Examen

Exercice 1 Ecrire une fonction permettant d'ajouter une valeur fournie en argument à une variable fournie également en argument, par exemple (`n` et `p` étant entiers): `ajouter (2*p+1, n);` ajoutera la valeur de l'expression `2*p+1` à la variable `n`.

Exercice 2 Qu'affiche ce programme?

```
#include <iostream>
using namespace std;
main()
{ int t[4]={10,20,30,40};
  int * ad[4];
  for(int i=0; i<4; i++) ad[i]=t+i;
  for(int i=0; i<4; i++) cout << * ad[i] << " "; cout << "\n";
  cout << * (ad[1]+1) << " " << * ad[1] + 1 << "\n";
}
```

Exercice 3 Faire un programme affichant les n premières lignes du triangle de Pascal avec n entré par l'utilisateur.

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

Exercice 4 Créer une classe `point` simulant un point du plan dont les coordonnées de type `double` seront des attributs privés et munissez la de exactement deux méthodes : un constructeur et une fonction membre `proche` renvoyant `true` si la distance du point appelant à celle d'un point en paramètre est inférieure à 1 (sans utiliser `include`). Faire une fonction binaire non-membre `proche2` avec la même fonctionalité.

Exercice 5 Soit une classe `vecteur3d` définie par:

```
class vecteur3d
{ float x,y,z;
 public:
  vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0){x=c1; y=c2; z=c3;}
  ...
};
```

Faire une fonction membre `coincide` permettant de savoir si deux vecteurs ont les mêmes composantes:

1) en utilisant la transmission par valeur;

2) en utilisant la transmission par adresse;

3) en utilisant la transmission par référence.

Faire une fonction amie `normax` permettant d'obtenir, parmi deux vecteurs, celui qui a la plus grande norme (euclidienne).

Exercice 6 Soit une classe `complex` avec trois constructeurs:

```
class complex
 double re,im;
public:
 complex(){re=0.0; im=0.0;}
 complex(double r){re=r; im=0.0;}
 complex(double r,double i){re=r; im=i;}
 ...
};
```

1) Remplacer les trois constructeurs par un seul constructeur équivalent.

2) Ajouter l'opérateur += à `complex` comme une fonction membre avec valeur de retour par référence.

3) Faire une fonction non-membre pour l'opérateur binaire + pour deux complexes en utilisant l'opérateur += du (2).

Exercice 7 Créer une classe `vect` simulant un vecteur de l'espace euclidien de dimension variable. On mettra en attribut privé `int dim` la dimension du vecteur et `double * adr` l'adresse du vecteur (dont les composantes seront de type `double`). Faire les méthodes suivantes pour gérer la partie dynamique et libérer l'espace mémoire : un constructeur prenant la dimension en paramètre, un destructeur, un constructeur de recopie et l'opérateur = surdéfini.

Problème 1 Faire une classe `point` simulant les points du plan euclidien munie d'une méthode `affiche` affichant les coordonnées d'un point. Faire une classe `boule` comprenant deux attributs privés `point centre` et `double rayon` et une méthode `affiche` affichant les coordonnées du centre et le rayon. Faire ensuite trois classes `boule1`, `boule2` et `bouleINF` dérivées de `boule` munie d'une méthode `appartient` disant (sans utiliser `include`) si un point en paramètre est ou non dans la boule, chacune des boules étant définie respectivement avec la norme 1 ($\sum_i |x_i|$), la norme 2 ($(\sum_i x_i^2)^{1/2}$) et la norme ∞ ($\max_i |x_i|$).

Un *amas* est une union de boules (euclidienne) du plan. Faire une classe canonique `amas` dérivée de `boule2` ayant en attribut privé `boule2 * am` poitant sur un tableau dynamique de boules. La munir d'une méthode `intersecte` disant si deux boules s'intersectent ou non, et d'une méthode `connex` testant si l'amas et connexe ou non.

Faire des classes équivalentes à `amas` pour les deux autres normes.