Vendredi 27 décembre 2023

Compte-rendu TP C++

Enseignant en charge : GINHAC Dominique

Corneille Martin

Dubost Lucie

Groupe : TD2-TP3

1ère année du cycle ingénieur en Infotronique

Plan du rapport

1. TP1 C++ : Un peu de lecture

1.1 – Motivation et fonctionnement

Afin de commencer ces TP, nous avons décidé de concert de débuter par l'un des travaux pratiques les plus simples, car nous n'avions encore jamais programmé en C++. Ainsi, appréciant tout deux tout particulièrement la lecture, nous avons naturellement choisi le premier TP, traitant d'une bibliothèque et de sa gestion. De plus, nous avons (justement) pensé que ce sujet nous permettrait de démarrer comme il se doit l'apprentissage de la programmation en C++.
Nous avons énormément codé ensemble, puisque les connaissances de l'un comblaient les lacunes de l'autre, et réciproquement.

1.2 – Organisation des classes

Après avoir lu une première fois le sujet, nous nous sommes plongés dans la réflexion de l'organisation des classes. Ainsi, comme les consignes le demandaient, nous avons commencé par adapter notre classe Date. Par la suite, nous avons créé les classes Livre, Auteur, Lecteur et Emprunt, avec les leurs méthodes et attributs respectifs. Il nous a de plus semblé intéressant de manipuler des énumérations en ce qui concerne les langues et les différents genres des livres. Il s'en retrouve plus simple d'ajouter des éléments à ces classes, et donc de les manipuler. Pour regrouper et gérer le tout, nous avons constitué une classe Bibliothèque. Mais permettez-nous de vous parler de ces classes plus en profondeur.

A) Livre

Un livre est défini par :
- un titre ;
- un auteur ;
- une langue ;
- un genre ;
- une date de publication ;
- un ISBN.

Sans compter le constructeur et les getters, les méthodes intéressantes sont :
- `setDispo(bool b)` : permet de rendre un livre disponible ou indisponible selon qu'il soit emprunté ou non par un lecteur. Elle se met donc à jour lors du rendu d'un ouvrage ;
- `operator == (Livre& livre) const` : une surcharge de l'opérateur == afin qu'il retourne `true` si les ISBN de deux livres sont similaires (on en conclut que ce sont les mêmes) ;
- `friend std::ostream& operator<<(std::ostream& out, Livre& livre)` : une fonction amie surchargeant l'opérateur << afin qu'il affiche toutes les informations d'un livre. Par exemple : La Divine Comédie de Dante Alighieri, écrit en 1320. Genre : Poésie. ISBN : 978-2080237224.

B) Auteur

Un auteur est défini par :
- un nom ;

- un prénom ;
- un identifiant composé de ses initiales suivies d'un nombre aléatoire ;
- une date de naissance.

Sans compter le constructeur et les getters, les méthodes intéressantes sont :

- `operator == (Auteur& auteur) const` : surcharge permettant de comparer deux auteur via leur identifiant respectif ;
- `std::ostream& operator<<(std::ostream& out, Auteur& auteur)` : surcharge affichant le prénom et le nom d'un auteur.

C) Lecteur

Un lecteur est défini par :

- un nom ;
- un prénom ;
- un identifiant composé de ses initiales suivies d'un nombre aléatoire ;
- une date de naissance ;
- une date d'inscription ;
- un vecteur des livres lus ;
- un vecteur des livres actuellement empruntés par un lecteur.

Sans compter le constructeur et les getters, les méthodes intéressantes sont :

- `display_livre_actuel()` : affiche la liste des livres actuellement empruntés par le lecteur ;
- `ajout_lus(Livre l)` : ajoute un ouvrage au vecteur des livres lus ;
- `suppr(Livre l)` : supprime un ouvrage de la liste des livres actuellement empruntés ;
- `friend std::ostream& operator<<(std::ostream& out, Lecteur& lecteur)` : une fonction amie surchargeant l'opérateur << et affichant les informations d'un lecteur ;
- `nombre_livres_lus()` : retourne le nombre de livres lus.

D) Genre et Langue

Les classes Genre et Langue sont deux énumérations. Il n'y a pas grand-chose à dire dessus.

E) Emprunt

Un emprunt est créé avec une date, un livre et un lecteur. Afin que le programme ne fasse pas de copie des objets manipulés, nous avons dû mettre des références à ces objets.
Les méthodes intéressantes de cette classe sont :

- `retour(Lecteur &lect, Livre &livre)` : si le livre est el et bien emprunté, alors la méthode le retire de la liste des livres actuellement empruntés par le lecteur et l'ajoute au vecteur des livres lus. De plus, elle passe la disponibilité de l'ouvrage à `true` ; sinon, le livre n'est pas emprunté et il n'y a rien à faire ;
- `autorisation(Livre l)` : un getter assez spécial, puisqu'il retourne la disponibilité du livre.

F) Bibliothèque

La classe Bibliothèque est la plus dense de toutes : en effet, elle regroupe toutes les autres et permet de manipuler l'entièreté de la bibliothèque. Son constructeur est vide car nous devons ajouter tous les éléments (livres, auteurs, lecteurs) dans le menu principal, en utilisant certaines des méthodes qui suivent :

- `add_lecteur(Lecteur &le)` / `add_livre(Livre &li)` / `add_auteur(Auteur &au)` : ajoute aux vecteurs correspondant le nouvel objet ;
- `livre_existe(long &ISBN)` / `lecteur_existe(std::string &id)` / `auteur_existe(std::string &id)` : nous permet de savoir si l'élément existe dans notre bibliothèque ;
- `cherche_livre(long &ISBN)` / `cherche_lecteur(std::string &id)` / `cherche_auteur(std::string &id)` : permet de retrouver un livre/lecteur/auteur grâce à son ISBN/id ;
- `livre_auteur(Auteur &au)` : liste tous les livres d'un auteur ;
- `livre_emprunte()` : liste le pourcentage des livres empruntés ;
- `emprunt(Date date_emprunt, Livre &livre, Lecteur &lecteur)` : créé un emprunt ;
- `retour (Emprunt e, Lecteur &lecteur, Livre &livre)` : fait appel à la méthode retour de la classe Emprunt voir paragraphe précédent) et passe la disponibilité du livre à `true` ;
- `classement()` : devait nous retourner le classement des lecteurs. Malheureusement, nous avons eu un problème : J'ai un souci avec le classement des lecteurs : la variable publique `_n_ISBN_lus` me retourne le nombre de livres lus dans la classe Lecteur. Or, cette même variable ne retourne pas la même valeur une fois utilisée dans la classe Bibliothèque. Nous ne sommes pas parvenus à résoudre ce problème.

1.3 – Conclusion

Ce premier TP nous a été extrêmement instructif.
Nous avons solidifié les bases acquises en CM et en TD et nous nous sommes familiarisés avec les principes de la programmation en C++. Par exemple, l'idée d'utiliser tous ces vecteurs nous est venue naturellement, puisque nous savions que pour des listes devant plus ou moins s'étendre, ce type de tableau était ce qu'il fallait utiliser. Mais encore, alors que nous n'en avions qu'une légère idée, la communication entre les classes nous apparaît à présent d'une clarté bien plus expressive. De plus, nous avons compris l'importance des références, car en effet, sans la petite esperluette, les objets sont copiés et non pas déplacés comme nous le voulions. Notre seul regret sur ce TP est de ne pas avoir réussi à établir le classement des lecteurs. Et à vrai dire, si la solution pouvait nous être donnée, nous en serions ravis.
Nous avons utilisé un peu plus de trois séances pour en venir à bout, mais nous y sommes parvenus !

2. TP4 C++ : Alea Jacta Est

2.1 – Motivation et fonctionnement

Nous avons fait le choix pour le second TP du numéro 4, notamment grâce à son sujet. En effet, il nous offrait la possibilité d'en apprendre plus sur les différentes techniques de codage et de décodage de messages. Autrement dit, il nous a permis d'avoir une première approche de la cryptographie et d'en découvrir les bases.
Pour ce TP nous avons très peu répartie le travail : pour être sûr de bien cerner toutes les nuances de chaque façon d'encoder un message, nous avons préféré réfléchir à deux sur une grande partie du sujet. Et puis, certaines consignes n'étaient pas comprises de la même manière par les deux participants, il a bien fallu débattre de la décision finale que nous vous avons soumise.

2.2 – Organisation des classes

Comme attendu par le sujet, notre code comporte 5 classes. La première, la classe BasicEncrypt, puis la classe Encrypt et enfin les trois classes qui représentent chacune une façon d'encoder ou décoder un message : Caesar, Caesar2 et Vigenere.

A) BasicEncrypt

La classe première classe créée, BasicEncrypt,, sert de base pour tout le reste du TP. Elle n'est pas de la plus grande importance par rapport aux prochaines classes. Parlons-en fugacement.
Cette classe contient les variables :

- `_cypher`
- `_plain`

qui contiennent respectivement le message codé et le message décodé. Nous avons ensuite créé deux fonctions :

- `decode()` : écrit `_cypher` dans `_plain` ;
- `encode()` : écrit `_plain` dans `_cypher`.

B) Encrypt

La classe Encrypt, classe mère des trois prochaines, est abstraite et utilisée par le reste du TP. Elle est donc le modèle pour les trois classes suivantes. Elle contient deux fonctions virtuelles :

- `virtual void encode(int decalage) = 0` ;
- `virtual void decode(int decalage) = 0`.

Comme vous pouvez le voir, une fonction virtuelle est une fonction déclarée avec une valeur par défaut à 0. Elle ne n'est pas déclarée dans notre fichier Encrypt.cpp, mais sert simplement à annoncer sa nécessité dans les classes filles, ce qui obligera ces dernières à la déclarer.
De plus cette classe contient les fonctions :

- read() : sert à lire les éléments d'un fichier texte et à stocker dans _plain le contenu de ce fichier ;
- writ()e : sert à écrire le contenu de _cypher dans un fichier texte.

Ces deux fonctions se basent sur le même principe : elles testent si le fichier « fonctionne ». Si ce n'est le cas, la fonction s'arrête avec un message d'erreur. Sinon, soit nous extrayons le contenu, soit nous écrivons dedans, selon la fonction appelée. N'étant pas très intéressantes dans les classes filles, nous n'en parlerons qu'une fois.

C) Caesar

La classe Caesar permet d'aborder une première façon de coder et décoder un message. Cette façon est assez basique : nous choisissons un décalage et nous décalons chaque lettre du décalage choisis.
Cette classe réutilise, et ce comme les deux classes qui suivent, les fonctions encode, decode, read et write.
Cette classe réutilise, et ce comme les deux classes qui suivent, les fonctions encode, decode, read et write.

- encode(int decalage) override : teste si le caractère est bien en minuscule, si c'est le cas il est encodé sinon, il reste tel quel. De plus si le décalage choisis par l'utilisateur est négatif nous faisons le choix de prendre la valeur absolue de ce dernier.
- void decode(int decalage) override : fait la même opération que decode() mais en sens inverse ;
- void read(std::string fichier) : comme dit plus haut, sert à écrire dans un fichier si celui-ci est valide. Elle renvoie à la fonction éponyme de la classe Encrypt ;
- void write(std::string fichier) : sert à écrire dans un fichier si celui-ci est valide. Elle renvoie également à la même fonction de la classe mère.

D) Caesar2

La classe Caesar2 reprend ce principe mais en améliorant les mots choisis. Nous ne nous contentons plus de décaler simplement les lettres minuscules mais bien tous les caractères classiquement utilisés. Ainsi la seule différence de code avec la classe Caesar et le nombre de caractères pris en compte.

E) Vigenere

La classe Vigenere se base sur le codage de Vigenère, qui en prend en entrée dans un premier temps un vecteur de chiffre où chaque chiffre correspond à un décalage. Nous avons retranscrit cela avec deux variables :

- _clef : ce vecteur est la variable privée contenant les différents chiffres de la clef ;
- _taille : et sa taille sera conservée dans une variable _taille. Ainsi, si l'on a 4 éléments dans le vecteur, nous aurons un cycle de décalage de 4.

Viennent ensuite les méthodes, plus compliqués que les précédentes.

- `transformation(std::string alpha_clef)` : nous prenons une clef sous forme de string. Pour cela, nous transformons simplement le caractère du string en son caractère ASCII correspondant à l'aide d'une fonction Cette fonction met ensuite le nombre généré dans le vecteur `_clef` ;
- les fonctions `encode(int decalage)` et `decode(int decalage)` deviennent donc plus complexes. En premier lieu, nous vérifions que chaque int présent dans le vecteur `_clef` est bien compris entre 0 et 26, sinon nous le mettons dans cet intervalle. En second lieu, pour les deux fonctions, nous ôtons les espaces ou caractères spéciaux du message d'entrée et nous stockons les lettres de ce message dans un string joliment nommé `inupi`. Nous appliquons le décalage à chacun des caractères de `inupi`. Nous injectons ensuite les nouvelles lettres dans `_cypher` (ou `_plain` selon la fonction utilisée), en prenant en compte la forme du message d'entrée. Cela nous permet de ressortir un message avec la même ponctuation mais les lettres encodées (ou décodées).

2.3 – Conclusion

Ce TP nous a permis d'aborder différentes notions du C++ et de la Programmation Orienté Objet. Parmi les notions vues, une domine : l'héritage. Cet exercice nous permet de se familiariser avec ce pilier de la POO. Le fait d'utiliser une classe abstraite comme classe mère nous permet de voir que l'on peut utiliser les classes comme templates pour créer d'autres classes du même genre de façon de façon efficace. De plus ce TP nous permet de renforcer nos connaissances sur les containers, particulièrement les vecteurs qui ont été utile pendant tout notre travail.
L'importance de ces notions nous permet de conclure sur le fait qu'à l'avenir la pratique nous sera nécessaire. En effet même si nous avons utilisés ces notions nous ne sommes pas familiers avec elles pour autant et qu'un gain en efficacité ne peut être que bénéfique pour le futur.

3. Conclusion générale

Pour conclure ce compte-rendu, nous tenions à remercier le professeur Ginhac ainsi que tous les encadrants qui l'ont aidé à nous apprendre la programmation lors de ces séances. Nous avons appris énormément en peu de temps (quoiqu'en 24h tout de même, le temps passe vite quand on s'amuse). Les sujets étaient tous très attrayants, ceux que nous choisis ont été très distrayants en plus d'être parfaitement instructifs.