UNIVERSITE GASTON BERGER **ANNEE 2016-2017**
DE SAINT-LOUIS
UFR SAT
SECTION INFORMATIQUE

Programmation Orientée Objet
Examen Première Session
Licence
Durée : 2h30
Documents non autorisés

<u>Exercice 1</u> (15 pts)

Une compagnie pharmaceutique souhaite connaître les coûts de fabrication des vaccins qu'elle produit. On considère pour cela le fichier `vaccin.cpp` fourni en annexe (qu'il s'agira de copier et de compléter) :

I.

Codez dans le fichier `vaccins.cpp` la classe `Vaccin` dotée des attributs suivants : son *nom*, le *volume par dose* (un `double`), le *nombre de doses à produire* (un `int`) et le *mode de fabrication* (qui peut être standard ou high tech, le type à donner est décrit plus bas).

Dotez votre classe `Vaccin` des méthodes publiques suivantes :

1. un constructeur dont les paramètres permettent d'initialiser l'ensemble des attributs. L'attribut *mode de fabrication* aura `Standard` comme valeur par défaut (voir le type énuméré `Fabrication` fourni en début de fichier);
2. une méthode d'affichage permettant d'afficher un vaccin selon le format suivant :

```
Triphas
volume/dose : 0.2
nombre de doses: 10000
mode de fabrication standard
```

3. une méthode `conditionnement` calculant le coût de conditionnement comme étant le *volume par dose* multiplié par le *nombre de doses à produire*, multiplié par `COND_UNITE` (une constante fournie dans le programme en annexe);
4. une méthode `fabrication` calculant le coût de fabrication d'un vaccin. Le coût de base pour la fabrication est *le volume par dose* multiplié par le *nombre de doses à produire*, multiplié par `PRIX_BASE` (une constante fournie dans le programme en annexe). Si le mode de fabrication est `HighTech`, ce coût est majoré d'un pourcentage `MAJORATION_HIGHTECH` (c'est à dire `coût += coût * MAJORATION_HIGHTECH`) où `MAJORATION_HIGHTECH` est aussi une constante fournie;
5. une méthode `production` calculant le coût de production d'un vaccin comme étant la somme des coûts de conditionnement et de fabrication.

II.

La compagnie a décidé de délocaliser la fabrication de certains vaccins.

Codez une classe spécialisée `Delocalise` de `Vaccin` ayant pour attribut supplémentaire un booléen indiquant si la production est délocalisée dans un pays frontalier ou non.

Dotez votre nouvelle classe des méthodes suivantes :

1. un constructeur approprié
2. une méthode `production` qui redéfinit la méthode héritée de `Vaccin`. Soit `c` le coût de production tel que calculé dans la classe `Vaccin`. Le coût de production d'un vaccin délocalisé vaudra :
    - `c - c * REDUC_DELOC` si le vaccin est délocalisé dans un pays frontalier,
    - `c/2` sinon.

Pour tester votre programme, vous ajouterez dans le `main` deux vaccins (v3, v4) délocalisés respectivement dans un pays non frontalier et dans un pays frontalier.

III.

Vous allez maintenant définir une classe `Compagnie` modélisant la compagnie pharmaceutique qui fabrique les vaccins. Cette classe sera caractérisée par :

- le nom de la compagnie;
- son stock de vaccins.

La classe `Compagnie` devra être dotée des méthodes suivantes :

- un constructeur prenant en paramètre une chaîne de caractères pour initialiser le nom de la compagnie
- un destructeur
- une méthode permettant d'ajouter un `Vaccin` au stock de vaccins de la compagnie
- une méthode permettant d'afficher l'ensemble de vaccins en stock
- une méthode permettant de calculer le coût de production de l'ensemble des vaccins du stock.

Complétez votre `main` avec :

- une instruction permettant de créer une `Compagnie` nommée `"ICIBA"` et ajoutant à son stock les vaccins `v1, v2, v3` et `v4`;
- une instruction permettant d'afficher le stock de vaccins de la compagnie;
- les instructions permettant d'afficher le coût de production de l'ensemble du stock.

Exercice 2 (5 pts)

On souhaite créer une classe *int2d* permettant de représenter des tableaux dynamiques d'entiers à deux indices, c'est-à-dire dont les dimensions peuvent ne pas être connues lors de la compilation. Plus précisément, on prévoira de déclarer de tels tableaux par une déclaration de la forme :

int2d t(exp1, exp2) ;

dans laquelle *exp1* et *exp2* désignent une expression quelconque (de type entier).

On surdéfinira l'opérateur ( ), de manière qu'il permette d'accéder à des éléments d'un objet d'un type *int2d* comme on le ferait avec un tableau classique.

Pour cela, on donne en annexe la définition à compléter de la classe *int2d*.
On ne cherchera pas à résoudre les problèmes posés éventuellement par l'affectation ou la transmission par valeur d'objets de type *int2d*. En revanche, on s'arrangera pour qu'il n'existe aucun risque de débordement d'indice.

- Dans un fichier « in2d.h », donnez la définition complète de la classe *int2d* (voir Annexe) *;*
- Dans un fichier « int2d.cpp », définissez le constructeur, le destructeur et la surdéfinition de l'opérateur ( ) ;
- Dans un fichier « main.cpp », donnez un exemple d'utilisation de la classe *int2d* dans une fonction « main ».

Annexe : la classe *int2d* à compléter

```
class int 2d
{
        int nlig ;        // nombre de lignes
        int ncol ;        // nombre de colonnes
        int *adr ;        // adresse emplacement dynamique contenant les valeurs
  public :
      // fonctions membres
}
```

Annexe

```
#include <string>
#include <iostream>

using namespace std;

// prix du conditionnement d'une unité
const double COND_UNITE = 0.5;

// prix de base de fabrication d'une unité
const double PRIX_BASE = 1.5;

// majoration du prix de fabrication pour vaccin "high tech"
const double MAJORATION_HIGHTECH = 0.5;

// reduction du cout du à la delocalisation
const double REDUCTION_DELOC = 0.2;

enum Fabrication {Standard, HighTech};

// CLASSES A COMPLETER ICI

// ========================================================================
//
int main() {



    Vaccin v1("Zamiflu", 0.55, 200000, HighTech);
    Vaccin v2("Triphas", 0.20 , 10000);



    // affichage des vaccins à compléter ici
    cout << v1 << endl;
    cout << v2 << endl;

    cout << "le  cout de production de  v1 et v2 est : ";
    cout << v1.cout_production() + v2.cout_production() << endl;

    // Test des parties suivantes
     cout << "test des parties suivantes ..." << endl;
    // A COMPLETER
  return 0;
}
```