: Ed Carter | Année: 1991
Nom: Moments In Time | Genre: Aventure | Pays: Japon | Réalisateur: Hector Lucas | Année: 1944
Nom: Pike's Puke | Genre: Aventure | Pays: Mexique | Réalisateur: Delbert Young | Année: 2008
Nom: Primarily Susan | Genre: Aventure | Pays: Royaume-Uni | Réalisateur: Andre Simmons | Année: 1980
Nom: Sweet Rainbow | Genre: Aventure | Pays: Japon | Réalisateur: Marvin Weaver | Année: 1958
Nom: Tennyson Alley | Genre: Aventure | Pays: France | Réalisateur: Kristi Sherman | Année: 1943
Nom: The 4AM Club | Genre: Aventure | Pays: Mexique | Réalisateur: Shaun Keller | Année: 1957
Nom: The Avenging Art Angel | Genre: Aventure | Pays: Chine | Réalisateur: Donnie Wilson | Année: 1965
Nom: The Centre of the World | Genre: Aventure | Pays: Canada | Réalisateur: Donnie Wilson | Année: 2008
Nom: The False Front | Genre: Aventure | Pays: Brésil | Réalisateur: Courtney Salazar | Année: 2013
Nom: The Final Cabaña | Genre: Aventure | Pays: Japon | Réalisateur: Alan Sandoval | Année: 1982
Nom: The Fire Storm Verdict | Genre: Aventure | Pays: Mexique | Réalisateur: Terence Watson | Année: 1977
Nom: The Last Cabaña | Genre: Aventure | Pays: France | Réalisateur: Erick Bowen | Année: 1944
Nom: The Lemonade Stand | Genre: Aventure | Pays: Japon | Réalisateur: Rufus Ballard | Année: 1940
Nom: The Pharmaceuticals of Heaven | Genre: Aventure | Pays: Canada | Réalisateur: Tara Fields | Année: 1981
Nom: The Truth Turtles | Genre: Aventure | Pays: Canada | Réalisateur: Hector Lucas | Année: 1967
Nom: Time Gone By | Genre: Aventure | Pays: États-Unis | Réalisateur: Christopher Daniels | Année: 1956
Nom: Uncoil and Float | Genre: Aventure | Pays: Brésil | Réalisateur: Brett Herrera | Année: 1964
Nom: Undivided Highway | Genre: Aventure | Pays: Mexique | Réalisateur: Alan Sandoval | Année: 1987
Nom: Water Monk | Genre: Aventure | Pays: Japon | Réalisateur: Linda Hill | Année: 2012
Nom: Wednesday Morning Picnic | Genre: Aventure | Pays: Mexique | Réalisateur: Tricia Saunders | Année: 1984

Films produits de 1960 à 1961:
Nom: Cathode Mary | Genre: Aventure | Pays: France | Réalisateur: Emmett Schultz | Année: 1960
Nom: DreamTown | Genre: Horreur | Pays: Canada | Réalisateur: Evelyn Vaughn | Année: 1960
Nom: First Snow | Genre: Comédie | Pays: Brésil | Réalisateur: Charlene Lyons | Année: 1960

```
        Nom: Freedom From Fear | Genre: Fantastique | Pays: Chine | Réalisateur: Doreen Dawson | Année: 1960
        Nom: Lap Tiger | Genre: Romance | Pays: France | Réalisateur: Mercedes Washington | Année: 1961
        Nom: Me First Cafe | Genre: Comédie | Pays: Japon | Réalisateur: Shawn Lawson | Année: 1961
        Nom: One Foot Out the Door | Genre: Romance | Pays: Chine | Réalisateur: Carmen Alexander | Année: 1960
        Nom: The Parafin Directive | Genre: Fantastique | Pays: Russie | Réalisateur: Betsy Weber | Année: 1960

Film le plus populaire (94 vues): Nom: Free Leon | Genre: Drame | Pays: Russie | Réalisateur: Wilma Miles | Année: 2008

5 films les plus populaires:
        Nom: Free Leon | Genre: Drame | Pays: Russie | Réalisateur: Wilma Miles | Année: 2008 (94 vues)
        Nom: Lars Makeshift | Genre: Romance | Pays: Chine | Réalisateur: Victoria Mckenzie | Année: 2015 (89 vues)
        Nom: Exoville | Genre: Comédie | Pays: France | Réalisateur: Corey Mason | Année: 1962 (74 vues)
        Nom: Holly’s Holding Together | Genre: Horreur | Pays: Canada | Réalisateur: Ed Carter | Année: 1975 (73 vues)
        Nom: Pandora’s Boxing | Genre: Drame | Pays: Chine | Réalisateur: Christopher Daniels | Année: 2007 (73 vues)

Nombre de films vus par l'utilisateur karasik@msn.com: 189

 ~/Documents/Poly 2020-1/INF1010 - Chargé de laboratoire/TP5/Corrigé

❯
```