Compte rendu TP C++

Dutoit Arthur

Binet Arnaud

3A ESIREM IT

Introduction Générale

Le but de ces TP est d'apprendre à développer en C++, de développer ses connaissances dans ce langage tout en codant un programme qui pourrait être utile en situation réelle.

Premier TP

Introduction

Pour ce premier TP Arthur et moi avons choisi de prendre dans un premier le TP 1, donc le TP Bibliothèque. Dans un premier temps il faut réfléchir à tout ce qu'il faut coder, qu'est ce qui va être utile de coder, comment vont être codé ce que l'on souhaite, et quelles sont les liens entre les différentes classes.

Date

On commence par coder la classe Date qui va nous permettre d'avoir une date de début d'emprunt du livre et à la fois d'initialiser la date de naissance d'un Auteur ou bien d'un Lecteur. Pour cela il faut vérifier la date ce qui est fait dans isDate où o vérifie que le jour est compris entre 1 et 31/30/28/27, que le mois est compris entre 1 et 12 et que l'année est 2023 ou moins. Et on a ajouté une fonction toString qui permet de passer la date en string pour l'afficher avec des « / ». Ce n'était pas demandé mais pour l'affichage ce sera plus sympathique.

```
std::string toString(Date date)
{
    return std::to_string(date.Jour)+"/"+std::to_string(date.Mois) + "/" + std::to_string(date.Annee);
}
```

Livre

Ensuite on a fait la classe Livre qui va nous permettre de créer un livre, et donc par la suite de créer une bibliothèque. Pour cela il faut déjà crée une variable qui sache si le livre est disponible et toutes les informations nécessaires à un livre (Titre, Auteur, Genre, ISBN (numéro correspondant au livre)). Et comme d'habitude les fonction getter et setter qui seront updateTitre()et titre() par exemple. Pour cette classe on a aussi fait une surcharge d'opérateur « << » de la classe Livre comme ceci,

```cpp
friend std::ostream& operator<<(std::ostream& os, const Livre& livre);
```

```cpp
std::ostream& operator<<(std::ostream& os, const Livre& livre)
{
    os << "Titre: " << livre._titre << std::endl;
    os << "Auteur: " << livre._NomAuteur << std::endl;
    os << "Genre: " << livre._genre << std::endl;
    os << "Langue: " << livre._langue << std::endl;
    os << "ISBN: " << livre._ISBN << std::endl;
    os << "Disponible: " << livre._disponible << std::endl;
    return os;
}
```

Nous avons aussi fait une fonction qui permet de vérifier si le nom est bien composé de lettre, donc voilà comment nous l'avons codé :

```cpp
bool Livre::checkLangue(std::string langue)
{
    for (std::string::iterator i = langue.begin(); i != langue.end(); i++)
    {
        if (std::isalpha(*i))
            return true;
        else
            return false;
    }
}
```

Pour finir avec la classe Livre nous avons ajouté une classe Auteur qui est liée à la classe Livre et qui comporte le nom complet de l'auteur, sa date de naissance et son identifiant numérique.

Lecteur

Ensuite il y a la classe Lecteur qi permet de créer un lecteur avec son nom, son prénom, et son ID que l'on crée avec la première lettre du prénom et le nom de famille ensuite. Avec les fonctions getter et setter, la vérification que ce ne soit que des lettres et les espaces, et la surcharge d'opérateur qui fonctionne de la même manière que celui d'avant.

Emprunt

Ensuite on a la classe Emprunt qui va nous permettre de simuler un emprunt de livre, pour cela on inclut la classe Lecteur, Livre, et Date car pour créer un emprunt il va nous falloir un objet livre, un objet lecteur et un objet date. Pour faire un emprunt on aura besoin d'une date de début puis de fin c'est là ou va rentrer en jeu la classe date car pour rendre un livre on a décidé de prendre un temps de deux semaines et après ça on considère que le livre sera rendu.

```cpp
std::string Emprunt::dateEmpruntDebut(Date date)
{
    //Initialisation de la date
    Livre livre;
    date.Date::Date(date.Mois, date.Jour, date.Annee);
    livre.disponible = false;
    return toString((date.Mois, date.Jour, date.Annee));
}

std::string Emprunt::dateEmpruntFin(Date date)
{
    //Ajout de 2 semaines à la date de début d'emprunt pour finir l'emprunt
    jourFin = date.Jour + 14;
    if (date.isDate(date.Mois, date.Jour, date.Annee) == false)
    {
        jourFin = date.Jour - 30;
        moisFin = date.Mois + 1;
        if (date.isDate(date.Mois, date.Jour, date.Annee) == false)
        {
            moisFin = date.Mois - 12;
            anneeFin = date.Annee + 1;
        }
        return toString((moisFin, jourFin, anneeFin));
    }
    else return toString((date.Mois, jourFin, date.Annee));
}
```

Main

Ensuite dans le main.cpp, on crée les objets des classes Livre, Auteur et Date de cette manière,

```cpp
//Création des dates
Date d1(1, 1, 2020);
Date d2(1, 1, 2021);

//Création des Auteurs
Auteur a1("Auteur1","1/1/1",1);
Auteur a2("Auteur2","2/2/2",2);
Auteur a3("Auteur3","3/3/3",3);
Auteur a4("Auteur4","4/4/4",4);
Auteur a5("Auteur5","5/5/5",5);

//Création des Livres
Livre l1( a1, "Harry Potter", "Français", "Fantastique", 1);
Livre l2(a1, "Paela", "Français", "Fantastique", 2);
Livre l3(a2, "Tourte", "Français", "Fantastique", 3);
Livre l4(a2, "Entrecote", "Français", "Fantastique", 4);
Livre l5(a3, "Michigan", "Français", "Fantastique", 5);
Livre l6(a3, "Lost Vegas", "Français", "Fantastique", 6);
Livre l7(a4, "Gnagna", "Français", "Fantastique", 7);
Livre l8(a4, "Gnogno", "Français", "Fantastique", 8);
Livre l9(a5, "Tratra", "Français", "Fantastique", 9);
Livre l10(a5, "Troutrou", "Français", "Fantastique", 10);
```

Et pour appliquer les surcharges d'opérateur il faut faire comme ci-dessous, ce qui permettra d'afficher les informations des différents objets, comme ceci

```
//Surcharge des opérateurs de Auteur
std::cout << a1 << std::endl;
std::cout << a2 << std::endl;
std::cout << a3 << std::endl;
std::cout << a4 << std::endl;
std::cout << a5 << std::endl;
```

Ça fonctionne de la même manière pour chaque surcharge d'opérateur.

Bibliothèque

Et ensuite pour finir il y a la classe Bibliothèque qui va nous permettre de stocker tous les livres que l'on crée et donc après savoir si le livre est disponible, ou bien le nom du livre ou toutes autres informations qui sont initialisées, pour stocker ces livres, on a choisi un vecteur car c'est la solution qui permet de pouvoir stocker une infinité de livre.

Conclusion

Ce TP nous a permis de comprendre comment fonctionnent les classes avec les objets et comment utiliser les classes comme variable.

Deuxième TP

Introduction

Tout d'abord, le but de ce TP est de concevoir une application de gestion de réservations d'un hôtel.

Côté technique, ce TP permet de mieux comprendre la programmation orientée objets (POO) en créant plusieurs classes peu complexes afin de les faires interagir ensemble.

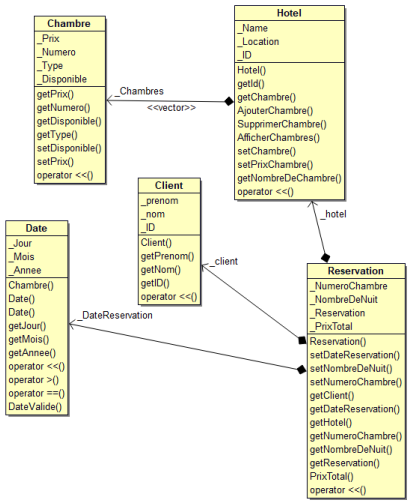
Nous avons choisi ce TP car nous nous sentons comme des débutants en C++ et nous ne pensions pas être capable de faire des TP plus durs.

Pour ce qui est du travail commun, nous avons surtout décidés de nous concentrer chacun sur un TP précis afin de mieux suivre l'avancée de notre TP et de mieux comprendre la relation entre nos

classes. Cependant nous avons fait deux TP se ressemblant fortement et nous avons donc pu nous aider et partager entre nous des méthodes utiles pour nos deux TP.

Nous avons utilisé GitHub pour ce qui est du versionning et pour pouvoir stocker nos codes dans un même environnement. Nous nous sommes aussi aidés de GitHub Copilot mais en nous efforçant de comprendre le code généré par l'outil et comme dans certains cas Copilot générait du code qui ne nous satisfaisait pas nous nous sommes seulement servis des bases générées et nous avons modifiés le code pour qu'il fasse ce que nous voulions.

Tout d'abord, voici la représentation UML de notre projet, cet UML a été généré à l'aide du logiciel BOUML qui permet la génération d'UML à partir d'un code source.



Nous avons tentés de modéliser cet UML dès le début du projet pour pouvoir poser notre code mais comme mentionné dans les commits sur GitHub nous avons rencontré plusieurs problèmes au moment de traduire cet UML en code, en effet notre UML n'était pas correcte et ne nous permettait donc pas de répondre à tous les besoins de ce TP.

Passons maintenant aux particularités de ce code :

Date

```cpp
bool Date::DateValide(int Jour, int Mois, int Annee)
{
    int t[4] = { 4, 6, 9, 11 };
    int *mois = std::find(std::begin(t), std::end(t), Mois);
    if (Annee < 0)
    {
        return false;
    }
    if (Mois < 1 || Mois > 12)
    {
        return false;
    }
    if (Jour < 1 || Jour > 31)
    {
        return false;
    }
    if (Mois == 2 && Jour > 29)
    {
        return false;
    }
    if (mois == std::end(t) && Jour > 30)
    {
        return false;
    }
    return true;
}
```

La fonction ci-dessus permet de vérifier la validité de la date entrée par l'utilisateur, nous avons trouvé intéressant d'utiliser un tableau afin de stocker les mois contenant 30 jours. La fonction « find » de la bibliothèque « algorithm » permet de rechercher si la valeur de la variable mois est comprise dans le tableau. Si la valeur de 'mois' est dans le tableau, on vérifie que le nombre de jour est bien inférieur a 30, si l'utilisateur a entré « 31 » alors la fonction renvoie une erreur.

Fonctions

Une autre particularité de notre code est l'utilisation du type auto.

```cpp
void AfficherReservationEtID(vector<Reservation>& reservations, unsigned int Reservation)
{
    for (auto reservation : reservations)
    {
        if (reservation.getReservation() == Reservation)
        {
            cout << reservation << endl;
        }
    }
    cout << "Aucune reservation trouvee" << endl;
}
```

Tout d'abord proposé par Copilot, nous somme allés nous renseigner sur le fonctionnement de ce type. Nous avons compris qu'il permet de choisir automatiquement le type du paramètre spécifié par rapport au type de la condition. Ici il s'agit d'un vector et permet ici à la fonction de vérifier si la réservation passée en paramètre de la fonction par l'utilisateur existe dans le vector réservation.

Classes

Nous allons maintenant faire un bref résumé de chaque classe :

Classe Chambre : Permet de créer un objet chambre et de définir ses paramètres (numéro, disponibilité, prix)

Classe Client : Permet de créer un objet chambre et de définir ses paramètres (Nom, Prénom, ID)

Classe Date : Permet de créer un objet date modifiable par l'utilisateur et de définir ses paramètres (Jour, Mois, Année)

Classe Fonctions : C'est dans cette classe que sont définies toutes les fonctions permettant d'agir directement sur la création et sur la modification d'objets dans ce projet.

Classe Hotel : Permet de créer un objet hôtel qui hérite des propriétés d'un objet chambre et de définir ses paramètres (Nom, localisation, ID)

Classe Reservation : Permet de créer un objet réservation qui hérite des propriétés d'un objet hôtel et de définir ses paramètres (Date de al réservation, hotel, client, Numéro de chambre, Nombre de nuits, ID de la réservation)

Conclusion

Ce TP nous a permis d'apprendre à manipuler des classes en C++, il nous a aussi permis d'apprendre à mieux utiliser les surcharges d'opérateurs et de faire un code plus rigoureux. Nous avons aussi décidé de mettre en place une modélisation UML, bien que nous ayons échoués la première fois, nous avons décidés de persévérer et de trouver un moyen de vérifier notre UML. Nous pensons que la modélisation UML est une partie importante du développement donc nous avons décider de ne pas l'oublier.

Conclusion Générale

En conclusion, ces deux TP nous ont permis d'apprendre à manipuler des objets en C++ et donc de mettre un pied dans la programmation orientée objet. Nous pensons aussi que ces TP nous ont permis de nous rendre compte de la rigueur nécessaire dans la programmation si l'on veut être rigoureux.