MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DE MONASTIR

Institut Supérieur d'Informatique et de Mathématiques

Département Informatique

FASCICULE DE PROGRAMMATION C++

ENSEIGNANT :

Taoufik SAKKA ROUIS

---Année Universitaire 2022-2023---

Travaux Pratiques N°1

Exercice 1

Écrire un programme C++ qui permet le remplissage d'un tableau **Tab** de **N** entiers (N<= 10). Calculer puis afficher la somme de ses éléments.

Exercice 2

1) Créer la procédure **permut (int *a, int *b)** qui permet la permutation de 2 entiers A et B.

2) Redéfinir la procédure **permut** en utilisant le passage par référence des arguments,

3) Ecrire la fonction **main()** qui teste les deux procédures décuites précédemment.

Exercice 3

1) Définir la structure de donnée **Vecteur** qui permet de représenter un vecteur de deux dimensions.

2) Écrire la fonction **prodSacl** qui permet de calculer le produit scalaire de deux vecteurs.

3) Écrire la fonction principale **main()** qui permet de tester la fonction **prodScal** sur des vecteurs saisies au clavier.

Exercice 4

1) Écrire un petit programme C++ qui permet de gérer des personnes. Une personne est codé dans le programme sous forme d'une structure avec les données nom et prénom.

2) Écrire une fonction lecturePers qui permet de lire une personne à partir du clavier.

3) Écrire une fonction affichePers qui permet d'afficher la personne à l'écran.

Exercice 5

On veut maintenant réaliser un programme qui gère un certain nombre de personnes. On a choisi de faire une gestion dynamique des personnes sous forme d'une liste chaînée.

1) Modifier le programme précédent pour tenir compte de la nouvelle structure.

2) Écrire une fonction addToListe qui permet d'ajouter une personne à la liste.

3) Écrire une fonction afficheListe qui permet d'afficher tous les éléments de la liste.

4) Écrire une fonction deletePersFromListe qui permet de supprimer une personne de la liste

Travaux Pratiques N°2

Exercice 1 :

Écrivez une classe Point comportant :

- Comme membres données privées : deux composantes de type int,
- Une fonction initialise avec deux arguments représentant les composantes,
- Une fonction deplace avec deux arguments permettant de déplacer un point,
- Une fonction affiche permettant d'afficher la position d'un point.

Toutes ces fonctions membres sont publiques. Écrivez ensuit une fonction main qui permet d'initialiser deux points, de les afficher, de les déplacer et de les réafficher.

Exercice 2 :

On se propose de définir la classe Etudiant. Cette classe est caractérisée par : Numéro d'inscription, nom, prénom, moyenne du premier semestre, moyenne du second semestre, et la moyenne générale. On vous demande de :

1) Définir la classe Etudiant.
2) Développer les fonctions suivantes :
   - Saisie_etudiant
   - Afficher_etudiant
   - Calcul_moy_gen_etudiant
3) Utiliser cette classe dans une fonction main en :
   - Créant un tableau de 10 étudiants représentant les étudiants d'un groupe
   - Calculant la moyenne générale du groupe.

Exercice 3 :

On se propose de définir la classe Entier qui représente tous les entiers. Cette classe est composée d'un seul attribut privé de type int.

1) Déclarer cette classe en la dotant des méthodes suivantes :
   - Saisie_entier
   - Afficher_entier
   - Additionner un entier avec un autre
   - Soustraire un entier d'un autre
   - Retourner_valeur
2) Développer les méthodes de cette classe.
3) Ajouter à la déclaration de la classe, tous les constructeurs possibles ainsi que le destructeur.
4) Utiliser cette classe dans une fonction main.

Travaux Pratiques N°3

Exercice 1 :

Écrivez une classe Cercle comportant :

- comme données privées trois composantes de type float : rayon, abscisse et ordonnée de l'origine du cercle`
- un constructeur public défini **en ligne**,
- une fonction amie périmètre qui calcule le périmètre d'un cercle (2*PI*R),
- une fonction amie air qui calcule la surface d'un cercle ($PI*R^2$),
- une fonction sortie indépendante (à l'extérieur de la classe), qui affiche le rayon, l'abscisse et l'ordonnée, le périmètre et la surface d'un cercle.

Écrivez une fonction main utilisant des objets qui tester ces fonctions.
NB. PI doit être défini comme constante.

Exercice 2 :

Créer une classe int_tab qui génère un tableau de n éléments du type int. Le nombre d'éléments du tableau sera défini à la génération d'un objet int_tab par un paramètre du constructeur. La classe doit également disposer d'un destructeur. Enfin, seule une deuxième classe, ami_tab, sera autorisée à créer des objets du type int_tab.

Exercice 3 :

Écrivez une classe Vecteur comportant :

- comme membres données privées : trois composantes de type double,
- une fonction affiche,
- deux constructeurs :
  - l'un, sans arguments, initialisant chaque composante à 0,
  - l'autre, avec 3 arguments, représentant les composantes,
- une fonction homothétie pour multiplier les composants par une valeur fournie en argument.
- une fonction prod-scal qui fournisse en résultat le produit scalaire de deux vecteurs,
- une fonction somme permettant de calculer la somme de deux vecteurs.

a- avec des fonctions membres indépendantes,
b- avec des fonctions membres en ligne.
Toutes les fonctions sont publiques. Écrivez ensuite une fonction main qui permet de les tester.

Exercice 4 :

Ecrire un programme C++ qui lit un nombre entier et l'élève au carré, sans ligne de code dans main. Pour cela, créer une classe square qui contient que le constructeur.

Travaux Pratiques N°4

Exercice 1 :

Soit la classe Vector dont la spécification est donnée comme suit :

```
class Vector {
      const int Size;
      int* Elements;
public :
      Vector(int S);
      Vector(int S, int* Elts);
      Vector(const Vector& V);
      ~Vector();
      void Show();
      void Set(); // Effectue la saisie de tous les éléments du vecteur
      void Set(int Index, int Value);
      int Get(int Index);
};
```

1- Donner les définitions des méthodes de la classe ***Vector***.

2- Donner une fonction principale qui montre l'instanciation de la classe ***Vector*** et l'utilisation de ses différentes méthodes.

Exercice 2 :

Soit la classe Matrix dont la spécification est donnée comme suit :

```
class Matrix {
      const int NbLines;
      const int NbColumns;
      int** Elements;
public :
      Matrix (int L, int C);
      Matrix (int L, int C, int** Elts);
      Matrix(const Matrix& M);
      ~Matrix();
      void Set(int LineIndex, int ColumnIndex, int Value);
      int Get(int LineIndex, int ColumnIndex);
      void Show(); // Affiche le contenu de la matrice
      void Set(); // effectue la saisie de tous les éléments
};
```

1- Donner les définitions des méthodes de la classe ***Matrix***.

2- Donner une fonction principale qui montre l'instanciation de la classe ***Matrix*** et l'utilisation de ses différentes méthodes.

Exercice 3 :

1- Proposer une fonction ***Multiply*** externe et amie aux deux classes ***Vector*** et ***Matrix*** qui effectue le produit d'une matrice et d'un vecteur.

2- Transformer la fonction amie ***Multiply*** de façon à ce qu'elle devienne une méthode de la classe ***Matrix.*** Indiquer comment faut-il modifier l'organisation du code pour pouvoir compiler avec succès.

Travaux Pratiques N°5

Problème A :

On désire programmer une mini-application de gestion de petit commerce. Pour cela, on va écrire une classe **Produit** et une classe **Magasin**.

Exercice 1 : Première version du magasin

Dans notre commerce, chaque produit est caractérisé par son nom, son prix d'achat, son prix de vente, le nombre d'exemplaires en stock et sa description.

A la création du produit, on fixe son nom, son prix d'achat et son prix de vente, c'est-à-dire que ces données sont fournies en argument au constructeur de la classe. Par défaut, la description du produit est «Pas de description», et la quantité de produits en stock est nulle.

La classe **Produit** dispose d'un certain nombre de méthodes, qui lui permettent d'afficher la description du produit, de l'éditer, d'augmenter ou de diminuer le nombre d'exemplaires en stock, ainsi que d'obtenir les valeurs des différents attributs.

Question 1.1: Écrivez la classe Produit.

Un magasin se caractérise par son solde et son stock de produits. Le stock de produit est représenté par un tableau d'objets «**Produit**».

Avant de pouvoir acheter ou vendre un produit, il faut l'avoir ajouté dans le stock. Pour cela, la classe **Magasin** dispose d'une méthode publique:

void ajouterProduit (String nom, float prixAchat, float prixVente)

L'indice de la case du tableau dans laquelle on a stocké le produit devient alors la référence de ce produit. Pour acheter ou vendre ce produit, on utilise alors la référence comme argument des méthodes : **void acheterProduit (int referenceProduit, int nombreExemplaires)**

void vendreProduit (int referenceProduit, int nombreExemplaires)

La classe **Magasin** dispose également des méthodes habituelles d'accès à ses attributs, et d'une méthode Bilan qui permet d'afficher un bilan du magasin.

Question 1.2: Écrivez la classe Magasin. Testez-la.

Exercice 2: Différents types de produits

Jusque-là, les produits disposent d'une description qui est une simple chaîne de caractère. Cela peut être insuffisant dans certains cas. Par exemple, un livre est un produit particulier qui est identifié par son auteur et son éditeur. De même, un produit alimentaire est caractérisé par sa date de fabrication et sa date d'expiration.

Question 2.1 : Écrivez la classe Livre et la classe Aliment qui héritent de la classe Produit. Certaines méthodes doivent être surchargées (c'est-à-dire réécrites).

Il va maintenant falloir faire les modifications nécessaires dans la classe **Magasin**, mais elles sont minimes. En effet, **Livre** étant une sous-classe de **Produit**, les objets de type **Livre** sont également de type **Produit**, donc le tableau de produits peut également contenir des livres et des aliments. C'est à la création des objets que les choses changent. En plus de la méthode **ajoueterProduit**, la classe **Magasin** doit également contenir une méthode **ajouterLivre** et une méthode **ajouterAliment**.

Question 2.2: Modifier la classe Magasin pour tenir compte des nouveaux objets.

Exercice 3: Recherche par mot

Dans les parties précédentes, pour acheter, vendre, éditer ou afficher la description d'un produit existant, il fallait le spécifier par sa référence, c'est-à-dire par son indice dans le tableau. Ce n'est pas un moyen très pratique d'aller chercher les produits, surtout s'il y en a un très grand nombre.

Nous allons écrire les méthodes :

- **Produit rechercheProduit (String nom)** qui renvoie un objet Produit dont le nom est spécifie en argument,
- **Produit rechercheProduitParMot (String mot)** qui renvoie un objet Produit dont la description contient le mot spécifié en argument.

Pour écrire ces méthodes, vous allez avoir besoin d'un outil capable de comparer deux chaines de caractères, et d'un outil capable de dire si une chaine est incluse dans une autre.

Question 3.1: Écrivez les méthodes rechercherProduit et rechercherProduitParMot.

Problème B :

On désire programmer une mini-application de gestion des ports de pêche. Pour des raisons de simplicité, on se limite à une description sommaire des barques (petits bateaux) et des ports. Une barque est caractérisée par son nom, son numéro d'autorisation ainsi que par une méthode **toChaine()** qui décrit ses propriétés. Il existe aussi des barques dotées de moteurs de puissances différentes. La puissance, exprimée en nombre entier, est supposée l'information unique qu'il faut représenter pour le moteur d'une barque. Une méthode **getPuissance()** est prévue pour chaque barque dotée de moteur. Un port de pêche est caractérisé, entre autres, par un nom et une capacité d'accueil de barques. Une taxe journalière est prévue pour chaque barque se trouvant dans le port. Le montant de cette taxe est approprié à chaque port. Une

majoration de la taxe est prévue pour les barques avec moteur à raison de 10% si la puissance du moteur est < 5 et de 20% si non. Il faut connaître, à chaque moment, le nombre actuel de barques dans un port donné. Aussi, il faut gérer les opérations d'entrée et de sortie de barques par des méthodes appropriées. En plus d'une méthode de description **toChaine()** qui décrit aussi les caractéristiques des barques se trouvant dans le port, il faut prévoir une méthode qui calcule le montant d'une recette de taxe pour la journée en cours. Utilisez un seul vecteur pour représenter toutes les barques dans un port (de type Barque et BarqueMoteur à la fois).

Questions :

1. Définir les classes Barque, BarqueMoteur et Port.
2. Définir une classe TestPort qui se charge de la création d'un objet Port. D'un objet BarqueMoteur. Cette fonction doit faire entrer les deux barques au port et calculer le montant de taxe.

Travaux Pratiques N°6

Exercice 1:

a) Ecrivez une classe nommé PileEntier permettant de gérer une pile d'entier. Ces derniers seront conservés dans un tableau d'entiers alloués dynamiquement.

La classe comportera les membres suivants :

Des champs privés :

- int nelem; // le nombre d'entier actuellement empilés
- int dim // la taille max de la pile
- int* adr // Tableau contenant les éléments de la pile

Des méthodes publiques:

- PileEntier (int n) : constructeur allouant dynamiquement un emplacement de n entiers
- PileEntier(): constructeur sans argument allouant par défaut un emplacement de 20 entier
- PileEntier (PileEntier&) : constructeur par recopie
- ~PileEntier() : destructeur
- void empiler(int p) : ajoute l'entier p sur la pile,
- void depiler () : dépile le dernier élément empilé et non encore dépilé.
- bool pleine () : fournit true si la pile est pleine sinon false
- bool vide() : fournit true si la pile est vide, false sinon
- int dernier() : fonction qui récupère le sommet de la pile.

b) Écrivez une fonction main utilisant des objets automatiques et dynamiques du type PileEntier. Mettez en évidence les problèmes posés par les déclarations de la forme :

```
PileEntier a (10) ;
PileEntier b=a ;
```

c) Introduisez les opérateurs > et < tels que si p est un objet un objet de type PileEntier et n une variable entière :

p<n ajoute la valeur n sur la pile p (en ne revoyant auccune valeur),
p>n supprime la valeur du haut de la pile et la place dans n.

On prévoira les deux situations :

- Fonction membre,
- Fonction amie.

Exercice 2 :

On souhaite réaliser une classe compte permettant de manipuler les comptes bancaires. On prévoit que sa déclaration se présente ainsi :

```
class compte{
        float solde ;
        static int numero ;
        const int cle;
        …
}
```

1. Écrire un constructeur qui reçoit en argument le montant du solde initial (0 par défaut), affecte à la donnée membre cle la valeur de numero et incrémente numero.
2. Peut-on définir une constructeur par recopie pour cette classe ? si oui, le définir.
3. Ecrire une fonction membre surdéfinisant l'opérateur <<, de telle sorte que c<<m ajoute au compte c le montant m.
4. Ecrire une fonction membre surdéfinisant l'opérateur >>, de telle sorte que c>>m retire du compte c le montant m.
5. Ecrire la fonction main () permettant de tester la classe compte ainsi définie.
6. Comment faut-il modifier la définition des opérateurs << et >> pour qu'ils puissent être utilisés sous les formes c<<m1<<m2<<m3 et c>>m1>>m2>>m3 ;

Exercice 3 :

1. Réaliser la classe **vecteur** permettant de manipuler des vecteurs à deux composantes de type réel. On prévoira :
    - Un constructeur permettant d'initialiser le vecteur, soit avec les deux composantes fournies explicitement à la déclaration, soit avec des composantes nulles.
    - Une fonction permettant de savoir si deux vecteurs ont mêmes composantes en utilisant une transmission par référence.
    - Une fonction permettant d'obtenir parmi deux vecteurs celui qui a la plus petite norme en prévoyant que le résultat soit renvoyé par référence et que l'argument explicite soit transmis par adresse.
    - Une fonction affiche permettant d'afficher un vecteur.
2. Ecrire la classe **vecteur3D** héritant de la classe **vecteur** et manipulant des vecteurs à trois composantes de type réel. Cette classe contient un constructeur et une fonction affiche redéfinies permettant d'afficher un vecteur 3D.
3. Ecrire la fonction main permettant de tester les deux classes.