CY UNIVERSITÉ	**ING2 GI FA : PROGRAMMATION C++** **EXAMEN SESSION NORMALE CORRIGE** **DUREE : 2H, AUCUN DOCUMENT N'EST AUTORISE**
Formation: ING2-GI-FA	Année : 2024-2025

Modalités

- Vous devez rédiger votre copie à l'aide d'un stylo à encre exclusivement.
- Pour ceux en distanciel, envoyer une image de votre copie papier
- Toutes vos affaires (sacs, vestes, trousse, téléphone, etc.) doivent être placées par terre.
- Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.
- Aucune sortie n'est autorisée avant une durée incompressible d'une heure.

Exercice 1 (2 pts) : Soit le programme principal suivant dans le fichier nommé calcul.cpp:

Soient les commandes suivantes dans un terminal
g++ calcul.cpp –o calcul
./calcul 1 -2 0

1. Donner la sortie complète du programme dans les 3 cas suivants de l'implémentation de la fonction ajouter :
   1. **void** ajouter (**int** a, **int** b, **int** c) { c = a+b; }
      apres appel de ajouter
      c = 0
   2. **void** ajouter (**int** a, **int** b, **int** & c) { c = a+b; }
      apres appel de ajouter
      c = -1

2. Comment on appelle chacun de ces passages de paramètres ?
   pour le paramètre c, l'appel se fait par :
   1. par copie valeur,
   2. par copie de référence
   pour les paramètres a et b, l'appel se fait par copie de valeur dans les 2 cas

Exercice 2 (6 pts) : soit la classe Image suivante

```
class Image {
 short **a;
 int nlignes, ncols;
public:
 Image();
 Image(int m, int n);
 Image(const Image &);
 Image & operator =(const Image &);
 ~Image();
 static short* & operator[](int i) { return a[i]; };
};
```

1. Expliquer la méthode à la ligne 6 et donner son code complet (voir le cours)
2. Expliquer la méthode à la ligne 7 et donner son code complet (voir le cours)
3. Expliquer la méthode à la ligne 8 et donner son code complet (voir le cours)
4. Expliquer la méthode à la ligne 9 et donner son code complet (voir le cours)
5. Que peut-on dire de cette classe ? justifier votre réponse
6. Comment appelle-t-on l'**operator** défini à la ligne 10 et que fait-il ?
   Opérateur de surcharge d'indice []
   short* & operator [] (int i) {return image[i];};
   Renvoie la référence (&) de l'endroit où se trouve l'adresse (short*) de la ligne à la position i renvoyée par l'opérateur

7. A quoi sert ce mot clé **static**
   Mot clé static pour dire que c'est une méthode de classe et non pas une méthode d'instance

Exercice 3 : voir le cours polymorphisme P25 et P26 identique (4 pts)

```
1  #include<iostream>
2
3  using namespace std;
4
5  class Polygone
6  {
7  virtual void perimetre();
8  virtual void aire() const;
9  virtual void tauxOccupation(int= 0);
10 void espaceDiponible();
11 };
12
13 class Rectangle: public Polygone
14 {
15  virtual void perimetre();
16  virtual void aire();
17  virtual void perimetre() override;
18  virtual void tauxOccupation(double = 0.0);
19  virtual void aire() override;
20  virtual void tauxOccupation( double = 0.0 ) override;
21  void espaceDiponible() override;
22 };
```

On considère les lignes de 15 à 21

1. Dites d'abord si les instructions de 15-21 sont [justes/Fausse] puis justifiez votre réponse (voir le cours)

Exercice 4 : simulation de la gestion des urgences d'un hôpital

On veut simuler le fonctionnement d'une urgence d'un hôpital, et pour cela, on vous demande d'implémenter une classe '*Malade'* et une classe '*FilePrioriteMalades'*. Les 2 classes dérivent de la classe '*Objet*' *fournie.*
Un malade est défini par son identifiant (id) et sa gravité (gravite) de son état de santé comprise de 1 à 5. La valeur 5 définit l'état le plus grave donc le plus urgent à prendre en charge.
Un malade arrive aux urgences, l'infirmière l'examine et le place dans la file d'attente '*FilePrioriteMalades'* composée d'une file de priorité (la priority_queue).
Pour utiliser la priority_queue, on utilise la classe foncteur 'Comparaison' avec un opérateur de comparaison de la gravité deux malades et retourne un booléen.

Votre programme sera donc constitué de 4 classes : Malade, FilePriorieMalades qui implémente une priority_queue et des 2 classes fournies (Comparaison, Objet).

```
// Foncteur de comparaison selon les priorités:
class Comparaison {
public:
bool operator()(const Malade &m1, const Malade &m2) { return m1.gravite < m2.gravite ; }
};

// Construit la classe FilePrioriteMalades avec une file de priorité:
priority_queue<Malade, vector<Malade>, Comparaison> pq;
```

```
//fichier objet.h
class Objet {
  public:
    virtual void afficher () =0 ;
};
```

Simulation (la méthode main)

1. Dans la classe Objet commenter la méthode afficher().

2. Le programme doit générer des urgences d'une capacité fixe de 25 malades. Pour chaque Malade, le programme principal fixe la gravité de chaque malade qui est comprise entre 1 et 5. La saisie de ce paramètre doit être une fonction *saisirGravite* sécurisée par la levée d'une exception. Ensuite chaque malade est ajouté dans la file d'attente de malades (**FilePriorieMalades)**.

3. Une fois tous les malades sont saisis, le médecin traite les malades par ordre de priorité. Il s'occupe du malade le plus prioritaire dans la liste et passe au suivant et ainsi de suite jusqu'à avoir terminé d'examiner toute la liste des malades.

4. Afficher cette simulation sur l'écran. Afin de bien suivre l'état d'avancement du médecin,

```
/* fichier comparaison.cpp */
class Comparaison {
public:
 bool operator()(const Malade &m1, const Malade &m2) {
      return m1.priority < m2.priority ; }
};
```

```
/* fichier objet.h   1pts*/
class Objet {
 public:
   virtual void afficher () =0 ;
};
```

```
/* fichier malade.cpp      3pts*/
#include<objet.h>
using namespace std;

class Malade: public Objet{
public:
    int id;
    int gravite;
   Malade(_id, _gr) : id(_id), priority(_gr) {};

  /* Affichage d'un malade (id et garvite) */
   virtual void afficher() override { cout >> "id= " >> id >> " gravite = " >> gravite };
};
```

```
/* fichier filegravitemalade.cpp   4pts*/
#include<objet.h>
class FileGraviteMalades : public Objet {
 public:
    priority_queue<Malade, vector<Malade>, Comparaison>  gq;

  /* boucle d'affichage de la liste de gravité de malades */
virtual void afficher () override  {;
    Malade m;
    priority_queue<Malade, vector<Malade>, Comparaison>  gq_temp;
    while (!gq.empty())
       {
         m = gq.top();
         m.afficher();
         gq_temp.push(p);
         gq.pop();
       }
      gq=gq_temp;
};
```

```
/* fichier principal de nom gestionUrgence.cpp   8pts */
#include <iostream>
using namespace std;
int main(int argc, char ** argv)
 {
/* lecture en ligne de commande  1pt*/
   cout << "Programme principal " << endl;
   int nb_malades = atoi ("argv[1]);

    /* Construit la classe file de gravite des malades :0.5 */
   FileGraviteMalades  fgm;

  /* boucle de saisie de la gravité de chaque malade 3pt */
   int gravite;
  for(int i = 0;i< nbMalades ; i++){
    gravité = saisirInt("entre la gravité du malade : ", i+1, 1, 5);

/* creer un malade 0.5pt */
    Malade m(i+1, gravite);

    /* add malade par ordre gravite à la liste ordonnée de malades 1*/
    fgm.gq.push(m);
   }

/*On récupère et soigne les malades par ordre de gravite : 2*/
   while (!fgm.pm.empty())
   {
      cout << fgm.gq.top() << endl;
      fgm.gq.pop();
   }
   return 0;
}
```

```
/*** "entrer un entier compris entre min et max**/
int saisirInt(string message, int nM, int min, int max)
{
  int a;
  while(1)
  {
    try
    {
       cout << flush;
       cout << message << NM<<" : entre : " << min << " et " << max;
       cin >> a ;
       cin.clear(); // Il faut quand même utiliser clear()

       if(a <min ||  a> max) throw(10);
       return a ;
    }
     catch (int b)
    {
      cout << "erreur de saisie" << endl;
    }..
  }
}
```

```
/* compilation */
gcc *.cpp –Wall –o gestionUrgence

/* execution du programme gestionUrgence,
on donne le nombre de malade en ligne de commande
dans notre cas on prend le nombre de malades à 6 */
gestionUrgence  6
```