

## TD Langage C++ n°3

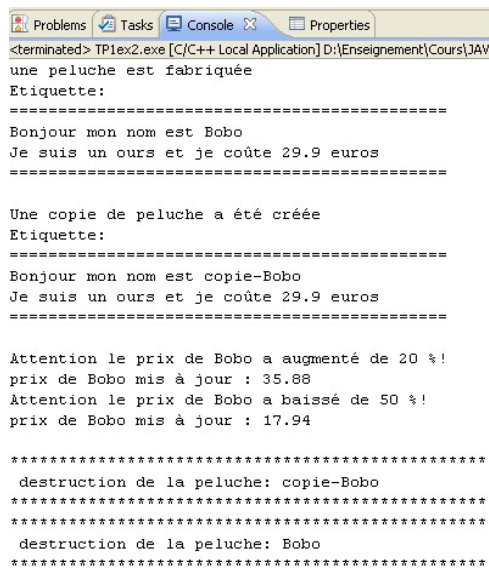
### Notion de Classe et d'objet

#### Forme Canonique de Coplien- Surdéfinition d'opérateurs - Agrégation

##### Exercice n° 1 :

Reprendre l'exercice 2 du TD2 et Imaginez une solution pour créer une copie d'un objet peluche (dont seul le nom sera différent (vous rajouterez la chaîne de caractères " copie-" devant le nom de l'objet copié

##### Exemple d'exécution Programme



```
<terminated> TP1ex2.exe [C/C++ Local Application] D:\Enseignement\Cours\JAV
une peluche est fabriquée
Etiquette:
=====
Bonjour mon nom est Bobo
Je suis un ours et je coûte 29.9 euros
=====

Une copie de peluche a été créée
Etiquette:
=====
Bonjour mon nom est copie-Bobo
Je suis un ours et je coûte 29.9 euros
=====

Attention le prix de Bobo a augmenté de 20 %!
prix de Bobo mis à jour : 35.88
Attention le prix de Bobo a baissé de 50 %!
prix de Bobo mis à jour : 17.94

*****
destruction de la peluche: copie-Bobo
*****
destruction de la peluche: Bobo
*****
```

##### Exercice n° 2 :

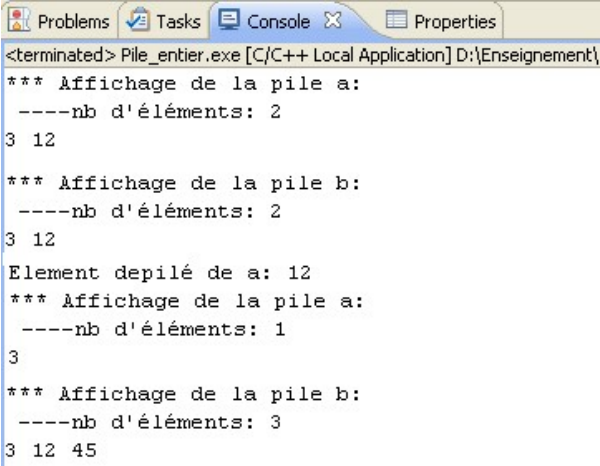
- Ecrire une classe nommée PileEntier permettant de gérer une pile d'entiers, qui seront conservés dans un tableau d'entiers alloués dynamiquement. La classe comportera les fonctions membres suivantes :
  - constructeur avec 1 paramètre (nb d'éléments max de la pile), ou sans paramètre (par défaut nb d'éléments max de la pile)
  - destructeur
  - void empile(int e) : ajoute l'entier e à la pile
  - int depile() : fournit la valeur de l'entier situé en haut de la pile, en le supprimant de la pile
  - bool pleine() : renvoie true si la pile est pleine false sinon
  - bool vide() : renvoie true si la pile est vide false sinon
  - void affiche : qui permet d'afficher le nombre d'éléments présents dans la pile, et les entiers stockés dans la pile
- Vous écrirez une fonction main utilisant des objets automatiques et dynamiques du type PileEntier
- Que se passe-t-il si on écrit des déclarations et instructions du type

```
PileEntier a(10) ;
a.empile(5) ;
PileEntier b(a) ;
```
- Corriger votre programme si nécessaire.

### Exemple d'exécution du programme principal suivant

```
int main() {
    PileEntier a(3); //une pile de 3 entiers

    a.empile(3);
    a.empile(12);
    cout<<"*** Affichage de la pile a: \n";
    a.affiche();
    PileEntier b;
    b=a;
    cout<<"*** Affichage de la pile b: \n";
    b.affiche();
    b.empile(45);
    int nb;
    nb= a.depile();
    cout<<"Element depilé de a: "<<nb<<endl;
    cout<<"*** Affichage de la pile a: \n";
    a.affiche();
    cout<<"*** Affichage de la pile b: \n";
    b.affiche();
}
```



```
<terminated> Pile_entier.exe [C/C++ Local Application] D:\Enseignement\
*** Affichage de la pile a:
----nb d'éléments: 2
3 12
*** Affichage de la pile b:
----nb d'éléments: 2
3 12
Element depilé de a: 12
*** Affichage de la pile a:
----nb d'éléments: 1
3
*** Affichage de la pile b:
----nb d'éléments: 3
3 12 45
```

e) Surdéfinissez les opérateurs suivants :

= : pour pouvoir faire l'affectation de 2 objets de type PileEntier

< : de telle sorte que p<n ajoute la valeur entière n sur la pile p

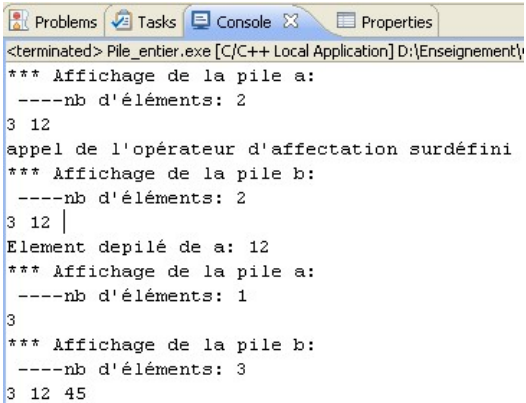
> : de telle sorte que p>n supprime la valeur du haut de la pile et la place dans n.

### Exemple d'exécution du programme principal suivant

```
int main() {
    PileEntier a(3); //une pile de 3 entiers

    a<3;

    a<12;
    cout<<"*** Affichage de la pile a: \n";
    a.affiche();
    PileEntier b;
    b=a;
    cout<<"*** Affichage de la pile b: \n";
    b.affiche();
    b<45;
    int nb;
    a>nb;
    cout<<"Element depilé de a: "<<nb<<endl;
    cout<<"*** Affichage de la pile a: \n";
    a.affiche();
    cout<<"*** Affichage de la pile b: \n";
    b.affiche();
}
```



```
<terminated> Pile_entier.exe [C/C++ Local Application] D:\Enseignement\
*** Affichage de la pile a:
----nb d'éléments: 2
3 12
appel de l'opérateur d'affectation surdéfini
*** Affichage de la pile b:
----nb d'éléments: 2
3 12 |
Element depilé de a: 12
*** Affichage de la pile a:
----nb d'éléments: 1
3
*** Affichage de la pile b:
----nb d'éléments: 3
3 12 45
```

### Exercice 3:

a) Reprenez la **classe Point** donné en exemple dans le cours, modifiez le type des attributs pour que les coordonnées puissent être des réels, et ajoutez y les méthodes d'accès et de modification des attributs (get, set), la méthode distance qui permet de calculer la distance entre 2 points (le paramètre implicite et explicite).

La classe point aura donc en dehors des constructeurs, les méthodes suivantes: les méthodes get et set pour chacun des attributs, la méthode translation, la méthode distance et la méthode affiche.

NB : vous incluez la librairie <cmath> pour utiliser la fonction sqrt (racine carrée)

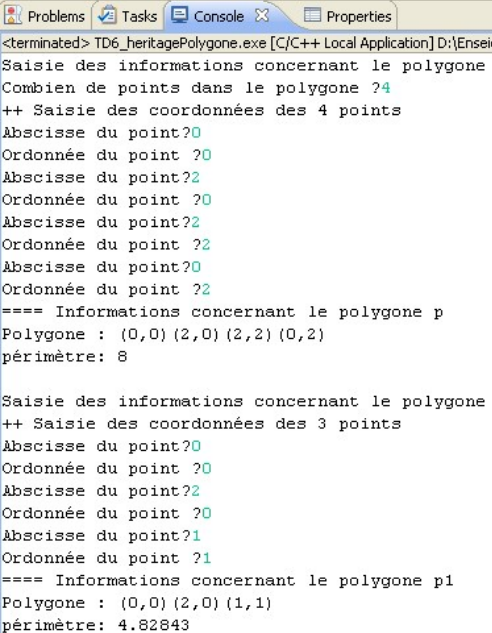
b) Définissez la **classe Polygone**

Un polygone est caractérisé par son nombre de points et par la liste de ses sommets successifs. Par exemple un rectangle est défini par les coordonnées de ses quatre sommets (x1,y1), (x2,y2), (x3,y3) et (x4,y4)

Cette liste de points successifs sera stockée dans un tableau alloué dynamiquement.

La classe Polygone disposera en dehors de la forme canonique de Coplien , des méthodes suivantes :

- méthode affiche() qui permettra un affichage du type  
Polygone : [(0.0,0.0) (2.0,0.0) (2.0,2.0) (0.0,2.0) ]
- méthode perimetre() qui permettra de calculer le périmètre du polygone



```
<terminated> TD6_heritagePolygone.exe [C/C++ Local Application] D:\Enseig
Saisie des informations concernant le polygone
Combien de points dans le polygone ?4
++ Saisie des coordonnées des 4 points
Abscisse du point?0
Ordonnée du point ?0
Abscisse du point?2
Ordonnée du point ?0
Abscisse du point?2
Ordonnée du point ?2
Abscisse du point?0
Ordonnée du point ?2
==== Informations concernant le polygone p
Polygone : (0,0) (2,0) (2,2) (0,2)
périmètre: 8

Saisie des informations concernant le polygone
++ Saisie des coordonnées des 3 points
Abscisse du point?0
Ordonnée du point ?0
Abscisse du point?2
Ordonnée du point ?0
Abscisse du point?1
Ordonnée du point ?1
==== Informations concernant le polygone p1
Polygone : (0,0) (2,0) (1,1)
périmètre: 4.82843
```