Notion de Classe

Dans cette section, nous abordons véritablement les possibilités de la P.O.O en C++, qui comme introduit initialement est entièrement basée sur le concept de Classe.

Une **classe** est un type de données dont le rôle est de rassembler sous un même nom à la fois données et traitements. La notion de classe n'est pas trop loin de la notion de structure:

- La déclaration d'une classe est presque similaire de celle d'une structure.
  - Il suffit de remplacer le mot clé ***struct*** par le mot clé ***class***;
  - Puis de préciser les fonctions ou données membres publics avec le mot clé ***public*** et les membres privés avec le mot clé ***private***.
- Syntaxe:

```
class X
{
        private :
            ...
        public :
            ...
        private :
            ...
};
```

Classe

Déclaration

En C++, la programmation d'une classe se fait en trois phases: déclaration, définition et utilisation.

- **Déclaration**: c'est la partie interface de la classe. Elle se fait dans un fichier dont le nom se termine par .h ou .hpp, .H, ou .h++ (appelé fichier d'entête).
  - La syntaxe est la suivante:

```
class Nom_de_la_classe {
        public:
        // déclarations des données et fonctions-membres publiques
        private:
        // déclarations des données et fonctions-membres privées
};
```

  - **Exemple:**

```
// Personne.hpp
class Personne {
        public:
        Personne(string, string);
        void toString();
        private:
        string p_nom;
        string p_prenom;
};
```

Classe

Définition

- **Définition**: c'est la partie implémentation de la classe. Elle se fait dans un fichier dont le nom se termine par .cc, .c++, .c ou .cpp. Ce fichier contient les définitions des fonctions-membres de la classe.

  - **Exemple**: //Personne.cpp

```
#include <iostream.h>
#include "Personne.hpp"

// constructeur (Il permet d'initialiser une nouvelle personne)
Personne::Personne(string prenom, string nom){
        p_prenom = prenom;
        p_nom = nom;
}

// une méthode (fonctions-membres)
void Personne::toString(){
        cout <<"Je m'appel"<<p_prenom<<" "<<p_nom<<endl;
}
```

Classe

Utilisation

- **Utilisation**: Elle se fait dans un fichier dont le nom se termine par .cc, .c++, .c ou .cpp. Ce fichier contient le traitement principal (la fonction ***main***).

  - **Exemple**: // test.cpp

```
#include <iostream.h>
#include "Personne.hpp"

// traitement principal
void main(){
        //  appel implicite du constructeur
        Personne p("Nabi" , "Barro");

        p.toString();
}
```

Classe

Constructeurs et destructeurs

- Un **constructeur** est une fonction-membre déclarée du même nom que la classe, et sans type.

  Syntaxe : Nom_de_la_classe(<paramètres>);

  - **Fonctionnement**: à l'exécution, l'appel au constructeur produit un nouvel objet de la classe, dont on peut prévoir l'initialisation des données-membres dans la définition du constructeur.

    Exemple: personne p("Nabi" , "Barro");

  - Dans une classe, il peut y avoir plusieurs constructeurs à condition qu'ils différent par le nombre ou le type des paramètres. *Un constructeur sans paramètre s'appelle constructeur par défaut*.

- Un **destructeur** est une fonction-membre déclarée du même nom que la classe mais précédé d'un tilde (~) et sans type ni paramètre.

  Syntaxe : ~Nom_de_la_ classe();

  - **Fonctionnement**: à l'issue de l'exécution d'un bloc, le destructeur est automatiquement appelé pour chaque objet de la classe déclaré dans ce bloc. Cela permet par exemple de programmer la restitution d'un environnement, en libérant un espace mémoire alloué par l'objet.

Classe

Constructeurs et destructeurs

L'exemple ci-dessous est un petit programme mettant en évidence les moments où sont appelés respectivement le constructeur et le destructeur d'une classe.

- Nous créons une classe test définissant que ces deux fonctions membres et une donnée membre ***num*** qui sera initialisée par le constructeur nous permettant d'identifier l'objet en question.

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

class test
{
    public :
        int num ;
        test (int) ; // déclaration constructeur
        ~test () ; // déclaration destructeur
} ;
```

```
test::test (int n)
{
    num = n ;
    cout << ''Appel du constructeur, avec num='' << num<<endl;
}
```

```
test::~test ()
{
    cout << ''Appel du destructeur, avec num='' << num<<endl;
}
```

Classe

Constructeurs et destructeurs

- Nous créons des instances de type test à deux endroits différents: dans la fonction ***main*** d'une part, dans une fonction ***fct*** appelée par ***main*** d'autre part.

```
main()
{
        void fct (int) ;
        test a(1) ;

        for (int i=1 ; i<=2 ; i++)
             fct(i) ;
}

void fct (int n)
{
    test t(2*n) ;
}
```

```
-> Appel du constructeur, avec num=1
-> Appel du constructeur, avec num=2
-> Appel du destructeur, avec num=2
-> Appel du constructeur, avec num=4
-> Appel du destructeur, avec num=4
-> Appel du destructeur, avec num=1
```

Classe

Constructeurs et destructeurs – quelques règles usuelles

- Un **constructeur** peut éventuellement comporter un nombre quelconque d'arguments.
- un **constructeur** n'a pas de type de retour et par conséquent, ne renvoie pas de valeur (la présence de void est dans ce cas une erreur).
- Un **destructeur** n'a pas de type de retour et donc, ne renvoie pas de valeur. Ici aussi, la présence de void est une erreur.
- Les **constructeurs** et **destructeurs** peuvent être **publics** ou **privés**. En pratique, à moins d'avoir de bonnes raisons de faire le contraire, il vaut mieux les rendre **publics**.

Classe

Visibilité des membres d'une classe

La visibilité des attributs et méthodes d'une classe est définie dans l'interface de la classe grâce aux mots-clés **public**, **private** ou **protected** qui permettent de préciser leurs types accès.

- **public**: autorise l'accès pour tous. Exemple: le constructeur ainsi que la méthode toString() de l'exemple précèdent peuvent être utilisés partout sur une instance de **personne**.
- **private**: restreint l'accès aux seuls corps des méthodes de cette classe. Sur l'exemple précédent, les attributs privés p_prenom et p_nom ne sont accessibles sur une instance de **personne** que dans les corps des méthodes de la classe. Ainsi, la méthode toString() a le droit d'accès sur ses attributs p_prenom et p_nom. Elle aurait aussi l'accès aux attributs p_prenom et p_nom d'une autre instance de **personne**.
- **protected**: comme private sauf que l'accès est aussi autorisé aux corps des méthodes des classes qui héritent de cette classe.

Classe

Objet

Un objet est une instance d'une classe (c'est-à-dire, une variable dont le type est une classe). Un objet occupe donc de l'espace mémoire. Il peut être alloué :

- ***statiquement***: dans ce cas, on met le nom de la classe suivi du nom de l'objet et éventuellement suivi par des arguments d'appel donnés à un constructeur de la classe.

  Exemple: Personne p("Nabi" , "Barro");

- ou ***dynamiquement***: dans ce cas, un pointeur sur la zone mémoire ou l'objet a été alloué est retourné. Lorsqu'on n'en a plus besoin, on le libère avec l'opérateur *delete*.

  Exemple:

```
personne * p = new personne("Nabi" , "Barro");
p->toString();
delete p;
```

Classe

Exemple complet

```
#include <iostream>
Using namespace std;

// déclaration de la classe
class personne
{
        public:
        personne(string, string);
        void toString();

        private:
        string p_nom;
        string p_prenom;
  };

// définition des membres publics
// constructeur
personne::personne(string prenom, string nom){
        p_prenom = prenom;
        p_nom = nom;
}

// une méthode
void personne::toString(){
        cout <<"Je m'appel"<<p_prenom<<" "<<p_nom<<endl;
}
```

statiquement

```
// Utilisation de la classe personne
void main(){
        //  appel implicite du constructeur
        personne p("Nabi" , "Barro");
        p.toString();
}
```

dynamiquement

```
// Utilisation de la classe personne
void main(){
        //  appel implicite du constructeur
        personne * p = new personne("Nabi" , "Barro");
            p->toString();
            delete p;
}
```

Classe et Objet

EXERCICES D'APPLICATIONS

- Application 28:

  Réaliser une classe *point* permettant de manipuler un point d'un plan.

  On prévoira :
  - un constructeur recevant en arguments les coordonnées (*float*) d'un point ;
  - une fonction membre *deplace* effectuant une translation définie par ses deux arguments (*float*) ;
  - une fonction membre *affiche* se contentant d'afficher les coordonnées cartésiennes du point.

  Les coordonnées du point seront des membres donnée privés.

  On écrira séparément :
  - un fichier source constituant la *déclaration* de la classe ;
  - un fichier source correspondant à sa *définition*.
  - un petit programme d'essai (*main*) déclarant un point, l'affichant, le déplaçant et l'affichant à nouveau.

- Application 29:

  Réaliser une classe *point*, analogue à la précédente, mais ne comportant pas de fonction *affiche*. Pour respecter le principe d'encapsulation des données, prévoir deux fonctions membre publiques (nommées *abscisse* et *ordonnee*) fournissant en retour l'abscisse et l'ordonnée d'un point. Adapter le petit programme d'essai précédent pour qu'il fonctionne avec cette nouvelle classe.