TP 1

Notion de classe, constructeurs, héritage simple, classe « string » de la STL

1 Une première classe : la classe *CIndividu*

La classe *CIndividu* est une classe représentant une personne de façon généraliste. Elle possède deux données internes **privées**:

- le nom : une variable « chaîne de caractères » de type *string*.
- l'age : une variable de type *int*

Pour pouvoir utiliser le type *string* vous devez inclure le fichier "*include*" standard *string.h* :

```
#include <string>
using namespace std;
```

Avant d'utiliser un objet, il faut d'abord le construire.

1.1 Les constructeurs:

Il y a plusieurs façons pour construire et initialiser les données des objets d'une classe.

- Constructeur avec arguments :

  Il s'écrira (dans un fichier *CIndividu.cpp*) comme suit :

```
CIndividu::CIndividu(int age, string nom)
{
m_nom = nom;      // initialise la donnée membre donnant le nom
m_age = age;      // initialise la donnée membre donnant l'age
}
```

  Vous lui ferez appel ainsi (dans le *main( )*) :

```
CIndividu monIndividu(14,"Pif");    // crée un individu
```

- Constructeur sans argument : il s'écrira

```
CIndividu::CIndividu()
{
m_nom = "";       // initialise la donnée membre avec une chaine vide
m_age = 0;        // initialise la donnée membre avec un age fictif
}
```

  Vous lui ferez appel ainsi (dans le *main( )*) :

```
CIndividu monIndividu;  // crée un individu
```

  Pour lui donner un nom et un âge par la suite (dans le *main( )*), il y a plusieurs méthodes :

  - soit utiliser les fonctions *setAge( )* et *setNom( )* (cf. § 1.2),
  - soit faire une affectation à l'aide d'un constructeur avec arguments, appelé explicitement , comme suit :

```
monIndividu = CIndividu(13, "Paf");    // initialise un individu
```

Quel est l'intérêt d'un constructeur sans argument ? il en existe au moins deux :

- le premier est d'avoir une classe relativement souple d'emploi. Il arrive en effet que certaines données des objets créés ne soient pas encore connues au moment de la création.
- l'autre point est que parfois vous n'avez pas le choix. Par exemple, si vous créez des tableaux d'objets comme nous le ferons bientôt, le compilateur fait automatiquement appel au constructeur sans arguments. Par exemple:

```
CIndividu tableau[10];            //construit un tableau de 10 objets
tableau[0]=CIndividu(12,"Pouf");// init du 1er élément du tableau
```

Dans les exemples précédents nous avons créé des objets "statiques" (en parlant de la gestion mémoire), mais on peut aussi créer des objets de façon dynamique, à l'aide de la fonction d'allocation "*new*", qui remplace en C++ le "*malloc*" du C.

"*new*" renvoie un pointeur sur un objet. On le construit de la façon suivante :

```
CIndividu* pMonIndividu = new CIndividu(10, "Toto");
//ou bien
CIndividu* pMonIndividu = new CIndividu();
*pMonIndividu = CIndividu(38, "Titi")
```

Attention ! Il ne faut pas oublier de libérer l'espace mémoire alloué une fois que vous ne vous servez plus de votre objet:

```
delete pMonIndividu;
```

Le code à écrire :

A présent vous pouvez créer la classe *CIndividu* (fichiers *CIndividu.h* et *CIndividu.cpp*). Créez également un fichier "*exercice2.cpp*" qui contiendra le *main( )* du programme. Vous construirez des objets *CIndividu* dans ce *main( )* afin de tester votre classe.

1.2 Les méthodes de type "*set*" et "*get*"

Comment faire pour lire et afficher les données d'un objet ou pour les modifier ?

Si les données *m_nom* et *m_age* étaient déclarées "*public*" dans la classe *CIndividu*, on pourrait directement les utiliser comme variables et donc les lire et les modifier, comme par exemple :

```
monIndividu.m_age = 20;
```

Ce n'est pas la méthode préconisée par l'approche objet, où on doit respecter la notion d'encapsulation des données. Les données *m_nom* et *m_age* ont donc été déclarées "*private*" dans notre classe. Pour y accéder, en lecture ou en écriture, on doit prévoir des méthodes dites "accesseurs", méthodes "*get*" pour la lecture et "*set*" pour l'écriture.

Voici l'implémentation d'une méthode "*set*" pour l'âge :

```
void CIndividu::setAge(int age)
{
      // si l'age est supérieur à 0 on le prend
      if( age > 0 ) m_age = age;
      // sinon on ne fait rien.
}
```

Voici l'implémentation d'une méthode "*get*" pour le nom :

```
string CIndividu::getNom()
{
      return m_nom;
}
```

Ces fonctions étant généralement très courtes (1 ou 2 lignes maximum), elles peuvent être écrites à l'intérieur même de la déclaration de la classe. Elles tirent partie alors d'une optimisation du compilateur car elles sont automatiquement déclarées *inline*. Cela permet d'accélérer l'exécution de la fonction. Cependant, pour des fonctions plus longues, définissez les toujours à l'extérieur du bloc de déclaration de classe.

Le code à écrire :

A présent, rajoutez les méthodes qui vont vous permettre de modifier et lire les données des objets de notre classe *CIndividu*.

Créez dans le *main( )* le code qui permet d'afficher les données d'un objet de la classe et de modifier ces données.

2 Des classes spécialisées.

Nous allons créer deux classes filles de notre classe *CIndividu*. Elles vont hériter des données et des méthodes de leur classe mère, et y rajouter des données et/ou méthodes supplémentaires. Habituellement, pour créer une classe fille, on utilise la syntaxe suivante :

```
class ClasseFille : public ClasseMere
{
//reste de déclaration de la classe
}
```

2.1 Une classe *CProfesseur* dérivée de *CIndividu*.

La classe *CProfesseur* est caractérisée par un salaire de type "*int*". On peut lui attribuer un constructeur comme suit:

```
CProfesseur::CProfesseur(int salaire)
{
      m_salaire = salaire;
}
```

Dans ce cas les données héritées (*m_nom* et *m_age*) seront initialisées automatiquement par le constructeur sans arguments de la classe *CIndividu*. Pour utiliser le constructeur avec arguments que vous avez défini dans la classe mère, vous devez utiliser la syntaxe suivante :

```
CProfesseur::CProfesseur(int age,string nom,int salaire):CIndividu(age, nom);
{
      m_salaire = salaire;
}
```

2.2 Une classe *CEleve* dérivée de *CIndividu*.

La classe *CEleve* est caractérisée par une année d'étude de type "*int*" (valeur de 1 à 5).

Le code à écrire :

Construisez ces classes. Mettez y tout ce qu'il faut pour qu'elles fonctionnent et testez les.

Créer un premier individu avec le constructeur par défaut.

Créer un second individu avec le constructeur surchargé.

Créer un tableau de 10 individus et rentrer un individu dans le tableau.

Affichez les caractéristiques des individus créés avec l'instruction cout.

Créer un pointeur sur un individu en utilisant le constructeur par défaut et le constructeur surchargé

Créer un professeur avec un seul argument et un professeur avec trois arguments.

Créer un élève avec un seul argument et un élève avec trois arguments.