université de BORDEAUX	**ANNEE UNIVERSITAIRE 2019/ 2020** **S1 DE PRINTEMPS** **PARCOURS / ETAPE : L3 MIAGE** **Code UE:** 4TYG602, 4TBC801 **Devoir Maison** **Date :5.05.2020 Rendu 7.05.2020 à 17h30** Documents : autorisés Auteur M/Mme : J. Benois-Pineau	**Collège Sciences et technologies**

Rendu du Devoir Maison. Produire un fichier .zip contenant

- **le code sous forme des fichiers .hpp et .cpp**
- **un court fichier – texte au format .pdf avec les l'explications de vos solutions, les commandes de compilation pour les exercices ou la compilation et/ou création de l'executable du codé est nécessaire.**

Vous veillerez au niveau de commentaires nécessaires à mettre dans votre le code. La lisibilité du code est un critère d'évaluation du devoir également.

Exercice 1 : **Fondements de la modélisation Objet. Protection**

Dans un système de gestion du personnel vous avez la classe *Personne* suivante présentée ci-dessous dans le fichier Personne.hpp.

```
#ifndef PERSONNE_H
#define PERSONNE_H
#include <iostream>
#include <string>
using namespace std;

class Personne{

private:

        string Nom;
        string Prenom;
        string Num_INSEE;
        string Date_Naissance;

public:

        Personne();
        Personne(string,string,string,string);
        string getNom();
        string getPrenom();
        string get Num_INSEE();
        string getDate_Naissance();
        void affiche();

};
#endif
```

Pour le système de gestion des patients aux urgences dans un hôpital, il est nécessaire de développer une classe *Patient* qui héritera de la classe *Personne*. Cette classe en plus de données-membres de la classe *Personne* comportera une donnée caractérisant l'état d'urgence pour le traitement du patient. Ceci est un entier qui peut avoir les valeurs : 3 (l'état grave, urgence absolue), 2 (l'état intermédiaire), 1 (pathologie légère).

La classe comporte une méthode homonyme *void affiche() ;* ainsi que les fonctions d'accès à toutes les données membres.

- Comment faut-il modifier la classe *Personne* pour en faire hériter la classe *Patient* ;
- Proposez la déclaration de la classe *Patient* dans un fichier .hpp (Ne développez pas les corps des fonctions – membres, pas de fichier .cpp)
- Proposez un programme main qui utilise les deux classes : *Personne* et *Patient*, c'est à dire instancie des objets de ces deux classes, et affiche le contenu des objets. *Compilez* uniquement le programme *main*.

(4 pts)

Exercice 2. Classes abstraites

On se propose de développer les classes suivantes :

```
class Forme{
protected:
        double x;
        double y;
public:
        Forme(double _x=0.0, double _y=0.0){x=_x;y=_y;}
        virtual void affiche()=0;
        void deplace(double dx,double dy){x=x+dx;y=y+dy;}
};
class Point:public Forme{
public:
        Point(double _x=0.0, double _y=0.0):Forme(_x,_y){}
         virtual void affiche() {cout<<"Affiche point x="<<x<<"
y="<<y<<endl;}

};
class Cercle:public Point{
protected:
        double rayon;
public:
        Cercle(double _x, double _y, double _r):Point(_x,_y){rayon=_r;}
        virtual void affiche(){cout<<"Affiche Cercle x="<<x<<" y="<<y<<"
r="<<rayon<<endl;}

};
class Sphere:public Cercle{
protected:
        double z;
public:
    Sphere(float _x, float _y, float _z, float _r):Cercle(_x,_y,_r){z=_z;}
        virtual void affiche(){cout<<"Affiche Sphere"<<" x="<<x<<" y="<<y<<"
z="<<z<<" rayon="<<rayon<<endl;}
        void deplace(double dx, double dy, double dz){
Cercle::deplace(dx,dy);z=z+dz;}
};
```

Vous avez écrit le programme main suivant :

```
int main(){
Forme F(2.1, 3.2);
Cercle C(4.2,5.3,5.0);
Sphere *PSphere;
Cercle *PCercle;
PCercle = &C;
PSphere = PCercle;
```

```
PSphere->affiche();
return 0;
}
```

Le compilateur vous communique une erreur. Quel est le sens de son message ? Corrigez le code. Donnez le résultat de l'exécution du programme après la correction. Insérez la saisie de l'écran de l'exécution dans le fichier texte (cf. les consignes Page 1)

(3 pts)

Exercice 3. Meta-programmation

On se propose de développer une liste ordonnée à l'aide de templates. Nous allons appeler cette classe *ListeOrd*. La liste est simplement chaînée avec le chainage en arrière (cf. Figure 1). Les données à insérer peuvent être comparées entre elles à l'aide de l'opérateur « <» et à l'aide d'un deuxième opérateur « = ». Chaque nouvelle donnée est insérée « à sa place » c'est à dire juste après le dernier élément dans la liste qui est supérieur ou égal l'élément à insérer.

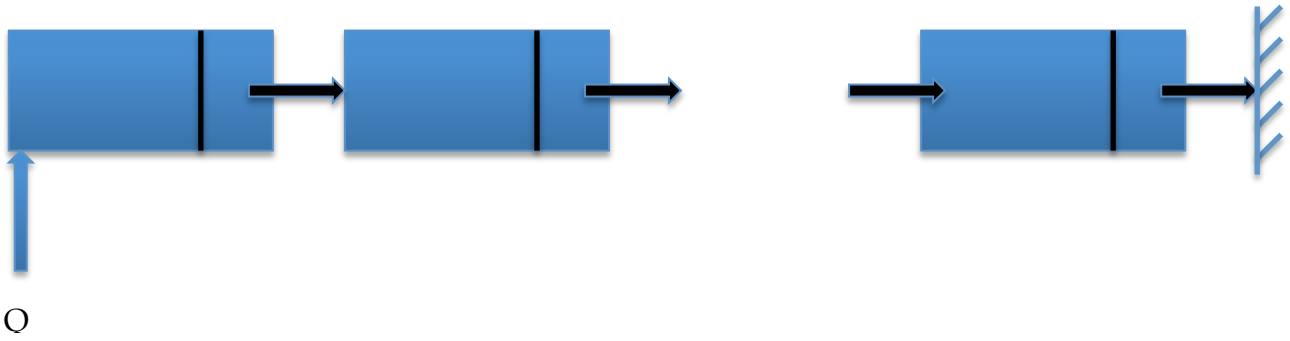

Figure 1. Structure de la Liste.

Les méthodes proposées sont :

```
void InsertInPlace(TypeE& data) ;//insère data à la place
 void popFirst() ;//retire le premier élément
 bool Is_empty() ;//vérifie si la liste est vide
 display () ;//affiche le contenu de la liste
 un constructuer à vide ;
 le destructeur ;
```

Développez la classe *ListeOrd* complétement avec toutes les méthodes dans un même fichier .hpp. Dans un fichier .cpp, développez le programme *main*, instanciez le template avec des entiers, insérez 3 éléments dans la liste, affichez, retirez deux éléments de la liste, affichez. Compilez le programme, créez un exécutable et exécutez le programme. Insérez la saisie de l'écran de l'exécution dans le fichier texte (cf. les consignes Page 1)
(9 pts)

Exercice 4. Surcharge des opérateurs

Compléter la classe *Patient* de l'Exercice 1 en y rajoutant la surcharge des opérateurs « < » et « = ».
Deux patients P1 et P2 sont « égaux » si les valeurs de leur état d'urgence sont égales. P1<P2 si la valeur de l'état de l'urgence du patient P1 est inférieure à la valeur de l'état de l'urgence du patient P2.
Chargez la liste – template de l'Exercice 3 par le type (classe) *Patient* dans un programme *main*. Ce programme rajoute 5 patients dans la liste avec les valeurs de l'état d'urgence, affiche le contenu de la liste, retire le premier élément, affiche le contenu de la liste. Exécutez le programme et insérez la saisie de l'écran de l'exécution dans votre fichier texte (cf. les consignes sur la page 1).
(4 pts)