INF1010

Programmation Orientée-Objet

Travail pratique #2
Vecteurs et surcharge d'opérateurs

Objectifs : Permettre à l'étudiant de se familiariser avec la surcharge d'opérateurs, les vecteurs de la librairie STL et l'utilisation du pointeur ***this***.

Remise du travail : Lundi 11 février 2019, 8h (AM)

Références : Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.

Documents à remettre : Tous les fichiers .cpp et .h exclusivement complétés et 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

Travail à réaliser

Le travail consiste à continuer l'application PolyFood commencée au TP précédent en y intégrant les notions de vecteurs, de surcharge d'opérateurs, de constructeur par copie et d'opérateur d'affectation.
Pour remplacer les tableaux dynamiques qui rendaient la gestion des dépenses et des utilisateurs difficile, ce TP fait appel aux vecteurs de la librairie STL, soit std::vector (ou vector si on utilise using namespace std). Par ailleurs, pour faciliter les interactions avec les différents objets, la surcharge d'opérateurs sera utilisée.
Les vecteurs implémentés en C++ (STL) sont très pratiques : ce sont des tableaux dont la taille est dynamique. On peut y ajouter des éléments sans se préoccuper de la taille de notre vecteur étant donné que la gestion de la mémoire est automatique. Lorsqu'il vous est demandé d'utiliser un vecteur de la STL plutôt qu'un tableau dynamique pensez à **bien supprimer tous les attributs qui n'ont plus lieu d'être et d'adapter les méthodes qui ont besoin d'être mises à jour**.
Le langage C++ est un langage avec lequel il est possible de redéfinir la manière dont fonctionnent la plupart des opérateurs (arithmétiques (+, -, *, /), d'affectation, etc..) pour de nouvelles classes. Nous pouvons donc redéfinir le comportement de ces opérateurs afin qu'ils effectuent une nouvelle opération ou englobent plusieurs opérations pour ces nouvelles classes.
Pour vous aider, les fichiers corrigés du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas et supprimer les attributs qui n'ont plus lieu d'être. Les méthodes à modifier vous ont été indiquées.
ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaire.
ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.
ATTENTION : Il faut utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP1.
Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h

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.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du plat.
- L'opérateur < qui compare 2 plats selon leur prix de vente.

Classe Table

Cette classe caractérise une table du restaurant, elle a un identifiant, une liste de plats (la commande de la table), et un nombre de places.

Elle contient les nouveaux attributs privés suivants :

- nbClientsATable_ : un entier qui représente le nombre de clients assis à la table.
- Par conséquent, nbPlaces_ se met à jour en fonction du nombre de clients qui s'assoit à la table. L'attribut occupee_ a été retiré de la classe.
- commande_: un vecteur de pointeurs à des plats commandés (pensez à supprimer les attributs obsolètes et d'ajuster les méthodes en conséquence voir les // A MODIFIER).

Les méthodes suivantes ont été <u>ré implémentées</u>:

- Une méthode d'accès à l'attribut nbClientsATable_.
- Une méthode estPleine() qui verifie si nbPlaces_ est égal à 0 ou non.
- La méthode libérerTable() rétablit le nombre de places initiales de la table, et met le nombre de clients à la table à 0.
- La méthode placerClients(int nbClients) prend désormais en paramètre un nombre de clients. Elle va mettre à jour la valeur de nbClientsATable_ et de nbPlaces_.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par copie
- L'opérateur = qui écrase les attributs de la table par les attributs de la table passée en paramètre et qui renvoie ensuite une référence à la table.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du table (voir l'affichage à la fin de l'énoncé).

Classe Menu

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

Elle contient le nouvel attribut privé suivant :

- listePlats_ : un vecteur de pointeur de plats présents sur le menu (pensez à supprimer les attributs obsolètes et d'ajuster les méthodes en conséquence voir les // A MODIFIER).

La méthode suivante a été <u>implémentée</u>:

- trouverPlatMoinsCher() : Méthode constante, qui permet de trouver le plat moins cher parmi la listePLats_ (elle utilise les vecteurs, donc elle ne compile pas avec la version de l'énoncé fourni, mais elle doit fonctionner une fois le tp terminé).

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du menu (voir l'affichage à la fin de l'énoncé).
- Un constructeur par copie.
- L'opérateur += (remplace la méthode ajouterPlat(const Plat& plat)).
- L'opérateur = qui écrase les attributs du menu par les attributs du menu passé en paramètre et qui renvoie ensuite une référence au menu.
- Une méthode d'accès au nouvel attribut.

Classe Restaurant

Cette classe caractérise le restaurant ayant des tables et des clients qui commandent des plats.

Elle contient les nouveaux attributs privés suivants :

- tables_ : un vecteur de pointeur de tables dans le restaurant (pensez à supprimer les attributs obsolètes et d'ajuster les méthodes en conséquence voir les // A MODIFIER).

Les méthodes suivantes doivent être implémentées :

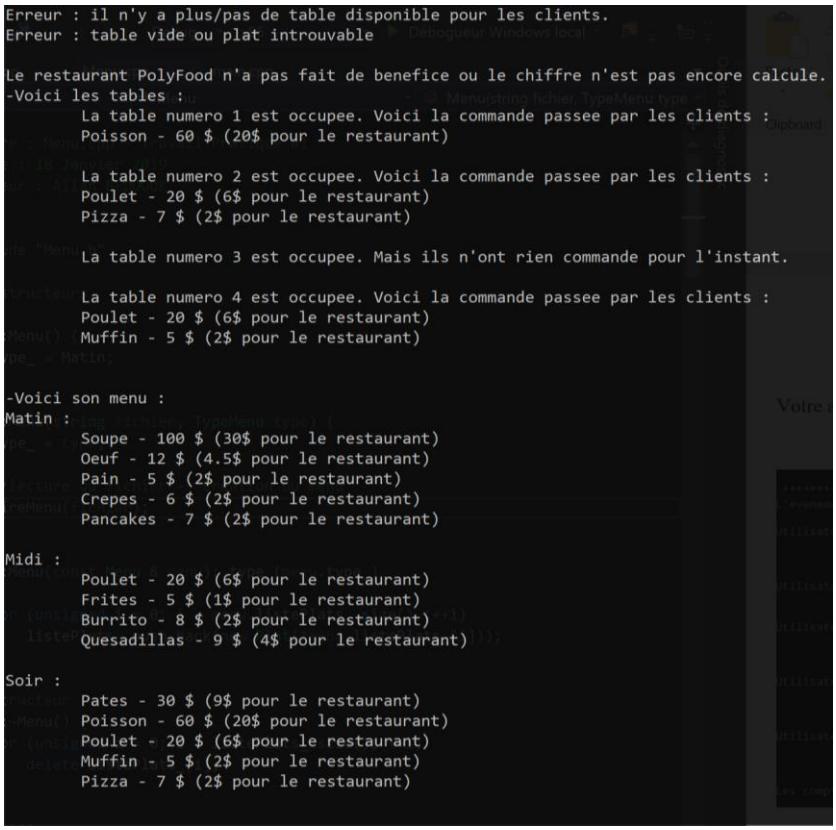
- L'opérateur += prenant en paramètre un pointeur à une table, permettant d'ajouter une table au restaurant (remplace ajouterTable()). Il doit être possible de faire des appels en cascade.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du restaurant.
- L'opérateur = qui écrase les attributs du restaurant par les attributs du restaurant passé en paramètre et qui renvoie ensuite une référence au restaurant.
- Un constructeur par copie.
- L'opérateur < qui compare le chiffre d'affaires de deux restaurants

Main.cpp

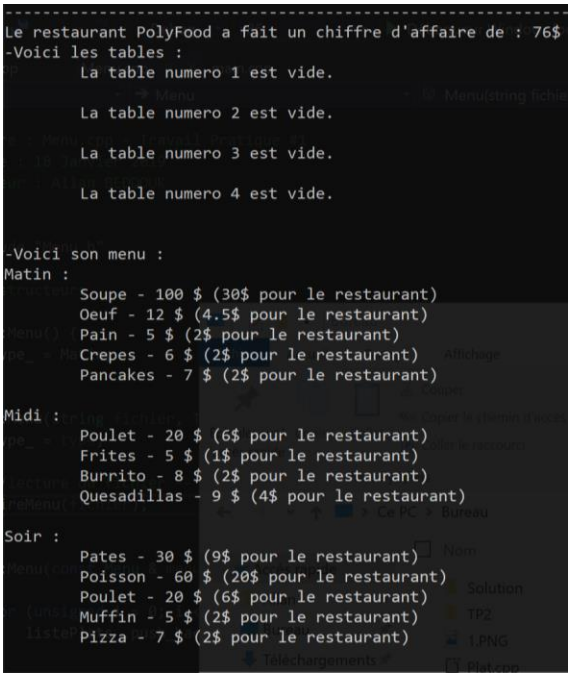
Des directives vous sont fournies dans le fichier main.cpp et il vous est demandé de les suivre.

Votre affichage devrait avoir une apparence semblable à celle ci-dessous.

1er affichage PolyFood :

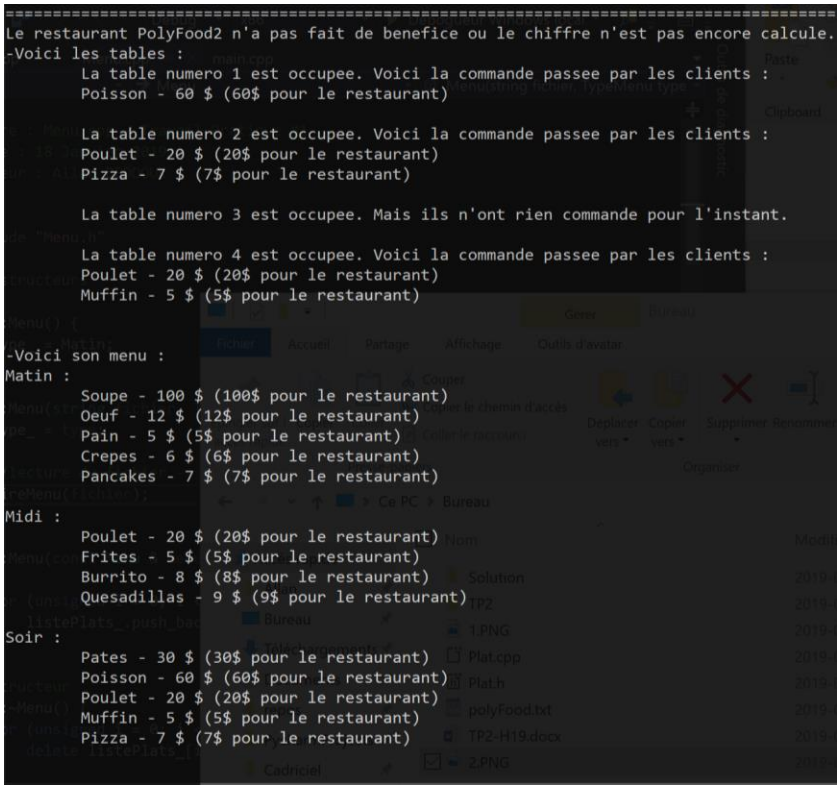


2eme affichage PolyFood :



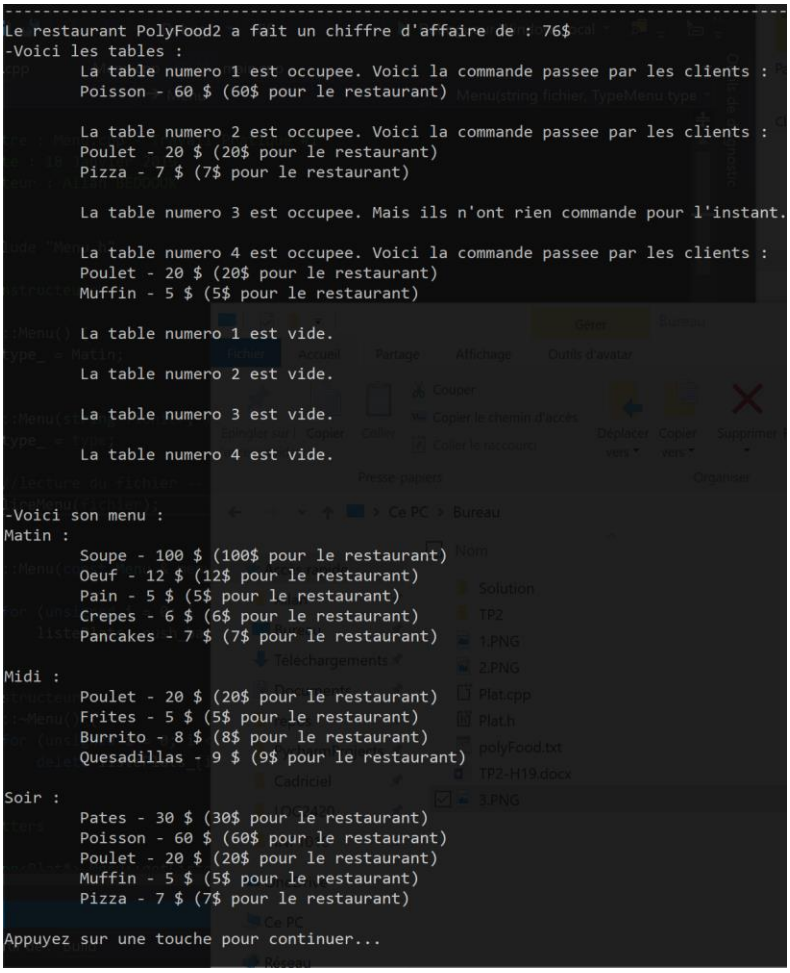
```
Le restaurant PolyFood a fait un chiffre d'affaire de : 76$
-Voici les tables :
        La table numero 1 est vide.

        La table numero 2 est vide.

        La table numero 3 est vide.

        La table numero 4 est vide.


-Voici son menu :
Matin :
        Soupe - 100 $ (30$ pour le restaurant)
        Oeuf - 12 $ (4.5$ 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)
```

1er affichage PolyFood2 :



```
Le restaurant PolyFood2 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 $ (60$ pour le restaurant)

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

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

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


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

2eme affichage PolyFood2 :



```
Le restaurant PolyFood2 a fait un chiffre d'affaire de : 76$
-Voici les tables :
        La table numero 1 est occupee. Voici la commande passee par les clients :
        Poisson - 60 $ (60$ pour le restaurant)

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

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

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

        La table numero 1 est vide.

        La table numero 2 est vide.

        La table numero 3 est vide.

        La table numero 4 est vide.

-Voici son menu :
Matin :
        Soupe - 100 $ (100$ pour le restaurant)
        Oeuf - 12 $ (12$ pour le restaurant)
        Pain - 5 $ (5$ pour le restaurant)
        Crepes - 6 $ (6$ pour le restaurant)
        Pancakes - 7 $ (7$ pour le restaurant)

Midi :
        Poulet - 20 $ (20$ pour le restaurant)
        Frites - 5 $ (5$ pour le restaurant)
        Burrito - 8 $ (8$ pour le restaurant)
        Quesadillas - 9 $ (9$ pour le restaurant)

Soir :
        Pates - 30 $ (30$ pour le restaurant)
        Poisson - 60 $ (60$ pour le restaurant)
        Poulet - 20 $ (20$ pour le restaurant)
        Muffin - 5 $ (5$ pour le restaurant)
        Pizza - 7 $ (7$ pour le restaurant)

Appuyez sur une touche pour continuer...
```

Spécifications générales

- Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.
- Appliquez un affichage « user friendly » (ergonomique et joli) pour le rendu final.
- Documenter votre code source.
- **Bien lire le barème de correction ci-dessous.**

Questions

Répondez aux questions au début du main.

1. Quel est l'utilité de l'opérateur = et du constructeur par copie ?
2. Qu'est-ce qui différencie l'opérateur = du constructeur par copie ?

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 ;
- (3 points) Surcharge correcte des opérateurs ;
- (2 points) Utilisation correcte des vecteurs ;
- (2 points) Documentation du code et bonne norme de codage ;
- (1 point) Utilisation correcte du mot-clé *this* pour les opérateurs ;
- (2 points) Réponse aux questions.