INF1010

Programmation Orientée-Objet

Travail pratique #1
Allocation dynamique, composition et agrégation

Objectifs : Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation.

Remise du travail : Lundi 4 février, 8h00 (AM).

Références : Notes de cours sur Moodle & Chapitre 2-7 du livre Big C++ 2e éd.

Documents à remettre : Tous les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.

Directives : Directives de remise des Travaux pratiques sur Moodle
Les en-têtes (fichiers, fonctions) et les commentaires sont **obligatoires**.
Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. **Pas de remise possible sans être dans un groupe .**
Veuillez suivre le guide de codage

La directive de précompilation « #ifndef »

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'entête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entrainerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H
#define NOMCLASSE_H
// Définir la classe NomClasse ici
#endif
```

La directive de précompilation « #include»

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

Mise en contexte

Le restaurant PolyFood vous contacte pour le mandat suivant : développer un programme permettant la gestion de leur restaurant. Vous décidez alors de mettre à profit les connaissances que vous allez acquérir tout au long de la session en suivant le cours de programmation orientée objet.

Travail à réaliser

Les fichiers plat.h, menu.h, table.h et restaurant.h sont fournis. On vous demande de compléter les fichiers *.cpp ainsi que de compléter le programme principal en suivant les directives du fichier main.cpp. Il vous faudra également compléter les fichiers *.h au besoin (ajout de mots clés et/ou destructeur).

Attention : un fichier PolyFood.txt vous sera fourni. Ce fichier représente un restaurant, il contient le menu et les tables. Vous pouvez trouver à la fin de document, le format du fichier. **Ce format ne peut être changé.**

Classe Plat

Cette classe caractérise un plat par son nom, son prix et le prix de production. Par exemple un paquet de frites pourrait être vendu à 10$, mais il ne coûterait que 3$ au restaurant pour le produire.

Cette classe contient les attributs privés suivants :

- nom_ (chaine de caractères, string)
- prix_ (double)
- cout_ (double)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut : le nom à inconnu, le prix et le cout à 0.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- Les méthodes d'accès aux attributs.
- Les méthodes de modification.
- Une méthode d'affichage.

Classe Menu

Cette classe caractérise un menu, ainsi il contiendra un tableau dynamique contenant des pointeurs vers des plats ainsi qu'un attribut issue d'un type énuméré (fourni dans le fichier header) permettant de déterminer le type de menu (Matin, Midi ou Soir).

Elle contient les attributs privés suivants :

- capacite_, qui représente la taille du tableau listePlats _ (entier non signé).
- listePlats_, tableau dynamique contenant des pointeurs vers des plats.
- nbPlats_, (entier non signé)
- type_, le type de menu issu de l'énumération TypeMenu

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut (dont une taille de tableau de `MAXPLAT` pointeurs à des plats).
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modification.
- Une méthode permettant de trouver un plat en passant le nom du plat, elle retourne un pointeur vers le plat en question (pointeur nul si le plat n'est pas trouvé).
- Deux méthodes d'ajout de plat : une utilisant un plat, et l'autre le nom, le prix et le cout du plat.
- Une méthode lireMenu() prenant en paramètre le nom du fichier contenant le menu (PolyFood.txt dans notre cas), cette méthode permet de remplir le menu. Attention : la lecture du fichier doit prendre en compte le type de menu (Matin, Midi ou Soir).
- Une méthode d'affichage.

Classe Table

Cette classe caractérise une table du restaurant, elle a un identifiant, une liste de plats (la commande de la table), un nombre de places et un attribut permettant de déterminer si la table est libre ou non.

Elle contient les attributs privés suivants :

- capacite_, qui représente la taille du tableau commande_ (entier non signé).
- commande_, tableau dynamique contenant des pointeurs vers des plats commandés.
- nbPlats_, (entier non signé)
- id_, (int)
- nbPlaces_, (entier non signé)
- occupee_, (bool, la valeur true indique que la table est occupée)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. La capacité a MAXCAP, id_ = -1, nbPlaces_ = 1, occupee_ = false, le tableau commande_ est alloué de taille MAXCAP.

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes et qui construit le menu avec la méthode de lecture.
- Les méthodes d'accès.
- Les méthodes de modification.
- libererTable() efface la commande et libère la table.
- commander(), prenant en paramètre un plat qui sera placé dans la commande.
- getChiffreAffaire(), retourne le chiffre d'affaire généré par la table pour le restaurant (calculé sur base du prix et du cout de production de chaque plat).
- Une méthode d'affichage.

Classe Restaurant

Cette classe caractérise un restaurant, avec un ensemble de tables, 3 menus en fonction du moment de la journée et un chiffre d'affaire.

Elle contient les attributs privés suivants :

- nom_, pointeur vers un string
- chiffreAffaire_, double
- momentJournee_, le moment de la journée, permettant de définir quel menu utilisé (il provient du type énuméré définit dans le menu)
- menuMatin, Midi, Soir, pointeur vers des menus.
- capaciteTables_, unsigned int.
- tables_, tableau dynamique contenant des pointeurs vers des tables.
- nbTables_, unsigned int.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. Le nom du restaurant est inconnu, le moment de la journée est le Matin, et le chiffre d'affaire est nul.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès.
- Les méthodes de modifications.
- Une méthode lireTable() prenant en paramètre le nom du fichier. Elle a pour but de créer les tables, elle est appelée dans le constructeur par paramètres.
- Une méthode ajouterTable(), utilisée dans la méthode lireTable().
- Une méthode libérerTable(), en plus de libérer la table, elle devrait mettre à jour le chiffre d'affaire du restaurant avec le chiffre généré par la table.
- Une méthode commanderPlat (prenant en paramètre le nom d'un plat et l'id d'une table).
- Une méthode placerClients (prenant en paramètre un entier symbolisant le nombre de clients à placer), la méthode devra alors placer les clients de manière intelligente. Par exemple si deux tables sont disponibles : une de quatre places et une autre de six et que le

groupe à placer est de 2 personnes, ils seront placés sur la table de quatre. De plus s'il n'y a pas de tables disponible, un message d'erreur apparaitra. Note : aucune classe client n'existe pour le moment, dans ce tp un client est symbolisé par un entier déclaré dans le main. Cet entier représente le nombre de personnes désirant manger au restaurant.

- Une méthode d'affichage.

Fichier PolyFood.txt

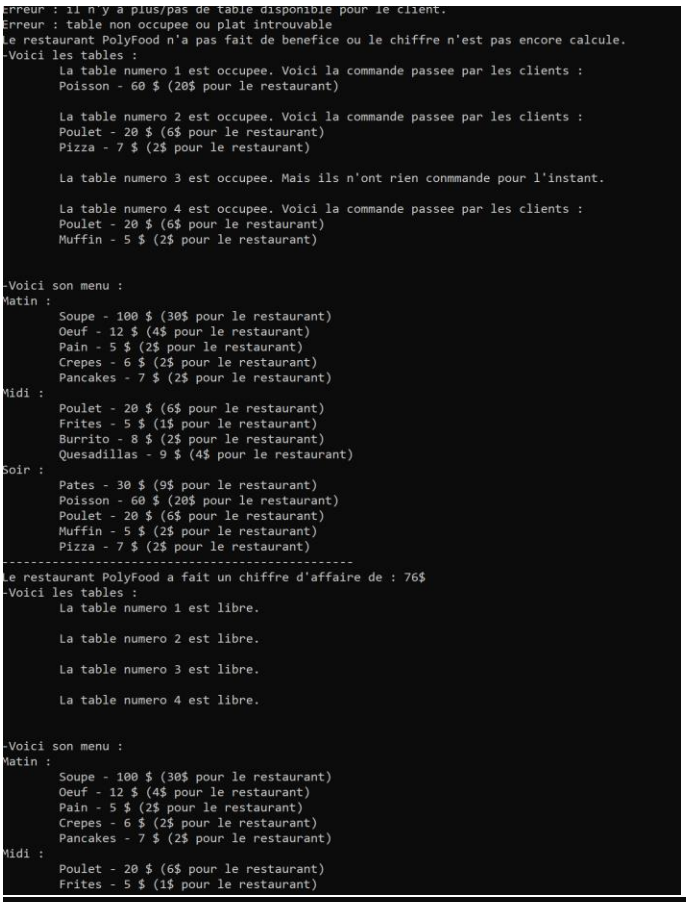
Format :

Début (non contenu dans le fichier)
-MATIN
Plat prix(double) cout(double)
.
.
.
-MIDI
Plat prix(double) cout(double)
.
.
.
-SOIR
Plat prix(double) cout(double)
.
.
.
-TABLES
Id(int) NbPlaces(int)
.
.
.
Fin (non contenu dans le fichier)

Main.cpp

Des directives vous sont fournies dans le fichier main.cpp et il vous est demandé de les suivre.

Votre affichage devrait avoir une apparence semblable à celle ci-dessous. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable si vous le désirez.



```
Erreur : il n'y a plus/pas de table disponible pour le client.
Erreur : table non occupee ou plat introuvable
Le restaurant PolyFood n'a pas fait de benefice ou le chiffre n'est pas encore calcule.
-Voici les tables :
        La table numero 1 est occupee. Voici la commande passee par les clients :
        Poisson - 60 $ (20$ pour le restaurant)

        La table numero 2 est occupee. Voici la commande passee par les clients :
        Poulet - 20 $ (6$ pour le restaurant)
        Pizza - 7 $ (2$ pour le restaurant)

        La table numero 3 est occupee. Mais ils n'ont rien conmmande pour l'instant.

        La table numero 4 est occupee. Voici la commande passee par les clients :
        Poulet - 20 $ (6$ pour le restaurant)
        Muffin - 5 $ (2$ pour le restaurant)


-Voici son menu :
Matin :
        Soupe - 100 $ (30$ pour le restaurant)
        Oeuf - 12 $ (4$ pour le restaurant)
        Pain - 5 $ (2$ pour le restaurant)
        Crepes - 6 $ (2$ pour le restaurant)
        Pancakes - 7 $ (2$ pour le restaurant)
Midi :
        Poulet - 20 $ (6$ pour le restaurant)
        Frites - 5 $ (1$ pour le restaurant)
        Burrito - 8 $ (2$ pour le restaurant)
        Quesadillas - 9 $ (4$ pour le restaurant)
Soir :
        Pates - 30 $ (9$ pour le restaurant)
        Poisson - 60 $ (20$ pour le restaurant)
        Poulet - 20 $ (6$ pour le restaurant)
        Muffin - 5 $ (2$ pour le restaurant)
        Pizza - 7 $ (2$ pour le restaurant)
------------------------------------------------
Le restaurant PolyFood a fait un chiffre d'affaire de : 76$
-Voici les tables :
        La table numero 1 est libre.

        La table numero 2 est libre.

        La table numero 3 est libre.

        La table numero 4 est libre.


-Voici son menu :
Matin :
        Soupe - 100 $ (30$ pour le restaurant)
        Oeuf - 12 $ (4$ pour le restaurant)
        Pain - 5 $ (2$ pour le restaurant)
        Crepes - 6 $ (2$ pour le restaurant)
        Pancakes - 7 $ (2$ pour le restaurant)
Midi :
        Poulet - 20 $ (6$ pour le restaurant)
        Frites - 5 $ (1$ pour le restaurant)
```

```
-Voici son menu :
Matin :
        Soupe - 100 $ (30$ pour le restaurant)
        Oeuf - 12 $ (4$ pour le restaurant)
        Pain - 5 $ (2$ pour le restaurant)
        Crepes - 6 $ (2$ pour le restaurant)
        Pancakes - 7 $ (2$ pour le restaurant)
Midi :
        Poulet - 20 $ (6$ pour le restaurant)
        Frites - 5 $ (1$ pour le restaurant)
        Burrito - 8 $ (2$ pour le restaurant)
        Quesadillas - 9 $ (4$ pour le restaurant)
Soir :
        Pates - 30 $ (9$ pour le restaurant)
        Poisson - 60 $ (20$ pour le restaurant)
        Poulet - 20 $ (6$ pour le restaurant)
        Muffin - 5 $ (2$ pour le restaurant)
        Pizza - 7 $ (2$ pour le restaurant)
Appuyez sur une touche pour continuer...
```

Spécifications générales

- Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.

- Appliquer un affichage « user friendly » (ergonomique et joli) pour le rendu final.
- Documenter votre code source.
- **Bien lire le barème de correction ci-dessous.**

Questions

1. Quel est le lien (agrégation ou composition) entre les classes Menu et Plat ?
2. Quel effet aura une méthode si elle a un const ?

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme ;

- (3 points) Exécution du programme ;

- (4 points) Comportement exact des méthodes du programme ;

- (2 points) Documentation du code et bonne norme de codage ;

- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés ;

- (2 points) Utilisation adéquate des directives de précompilation ;

- (2 points) Allocation et désallocation appropriées de la mémoire ;

- (2 points) Réponse aux questions ;