Projet C++ : Le monde d'après

Lorette DAUSSY et Daran NEZET

Le thème de notre projet est : Le monde d'après. Pour nous, le monde d'après (le covid) comme nous l'imaginons et le souhaitons ne serait pas un monde de zombies, d'extraterrestres, de vaccins ou de distanciation mais un monde qui ressemblerait à celui d'avant. Pour cette raison, nous avons décidé de créer une application permettant de gérer un bar.

1. Description de l'application

a) Fonctionnalités

Notre bar est composé de 22 tables, 5 serveurs et 6 cuisiniers/barmans.

Notre application est une application que l'on mettrait sur une tablette donnée à un groupe de client en arrivant. Cela leur permettrait d'obtenir un numéro de table, d'avoir la carte, de passer leurs commandes et d'obtenir l'addition. Lorsqu'ils partent cette tablette serait donnée à un nouveau groupe. A la fin de la soirée, l'application afficherait la liste des clients ainsi que le prix qu'ils ont payé et également le stock restant.

Afin de simuler le fonctionnement de plusieurs tablettes en réseau pour gérer tout le bar nous attribuons des tables aléatoirement afin que notre application voit un bar qui est partiellement occupé. A chaque nouveau groupe de clients qui prend la tablette, l'occupation du bar change.

Au lancement de l'application, les serveurs et cuisiniers qui travaillent sont affichés avec leur contrat.

Quand un groupe de clients arrive, il peut s'enregistrer et une table libre ayant le bon nombre de places lui est attribuée. Nous avons pour cela créé un algorithme permettant de parcourir toutes les tables libres et de déterminer quelle table a une capacité la plus proche possible du nombre de personnes du groupe. Cela nous permet de ne pas mettre un groupe de 3 clients à une table de 8 par exemple, car cela ferait perdre de la place inutilement.
Si il n'y a aucune table suffisamment grande disponible, nous demandons au groupe de revenir plus tard et donc de passer la tablette au groupe suivant. En effet, lorsque ce groupe aura fini de manger et partira, l'occupation du bar aura changé.

Une fois que le groupe est assis à sa table, la carte va ensuite s'afficher avec les prix. Le groupe va pouvoir faire des commandes. Sa commande complète et le prix auquel elle correspond s'affiche au fur et à mesure de la commande. Le groupe quitte ensuite la partie commande quand il a fini et veut maintenant payer.

Le groupe paye la commande puis restitue la tablette à l'entrée. Un serveur choisit si la tablette est donnée à un nouveau groupe ou si la soirée est finie.

Quand le serveur indique que la soirée est finie, l'application affiche les clients ayant utilisé cette tablette et leur addition ainsi que le stock restant.

b) Les classes

Notre projet est composé de huit classes : une classe Bar, une classe Personne, une classe Client, une classe Personnel, une classe Serveur, une classe Cuisinier, une classe Table et une classe Stock. Nous avons 3 niveaux de hiérarchies : Une Personne peut être un Client ou faire partie du Personnel et un membre du Personnel peut lui-même être Serveur ou Cuisinier. Les classes Personne et Personnel sont abstraites car nous ne pouvons pas instancier directement une personne ou un membre du personnel.

La classe Personne a un attribut nom et un attribut prénom. Elle possède deux méthodes qui sont des accesseurs permettant d'obtenir le nom et le prénom de la personne. Elle possède aussi une méthode virtuelle pure permettant d'obtenir toutes les informations nécessaires sur la personne, cette méthode sera définie dans les classes filles.

La classe Client a plusieurs attributs, en plus de ceux hérités de la classe Personne : le nombre de personnes du groupe auquel il appartient, le prix que le groupe doit payer, une map avec tout ce que le groupe a commandé et la table du groupe. Nous avons une méthode permettant de donner un nom, un prénom et un nombre de personnes au client. Cette fonction est utilisée dans le main avec un cin pour que le client tape ses informations au clavier.

La classe Personnel est une classe abstraite. Elle possède un attribut regime qui correspond au nombre d'heures de travail par semaine de l'employé. Il y a également un attribut étant un vector de tables comprenant toutes les tables dont cette personne s'occupe, et un attribut correspondant au nombre de tables.

La classe Serveur permet de définir la fonction sePresenter(), avec des phrases personnalisées affichées de manière aléatoire.

La classe Cuisinier fait de même.

La classe Table a un attribut serveur, un attribut cuisinier, un attribut état (0 si elle est libre et 1 si elle est occupée) et enfin le nombre de places de cette table.

La classe Stock contient deux map, l'une permettant de gérer le stock et l'autre d'associer les produits à leur prix. Les méthodes de cette classe permettent de créer un stock puis d'y ajouter des produits. Nous avons aussi des méthodes pour afficher la carte et passer des commandes (diminuer le stock).

La classe Bar contient des vector : un vector de tables, un vector de serveurs, un vector de cuisiniers et un vector de clients. On y stocke donc toutes les tables du bar, tous les serveurs et les cuisiniers qui y travaillent en ce moment et enfin les clients qui vont utiliser cette tablette. Nous avons des méthodes pour associer les tables aux clients,

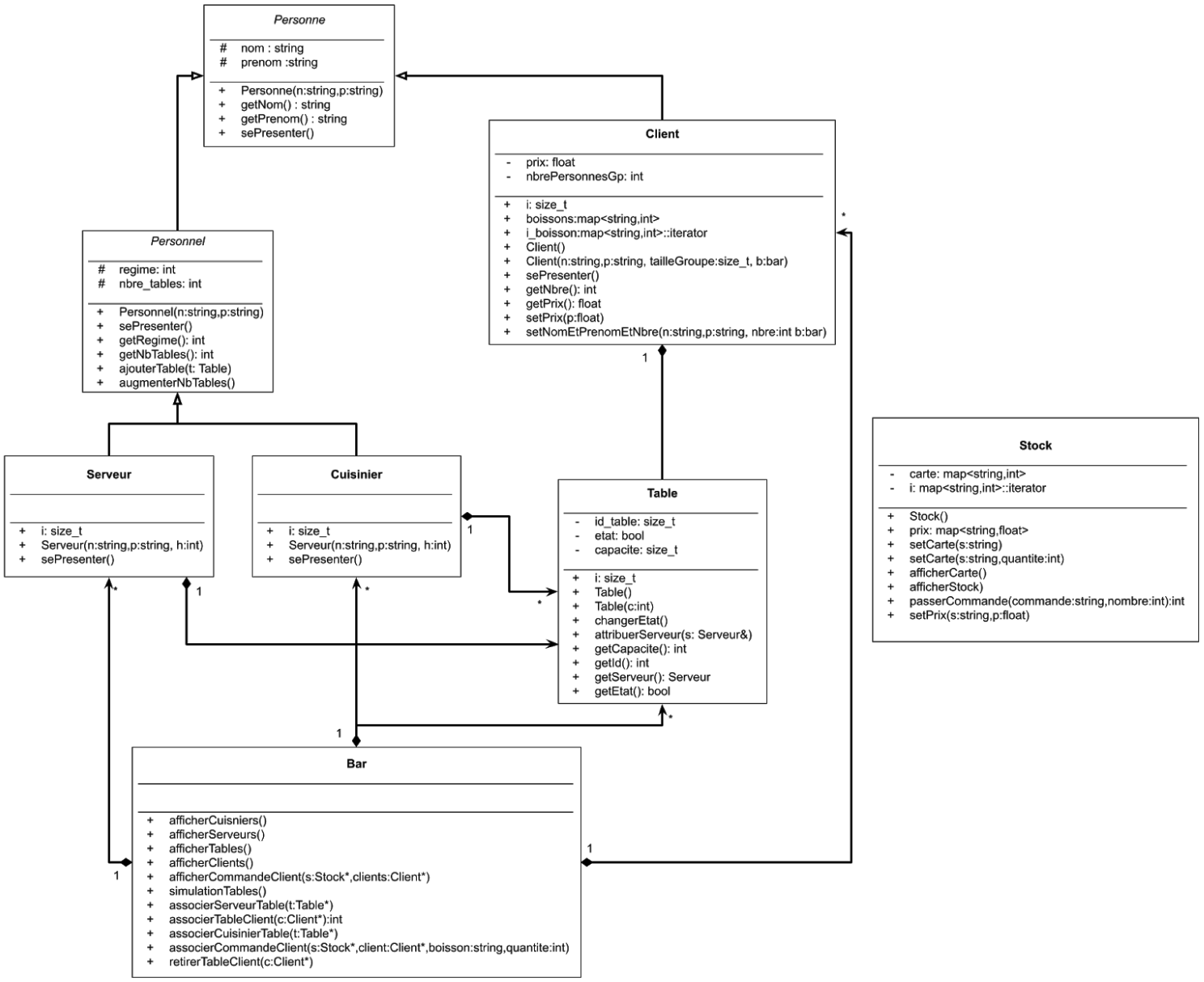
serveurs et cuisiniers et les commandes aux clients. Nous avons aussi des fonctions d'affichage et enfin la fonction qui permet de simuler le remplissage du bar.

Et enfin dans le main nous créons un bar puis des tables, serveurs, clients et cuisiniers que nous ajoutons aux vector du bar. Nous remplissons également le stock et créons la classe en associant des prix aux produits.
Nous avons un premier affichage au démarrage, nous affichons les serveurs et les cuisiniers.
Nous entrons ensuite dans une boucle while qui va permettre de gérer les clients pendant toute une soirée. Nous affichons les tables libres et occupées. Nous entrons ensuite dans une boucle jusqu'à ce qu'un groupe de client se voit attribuer une table.
La carte est ensuite affichée et le groupe peut passer autant de commandes qu'il le désire. Des messages apparaissent si les clients essayent de commander un produit qui n'est pas en stock ou une quantité qui dépasse le stock. Au fur et à mesure des commandes le groupe sait ce qu'il a commandé et le montant de l'addition. Lorsque les clients ont fini de consommer et veulent l'addition ils doivent taper "Q 0".
Le groupe doit ensuite payer au comptoir et rendre la tablette à un serveur. Le serveur indique si la soirée est finie ou si la tablette est donnée à un nouveau groupe.
A la fin de la soirée, la liste des clients ayant utilisé la tablette et le stock restant sont affichés.
Nous gérons tous les affichages grâce à des cin et des cout dans des boucles while. Nous utilisons std::system("clear"); afin de mettre à jour le terminal régulièrement pour avoir un affichage clair pour l'utilisateur.

Nous avons évidemment des constructeurs et destructeurs pour chaque classe.

c) diagramme UML

Voici le diagramme UML de notre projet :



d) La partie dont nous sommes le plus fiers

Nous avons créé deux fonctions dont nous sommes assez fiers. La première permet de trouver la table la plus adaptée à un groupe de personnes. C'est la fonction associerTableClient(Client *c) de la classe Bar. Il y a également la fonction permettant de simuler aléatoirement l'occupation du bar. C'est la fonction simulationTables() de la classe Bar.

2. Respect des contraintes

Nous utilisons des conteneurs de la STL : des tableaux dynamiques (vector) dans la classe Bar pour gérer toutes les tables, clients, serveurs... et des tables associatives (map) dans la classe Stock pour gérer le stock et la carte et dans la classe Client pour gérer sa commande. Le conteneur map est très pratique pour notre utilisation car cela permet d'accéder aux éléments grâce à leur nom et que si on ajoute un élément correspondant à une clé existante cela incrémente sa valeur et si cette clé n'existe pas cela la crée.

Nous utilisons aussi des fonctions virtuelles pures comme sePresenter() dans la classe Personne. Cette fonction étant différente dans chaque classe fille, elle est donc virtuelle pure dans la classe mère et également dans la classe Personnel. Elle peut donc être définie dans les classes filles en prenant en compte le type de personne confirmé. Ainsi Personne et Personnel sont des classes abstraites. Ainsi nous utilisons cette fonction dans la classe Bar pour afficher les Serveurs, les Cuisiniers et les Clients en affichant leurs caractéristiques propres.

Nous utilisons également un opérateur de surcharge pour afficher les tables. La fonction operator<< permet de pouvoir afficher une table directement de cette manière : cout << table_1; . Nous l'utilisons donc dans notre classe bar lorsque l'on doit afficher toutes les tables. Nous pouvons donc faire une simple boucle pour afficher toutes les tables de notre vector Tables.

Nous utilisons Valgrind qui nous permet de vérifier les accès en lecture et en écriture, de contrôler les fuites de mémoire et de vérifier que nous n'utilisons pas de variable non initialisée :

```
==19173==
==19173== HEAP SUMMARY:
==19173==     in use at exit: 0 bytes in 0 blocks
==19173==   total heap usage: 64 allocs, 64 frees, 90,240 bytes allocated
==19173==
==19173== All heap blocks were freed -- no leaks are possible
==19173==
==19173== For lists of detected and suppressed errors, rerun with: -s
==19173== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3. Utilisation de l'application

Pour commencer, télécharger la bibliothèque toilet grâce à la commande : sudo apt install toilet.

Nous avons un Makefile avec une règle all et une règle clean. Vous pouvez utiliser ce Makefile pour compiler notre code avec “make”. Vous pouvez ensuite lancer l’application en tapant “./main” dans le terminal. Il suffit ensuite de suivre au fur et à mesure les instructions sur le terminal.

Pour accéder à notre dépôt Github, vous pouvez vous rendre ici (https://github.com/Tapavinbal/Le_Monde_Dapres).