Virtualité et polymorphisme

Un **objet polymorphe** est un objet qui peut prendre différentes formes. En C++, il dispose de méthodes virtuelles et conserve des informations sur son type réel. A l'exécution, la fonction appelée sur un objet polymorphe dépendra du type réel de l'objet.

Pour montrer le fonctionnement du polymorphisme, nous allons utiliser :

- **la notion d'héritage** déjà vue entre la classe Document et ses classes dérivées Livre et Article,
- **la notion de virtualité** implantée par les fonctions virtuelles,
- **les pointeurs (**pour la notion de liaison dynamique**).**

.

1 Créez et testez la classe Bibliothèque

On souhaite construire une classe **Biblio** permettant de stocker un certain nombre de documents (que ce soient des articles, des livres …) dans une liste (vous utiliserez la classe "list" de STL, voir le rappel en fin d'énoncé) et proposant différentes méthodes pour la gestion des documents qu'elle contient, notamment :

- l'ajout d'un document particulier en fin de liste,
- la recherche d'un document particulier à partir de son titre (si plusieurs documents possèdent un même titre, le premier sera retourné,
- l'affichage du contenu de la bibliothèque.

- Complétez la classe **Biblio** ci-dessous (vous remplacerez les ??? avec le type d'objet qui convient).

```
#include <list>
#include <iostream>
#include "Document.h"
using namespace std;

class Biblio  {
   list <???> tab;
   public:
        void ajouter (Document *D);
        ??? rechercher(string T);
        void afficher();
};
```

- Ecrire le code des méthodes dont le prototype figure dans la classe **Biblio**.

2 Notion de classe abstraite

- Ajoutez dans la classe Document une **fonction virtuelle pure** (fonction n'ayant pas de code) pour calculer le coût du Document : la classe Document devient alors une classe abstraite.
- Codez cette fonction de coût dans les classes dérivées Livre et Article.
- Testez alors cette fonction.

Rappels : utilisation du type list de la STL

```
#include <list>
using namespace std ;
list <int> l ;
l.push_front(1) ;
l.push_back(2) ;
for (list<int>::iterator it = l.begin(); it !=l.end(); it++)
        cout << (*it);
```