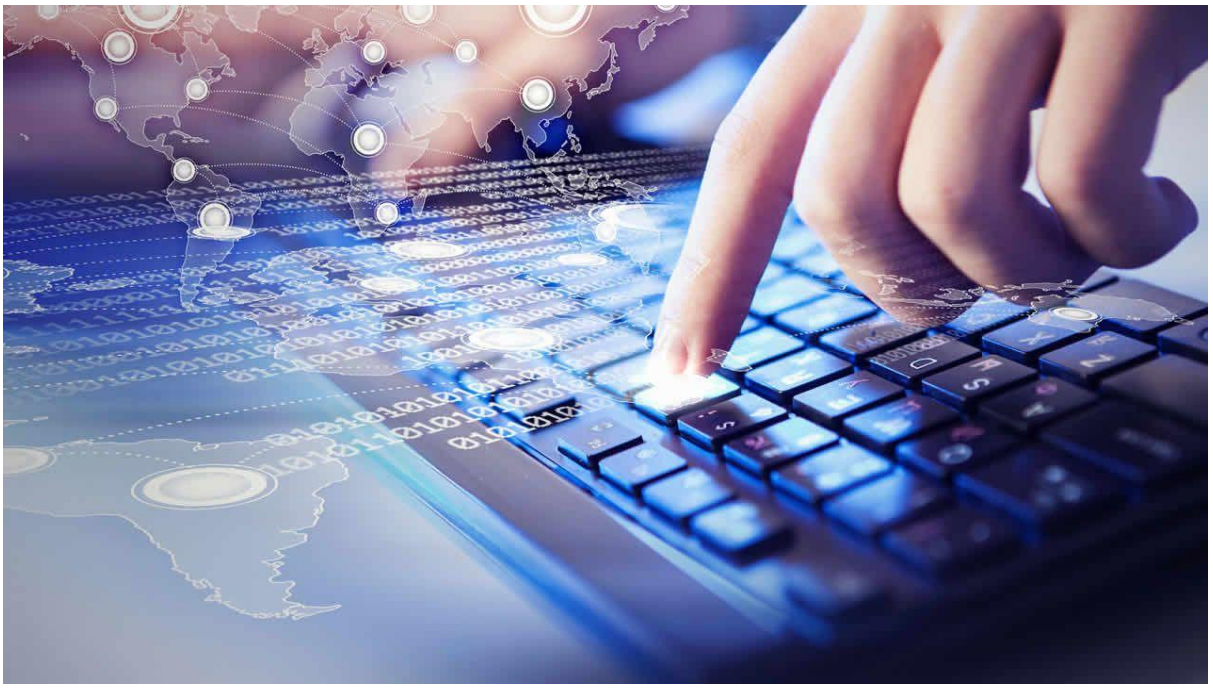
Compte Rendu TP de C++



M. Ginhac Dominique

ROB 3A

Baptiste Lesquerré

1)Introduction

Au cours des six séances de TP, nous avons décidé de réaliser les deux premiers TP de difficulté facile. Nous avions convenu qu'il serait plus sage de commencer par ces TP plutôt que de se lancer dans un TP trop complexe et de ne pas comprendre ni pouvoir finir celui-ci. Pour la répartition du travail nous avions commencé par travailler sur le TP2 en nous répartissant les questions. Nous avons ensuite décidé de prendre chacun un TP pour avancer plus rapidement dans leur exécution. Elie Bienvenue s'est occupé du premier TP et Baptiste Lesquerré à continuer la réalisation du deuxième. Même en travaillant sur deux TP différents, nous nous concertions régulièrement à propos d'un TP ou de l'autre sur différent choix à réaliser ou pour résoudre différents problèmes.

2) TP2

Au cours de ce TP l'objectif était de réaliser plusieurs classes et de les compléter au fil de celui-ci en créant les méthodes et les fonctions nécessaires de manière à répondre aux attentes de l'énoncé. Pour celà il a été nécessaire de créer des getters, setters, helpers, surcharges d'opérateurs et itérateurs tout en manipulant des vecteurs.
Le but de ce TP était de créer des classes et des programmes permettant la gestion des réservations d'un hôtel.

Création des classes (parti 1):

Les premières questions du TP ont conduit à la création de plusieurs classes : Client, Hotel, Chambre, Date, Reservation. La classe Date a été créée en utilisant la classe date précédemment créé comme exemple dans le cours. Les classes créées à ce point sont les plus minimales possibles. Chaque classe comporte au moins un constructeur, des variables et des getters pour chaque variable. On crée dans certaines classes des setters pour certaines variables s'il seront utiles pour les futures questions(aurions nous besoin de changer ces variables dans le futur?).
Au cours de la création de chaque classe nous avons commencé par créer chacune dans son propre namespace, seulement pour réaliser la difficulté apportée notamment lors de l'appel des méthodes. Nous avons donc décidé de créer un unique namespace "hotel" contenant les classes hotel et chambre, celà facilitant l'interaction entre les deux classes. En effet un objet hotel comporte un tableau de chambre.
Dans différentes classes des variables identifiants sont créées. On les définit comme des chaînes de caractère de manière à ce que l'utilisateur puisse écrire ceux-ci comme bon lui semble.
On note que pour la classe Reservation on rajoute la variable "_id_reservation" pour l'identifiant de la réservation. Cette variable sera utile ensuite pour identifier les différentes réservations et les distinguer sachant qu'un client peut réserver plus d'une fois le même type de chambre sur le même créneau.
Certaines classes sont appelé dans d'autre classes (#include "date.h" dans reservation.cpp par exemple). On écrit donc les lignes #ifndef … , #define …, #endif … dans le fichier .h (date.h ici) de manière à ce que la classe ne soit appelée qu'une seule fois lors de l'exécution du programme.
Chaque classe est créée avec un fichier .ccp et un fichier .h. On crée également un fichier main.cpp dans lequel l'utilisateur pourra interagir avec les différents objets créés.
Après avoir écrit chaque classe, on teste le bon fonctionnement de celle-ci (de ces différentes méthodes) grâce à un programme test écrit dans le fichier main.cpp. Nous nous heurtons cependant à certains problèmes notamment au niveau des classes Hotel Chambre et Client . En effet, pour vérifier le bon fonctionnement des setters on se rend compte qu'il serait important de pouvoir afficher les objets clients mais également chambre (un hôtel possédant un vecteur de chambre) pour ainsi mieux vérifier les modifications apportées à ces objets. Ceci entame la seconde partie du TP.

Création de nouveaux helpers et setters (partie 2):

La question 6 débute par la création d'un hôtel (h1) qui servira pour toute la suite du TP. Cet hôtel contient dix chambres réparties en trois types différents (single, double, suite). On utilise une méthode "chambre_creator" qui permet d'ajouter plusieurs chambre du même

type (même prix) avec des identifiants différents, a la variable “_tabchambre” de l'hôtel crée.

Toute interaction du client se fera sur l'hôtel ou sur un de ses objets. Comme remarqué précédemment il est nécessaire d’afficher les objets créés. L'énoncé nous indique alors de réaliser une surcharge d'opérateur “<<” pour les classes Client, Hotel et Chambre. Pour chacune on commence par créer une méthode “tostring” créant une chaine de caractère figurant les informations d’un objet. La surcharge d’opérateur utilise ensuite cette fonction et affiche ainsi la chaîne de caractère créée. Ceci est très utile rendant visible les informations d’un objet lorsque l’on utilise l’opérateur <<. On réalise ainsi un meilleur contrôle des modifications. On crée également une fonction display pour Client et Chambre. Ces fonctions prennent en paramètre un vecteur (de clients/ de chambre) et utilisent une fonction “tostring” retournant une chaîne de caractère correspondant aux éléments du tableau. On peut ensuite afficher les éléments de ces vecteur de manière ordonnée. On peut ainsi régulièrement surveiller les changements dans ces vecteurs notamment dans le vecteur de chambre l’un des éléments d’un objet hotel.
Un premier exemple d’utilisation de ces fonctions est avec un tableau de client initialisé dans le fichier main.cpp. On crée quatre clients et on les ajoute dans le tableau. On utilise ensuite la surcharge d'opérateur ou la fonction display (couplé avec un “cout”) pour afficher les informations de chaque client. On vérifie ensuite s'il y a eu des changements dans les informations saisies.

Dans la question 7, on débute par la création d’une code permettant d’afficher les informations d’un client et/ou d’ajouter celui-ci au tableau de client. Le programme demande à l'utilisateur de saisir le nom du client. On parcourt ensuite le tableau comparant le nom à celui de chaque client enregistré. S’il figure dans le tableau, on affiche les informations du client au moyen de la surcharge d'opérateur et on affecte à un objet string “id” l’identifiant du client (sera nécessaire pour la suite). Si le client n’est pas dans la liste, l’utilisateur remplit les informations du client. On affiche ensuite celle-ci et “id” est affecté du nouvel identifiant. Après l’écriture du code on transforme celui-ci en une fonction helper “saisi_client” de la classe client retournant un objet type string (l’identifiant) et agissant sur un vecteur d’objet Clients. Celà facilite la lisibilité et permet d'exécuter toutes ces différentes étapes en une seule ligne dans le fichier main.cpp.

Au vu des questions suivantes on décide d’écrire un code qui sera également transformé en fonction helper (de la classe Reservation) permettant de saisir une réservation tout en vérifiant la validité de celle-ci : “saisi_reservation”. Pour que la réservation soit valide il faut que le créneau (date) soit disponible et que le type de chambre souhaité soit libre pour ce créneau. On commence par initialiser un tableau de Chambre “chambre_dispo” qui contiendra les chambres disponibles. L’utilisateur saisi la date et le nombre de nuit souhaité, la fonction fait appelle a la fonction "creneau_valide" qui vérifie l’existence de créneau déjà réservé chevauchant le créneau désiré. S’il n’y a pas chevauchement on ajoute la chambre concernée dans le tableau “chambre_dispo” qui sera ensuite complété par les chambres sans réservations sur le créneau (fonction “chambre_libre”). Si le créneau est non disponible, l'utilisateur est invité à en saisir un nouveau. Après cela, une fonction “search_chambre” va vérifier si le type de chambre désiré est présent dans “chambre_dispo”. Si ce n’est pas le cas, l'utilisateur est invité à choisir un nouveau type de chambre. En débordant sur la question 8 on crée une fonction qui calcule le montant de la réservation (dans “search_chambre” si le type de chambre est disponible) et l’affecte à la

réservation. Initialement dans “saisi_reservation”, une réservation par défaut a été créée et sera modifiée au fur et à mesure. Lorsque la réservation est terminée, on affiche celle-ci et le montant pour confirmer sa bonne saisie. Pour cela nous avons donc créé une fonction “tostring” et une surcharge d’opérateur “<<” dans la classe Reservation. Finalement on ajoute cette réservation a un tableau de réservation pour conserver celle-ci. Ce tableau est initialisé dans le fichier main.cpp grâce au clients précédemment créé. On remarque cependant qu’il aurait été plus judicieux lors d’une impossibilité dans la réservation, de demander à l'utilisateur s’il souhaite saisir celle-ci ou abandonner la saisie.

Une fois cette étape terminée nous testons différents cas de figure de réservation, différents problèmes de réservation pour ainsi voir si le programme réagit correctement. Malheureusement ce n’est pas le cas et nous nous heurtons a beaucoup de problèmes différents notamment au niveau de la vérification de la validité du créneau et de la disponibilité du type de chambre. En effet plusieurs chambres peuvent être réservées sur le même créneau, certaines chambres semble ne pas pouvoir être disponible, les créneau se chevauchant renvoi une indisponibilité alors que d’autres chambres sont libres…
Pour résoudre ces problèmes, on réalise des débogages manuels et nous utilisons un débogueur dans le but d’identifier les sources de ces problèmes. Certains problèmes sont rapidement résolus, étant dû à une faute de frappe ou une étourderie. D’autres erreurs sont en revanche plus complexes à résoudre et nécessitent une bonne compréhension du problème. On peut citer par exemple pour la fonction “creneau_valide” qu’il était essentiel de bien comprendre ce qui définissait un “créneau valide” par rapport aux autres créneaux sélectionnés dans les réservations. Certains problèmes plus subtiles ne sont découverts que lors d’une manipulation spécifique du programme (problème de deux réservations réalisées sur des créneaux proches par exemple).

On aborde ensuite la question 9. Pour commencer il faut créer une fonction permettant d’afficher toutes les réservations. On procède alors comme avec les classes Client et Chambre en utilisant la fonction “to_string” dans une fonction “display” qui prend en argument un vecteur de réservation (ici se sera le vecteur de toutes les réservations) et affiche ensuite toutes les réservations en utilisant l’opérateur “<<”. On crée ensuite deux autres fonctions qui vont permettre d’afficher une réservation selon les informations mis en paramètre. “display_reservation” affiche une réservation selon le numero de reservation donné. “display_reservation_client” prend en paramètre le tableau des réservations et celui des clients, et demande à l’utilisateur soit un identifiant soit le nom d’un client. La fonction affichera ensuite les réservations réalisées par ce client. Cette fonction est plus difficile à réaliser et celle-ci utilise une autre sous fonction “display_reservation_clientid” qui détecte si l’objet passé en paramètre est bien un identifiant et affiche dans ce cas la réservation. Cette fonction renvoie un booléen a la fonction “display_reservation_client” lui confirmant s’il s’agit bien d’un identifiant ou pas. Si un nom à été saisi, la fonction recherche l’identifiant correspondant et affiche les réservations (“display_reservation_clientid”).

Avant de continuer dans la progression du TP, on se penche de nouveau sur la génération de réservation. On décide alors pour simplifier et pour mieux arranger le code, de réaliser la création de réservation sous forme d’une d’une fonction helper de la classe réservation : “reservation_creator”. Cette fonction prend en paramètre un hôtel et un vecteur de client. Elle initialise un vecteur de réservation (le tableau des réservations), puis grâce à une boucle while elle enregistre les clients et leurs réservations (dans les deux vecteurs). On

utilise “saisi_client” et “saisi_reservation”. La boucle while demande a chaque itération si l’utilisateur veut continuer à saisir des réservations. S’il refuse (toute autre réponse que “yes”) la boucle s’arrête. Après chaque saisie de réservation, le tableau est affiché.
Suite à certains problèmes liés à un tableau de réservation initialement vide, on initialise celui-ci par une réservation 0 d’une salle ne pouvant être réservée par les clients et n’étant pas affichée à chaque affichage du tableau des réservations.

En suivant les dernières questions du TP on crée ensuite les fonctions “modif_reservation” et “delete_reservation”. On a précédemment divisé la fonction “saisi_réservation” en “saisi_reservation1” et “saisi_reservation2”. Cela nous permet de réutiliser la fonction “saisi_reservation1” dans “modif_reservation” après la saisie de l’identifiant de la réservation à modifier. On évite ainsi de réécrire du code. Pour la fonction “delete_reservation” on va utiliser un itérateur. Le tableau des réservations est placé en paramètre et après avoir récupéré l’identifiant de la réservation à supprimer, on parcourt le tableau grâce à l'itérateur. Celui-ci étant un pointeur sur les cases mémoires du tableau (successivement) on peut ainsi (utilisant “erase”) facilement supprimer la case mémoire de la réservation désirée.

On modifie plusieurs fois le fichier main.cpp pour tester les différentes fonctions récemment implémenter (certaines sont ajoutées à “reservation_creator”). Ce sont les fonctions “display_reservation_client”, “delete_reservation” et “modif_reservation” qui posent le plus de problème. Pour “modif_reservation” il faut pouvoir modifier la réservation sans créer une seconde réservation (conservation des versions modifiées et non modifiées). Pour “display_reservation_client” le programme doit fonctionner également lorsqu'on saisit le nom d’un client. Pour “delete_reservation” la bonne compréhension de la manipulation d’un itérateur est crucial pour bien parcourir le tableau et ainsi supprimer la bonne réservation. Après avoir résolu ces différents problèmes et s’être assuré que le code fonctionne bien, on réalise une dernière modification dans “reservation_creator”. On sort la création du tableau de réservation de la fonction (ajoutée dans le fichier main.cpp). Dans le cas contraire, le tableau n’existerait qu’au sein de la fonction rendant sa manipulation difficile en dehors de celle-ci.

Conclusion et remarques sur le TP

Ce TP a consisté en une expérience constructive de programmation c++. La réalisation des classes et leurs évolutions au cours du TP a permis de mieux comprendre comment choisir et privilégier certaines méthodes et helper pour parvenir aux fins désirés. Il est important également de réaliser quelles parties du code peuvent être réutilisées pour ainsi éviter d’écrire du code redondant. Ce TP a permis également de bien prendre conscience pourquoi il est plus avisé de rédiger quelques notes sur une feuille que de s’acharner sur le code face à un problème. Ce TP à également apporté certains savoir-faire, comme la meilleure compréhension d’un itérateur, d’une surcharge d’opérateur “<<” et de comment créer des fonctions type “tostring” nécessaire pour afficher des objets créés. Au niveau des choses à changer ou améliorer, il serait important de mieux ranger et arranger le code des différents fichiers pour qu’il soit présentable et plus lisible (celà aurait dû être pris en compte dès le début du TP). Il faudrait également faire attention aux pertes de temps. En effet certains problèmes était très long à résoudre, et il serait donc intéressant de s’améliorer sur ce front pour ainsi gagner en efficacité et en vitesse. Pour continuer à progresser il faut donc continuer à s'entraîner tout en faisant attention à ces détails.