Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed Bougara de Boumerdès
Dep. Informatique, Faculté des Sciences

Evaluation (2018/2019) **Module : POO** Durée : 01h10	**Filière :** LMD – L2 Info **Année :** 2ème année – S3	Gaceb

Problématique (15 points)

Une image numérique peut être représentée dans un tableau 2D (matrice) où la valeur de chacune de ses cellules représente une couleur ou un niveau de gris de pixel correspondant dans l'image. Un pixel en couleur est basé sur un mélange de trois composant primaires ***(R = Rouge, V=Vert, B= Bleu)***. Un pixel en niveau de gris doit avoir une seule valeur ***NG ∈ [0, 255]*** qui représente sa luminance. Afin de développer une application permettant de gérer des images d'une manière souple et organisée, il est nécessaire de faire appel à la POO à travers les démarches suivantes :

1) Créer les deux classes ***PIXEL_RVB*** (attributs : ***R, V et B*** entiers) et ***PIXEL_NG*** (Attribut : NG entier). Les deux classes doivent embarquer les méthodes : ***void set(int val)*** pour initialiser le/les composant(s) d'un pixel à val, ***void lire ()*** pour lire le/les composant(s) d'un pixel fournis par l'utilisateur, ***void afficher()*** pour afficher à l'écran le/les composant(s) d'un pixel. La construction est implicite pour les deux classes.

1.5pt =(0.75 pt chaque classe)

```
class PIXEL_RVB
{
  public :
  int R, V, B;
  void setpix (int val) {R=val; V=val; B=val; }
  void lire (){ cout<<"R V B ? "; cin >> R>>V>>B;}
  void afficher(){ cout<<"("<<R<<","<<V<<","<<B<<")";}

 };   (0.75pt)

class PIXEL_NG
 {
   public : int NG;
  void setpix(int val) {NG=val; }
  void lire (){ cout<<"NG ? "; cin >> NG;}
  void afficher(){ cout<<NG; }

};  (0.75pt)
```

2) Ajouter la classe ***IMAGE*** qui encapsule un tableau *2D (appelé* ***DATA : privé)*** avec ses dimensions publics ***H*** *(hauteur),* **W** *(largeur) (0.5 pt),* un constructeur permettant de créer dynamiquement ce tableau avec les arguments suivants : H, W et VAL (VAL par défaut =0) (1pt).

1 .5pt

```
class IMAGE  {
  T **DATA;
  public:
  int L, C;
  IMAGE(const int row, const int col, int val=0)
  {
     L = row;
     C = col;
     DATA = new T *[L];

     for (int i = 0; i < L; i++)
     {
              DATA[i] = new T[C];
              for (int j = 0; j < C; j++)
               DATA[i][j].setpix(val);
     }
  }

};
```

3) ***DATA*** doit être de type générique (***PIXEL_RVB*** , ***PIXEL_NG***) pour pouvoir créer des images en couleurs et des images en ***NG***. Apporter les modifications nécessaires à la classe pour qu'elle devient générique.

1pt

```
template <class T>
 class IMAGE
 {
   T **DATA;
   public:

   int L, C;

   IMAGE(const int row =0, const int col=0, int val=0)
   {
     L = row;
     C = col;
     DATA = new T *[L];

     for (int i = 0; i < L; i++)
     {
              DATA[i] = new T[C];
              for (int j = 0; j < C; j++)
               DATA[i][j].setpix(val);
     }
   }

};.
```

4) Ajouter à la classe ***IMAGE*** la méthode ***void lire ()*** permettant de saisir les éléments de ***DATA*** par l'utilisateur et ***void afficher()*** pour afficher les valeurs de ***DATA*** à l'écran. Les deux méthodes reposes sur leurs équivalentes dans les classes **PIXEL_RVB** et ***PIXEL_NG._***

1pt

```
template <class T>  class IMAGE  {    T **DATA;    public:      . . .
   void lire()   {
      cout<<endl<<"Saisir les élements de l'image: ";
      for(int i = 0; i < L; i++)
      for(int j = 0; j < C; j++)
                {
                 cout<< endl<<"Pixel("<<i<<","<<j<<")=";
                 DATA[i][j].lire();
                }
   }

   void afficher()    {
      cout <<"Image :\n";
      for(int i = 0; i < L; i++)
      {
                for(int j = 0; j < C; j++)
                {
                  DATA[i][j].afficher () ; cout<<"   ";
                }
                cout <<endl;
      }
   }

}
};
```

5) Faut-il ajouter un destructeur? justifier votre réponse.

0.5pt

réponse oui et justif et 0.25pt pour le code

Oui, car la naissance de tout objet de type IMAGE fait l'objet d'une allocation dynamique de la mémoire pour l'attribut DATA et pour assurer une destruction correcte de cet objet et une libération complète de la mémoire il faut absolument ajouter un destructeur qui libère l'espace réservé pour DATA 0.25pt

```
template <class T> class IMAGE  {   …..  public:  ….
   ~IMAGE()   {
     for (int i = 0; i < L; i++)
      delete []  DATA[i];
      delete [] DATA;
   }

};
```

6) Ajouter à la classe ***IMAGE*** une surcharge de l'opérateur () permettant d'accéder directement aux pixels de ***DATA*** *(ex. img.DATA[i][j] sera accessible par img(i,j)).*

0.5pt

template <class T> class IMAGE {…   public: ….

T operator() (const int y, const int x)   {   return DATA[y][x];   }

};

7) Ajouter à la classe ***IMAGE*** une surcharge de l'opérateur – permettant de calculer directement la différence (gradient) entre deux images *(ex. img3=img2-img1).* Les différences entre les pixels doivent être calculée en valeur absolue.

1pt

template <class T> class IMAGE { T **DATA; public: ….

```
  IMAGE  operator- (const IMAGE & m)
 {
    if(L!=m.L || C!=m.C) // optionnelle n'est pas exigé dans l'énoncé
     return  IMAGE(0,0);

    IMAGE r(L, C);
    for(int i = 0; i < L; i+
      for(int j = 0; j < C; j++)
            {
             r.DATA[i][j] = DATA[i][j] - m.DATA[i][j];
            }
    return r;
 }
};
```

8) Faut-il ajouter une surcharge de ***l'opérateur –*** aux classes ***PIXEL_RVB*** , ***PIXEL_NG*** pour que Celle de la question (7) fonctionne sans incident.

075pt=Oui 0.25pt + code = 0.25pt pour chaque classe

class PIXEL_RVB { public : ….

```
 PIXEL_RVB  operator- (const PIXEL_RVB & p)
 {
   PIXEL_RVB s;

  s.R=abs(R-p.R);
  s.V=abs(V-p.V);
  s.B=abs(B-p.B);

 return s;
 }

 };
```

class PIXEL_NG { public : ….

```
 PIXEL_NG  operator- (const PIXEL_NG & p)
 {
  PIXEL_NG s;
  s.NG= abs(NG-p.NG);
  return s;
 }

};
```

9) Faut-il ajouter une surcharge de l'opérateur de copie et de constructeur de copie à la classe ***IMAGE*** pour que celle de la question (6) et (7) fonctionne sans incident ? Justifier votre réponse, si oui donner le code nécessaire.

1.25pt = (oui +justif)0.25+0.5 operateur copie+ 0.5 pt constructeur de recopie

Oui, car allocation dynamique de mémoire (0.25 pt), il faut ajouter les deux pour assurer une copie intégrale des données de DATA ou éviter leur perte (0.25 pt).

```
template <class T>  class IMAGE  {    T **DATA;    public: …..

 // constructeur par recopie (0.5pt)
  IMAGE(const IMAGE & m)    {       L = m.L;       C = m.C;
    DATA = new T *[L];

     for (int i = 0; i < L; i++)  {
                DATA[i] = new T[C];
                for (int j = 0; j < C; j++)
                DATA[i][j] = m.DATA[i][j];
     }
  }




// opérateur de copie (0.5pt)
   IMAGE & operator= (const IMAGE & m)    {
   // désaloué
   for (int i = 0; i < L; i++)    delete []  DATA[i];     delete [] DATA;

      L= m.L;   C= m.C;

//Allouer
      DATA = new T *[L];
     for (int i = 0; i < L; i++)       {       DATA[i] = new T [C];       }


       for(int i = 0; i < L; i++)
        for(int j = 0; j < C; j++)    {     DATA[i][j] =  m.DATA[i][j];     }

      return *this;
   }

};
```

10) Faut-il ajouter une surcharge de l'opérateur de copie et de constructeur par recopie aux classes ***PIXEL_RVB*** , ***PIXEL_NG*** pour que celle de la question (8) fonctionne sans incident ? justifier votre réponse, si oui donner le code nécessaire.

Non pas besoin (0.25pt), pas d'allocation dynamique de mémoire pour des objets des types pixels, la copie ou la construction par copie est faite systématiquement et correctement : attribut vers attribut (0.75pt).

1pt

11) Ajouter une fonction de surcharge ***d'opérateur*** **+** de deux images (arguments d'entrée de cette fonction transmis par référence) qui donne une image somme *(ex. img3=img2+img1).* Cette fonction doit être externe à la classe ***IMAGE*** mais amie a elle pour pouvoir accéder au pixels de tableau ***DATA*** (privé). Ecrire la définition de cette fonction (en dehors de la classe ***IMAGE***) et modifier le code des classes ***IMAGE, PIXEL_RVB*** , ***PIXEL_NG*** *(si nécessaire)* pour le bon fonctionnement de cette surcharge.

1.5pt

```
template <class T>  class IMAGE  {    T **DATA; public ….

   template <typename s>  friend IMAGE <s>  operator+ (const IMAGE <s> & m1, const IMAGE <s> & m2);
(0.25pt)

};


template <typename s> IMAGE <s>   operator+ (const IMAGE <s> & m1, const IMAGE <s> & m2)    {
        IMAGE <s> r(m1.L, m2.C);
       for(int i = 0; i < m1.L; i++)
        for(int j = 0; j < m1.C; j++)    {
                  r.DATA[i][j] =m1.DATA[i][j] + m2.DATA[i][j];
                }
      return r;
   }   (0.75pt)
```

Il faut aussi ajouter les deux fonctions suivantes aux classes (0.5 pt= 0.25 chacune)

```
class PIXEL_RVB {   public : …
  PIXEL_RVB  operator+ (const PIXEL_RVB & p)   {       PIXEL_RVB s;
   s.R=R+p.R;       s.V=V+p.V;       s.B=B+p.B;     return s;   }
 };

class PIXEL_NG … {  public : ….
  PIXEL_NG  operator+ (const PIXEL_NG & p)   {     PIXEL_NG s;     s.NG=NG+p.NG;    return s;  }
};
```

12) Donner un exemple d'utilisation de la classe ***IMAGE*** et ses méthodes dans la fonction ***main()*.**

1.5pt

(il faut que l'étudiant teste ici toutes les méthodes et mécanismes offerts par la classe IMAGE)

```
int main(){

   IMAGE <PIXEL_NG>  a(2,2),  b(2,2), k(2,2), m(2,2);  // tester template et création
   PIXEL_NG p;
   a.lire(); // tester la méthode afficher
   b.lire();
// tester l'opérateur + et l'amitié , opérateur de copie et construction de copie
   k=a+b;  // tester l'opérateur +
   m=a-b; // tester l'opérateur -
   p=a(1,1);  // tester l'opérateu ()
   k.afficher(); // tester l'affichage
   k=a;     k.afficher();  // tester l'opérateur de copie
   IMAGE <PIXEL_NG> f=a; f.afficher(); // tester le constructeur de recopie
```

return 0 ; }

13) Je souhaite créer un tableau ***1D*** de pixels hétérogènes (contenant des pixels ***RVB*** et des pixels en ***NG*** dans le même tableau) à l'aide de la bibliothèque ***STL <vector>.*** Quel mécanisme de POO faut-il utiliser ? apporter des modifications simples à votre programme principale (avec des explications) pour pouvoir réaliser cette objectif.

1pt

Quel mécanisme de POO faut-il utiliser ? c'est le polymorphisme (0.25 pt)
On ajoute le code en couleur verte (0.5pt) :

```
class PIXEL  // Classe abstraite pour créer le polymorphisme de la question 13
{
 public:
 virtual void setpix (int val)=0;
 virtual void lire ()=0;
 virtual void afficher()=0;
};
```

0.25pt (héritage)

```
class PIXEL_RVB : public PIXEL {….};
class PIXEL_NG :  public PIXEL {….};
```

14) Donner un exemple ***main()*** de création du tableau ***T*** de 5 pixels suivants (en se basant sur le travail fait pour la question (13) ***T= [NG| RVB|RVB|NG|RVB]*** **(0.5 pt)** et de saisie des 5 pixels à l'aide d'une boucle for de 5 itérations **(0.5pt).**

1pt

```
int main(){
    // Création de taleau T
   vector <PIXEL *> T;
   T.push_back(new PIXEL_NG());
   T.push_back(new PIXEL_RVB());
   T.push_back(new PIXEL_RVB());
   T.push_back(new PIXEL_NG());
   T.push_back(new PIXEL_RVB());

   // Saisie des valeurs des pixels dans T
   for(int i=0; i<5; i++)
    T[i]->lire();

   // Affichage
   for(int i=0; i<5; i++)
    T[i]->afficher();

    return 0;
}
```

Bon courage

Corrigé, code complet :

```
// Corrigé Evaluation POO 2018-2019_GACEB

#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

// Q13 : Classe abstraite pour créer le polymorphisme de la question 13

class PIXEL
{
 public:
 virtual void setpix (int val)=0;
 virtual void lire ()=0;
 virtual void afficher()=0;

};

class PIXEL_RVB : public PIXEL
{
  public :
  int R, V, B;
  void setpix (int val) {R=val; V=val; B=val; }
  void lire (){ cout<<"R V B ? "; cin >> R>>V>>B;}
  void afficher(){ cout<<"("<<R<<","<<V<<","<<B<<")";}

  PIXEL_RVB  operator+ (const PIXEL_RVB & p)
  {
     PIXEL_RVB s;

    s.R=R+p.R;
    s.V=V+p.V;
    s.B=B+p.B;

   return s;
  }


  PIXEL_RVB  operator- (const PIXEL_RVB & p)
  {
     PIXEL_RVB s;

    s.R=abs(R-p.R);
    s.V=abs(V-p.V);
    s.B=abs(B-p.B);

   return s;
  }

 };

class PIXEL_NG : public PIXEL
 {

   public : int NG;

  void setpix(int val) {NG=val; }
  void lire (){ cout<<"NG ? "; cin >> NG;}
```

```
  void afficher(){ cout<<NG; }

  PIXEL_NG  operator+ (const PIXEL_NG & p)
  {
    PIXEL_NG s;
    s.NG=NG+p.NG;
    return s;
  }

  PIXEL_NG  operator- (const PIXEL_NG & p)
  {
    PIXEL_NG s;
    s.NG= abs(NG-p.NG);
    return s;
  }

};

 template <class T>
 class IMAGE
 {
   T **DATA;
   public:

   int L, C;

   IMAGE(const int row, const int col, int val=0)
   {
      L = row;
      C = col;
      DATA = new T *[L];

      for (int i = 0; i < L; i++)
      {
          DATA[i] = new T[C];
          for (int j = 0; j < C; j++)
           DATA[i][j].setpix(val);
      }
   }
  // constructeur par recopie

   IMAGE(const IMAGE & m)
   {
      L = m.L;
      C = m.C;
      DATA = new T *[L];

      for (int i = 0; i < L; i++)
      {
          DATA[i] = new T[C];
          for (int j = 0; j < C; j++)
          DATA[i][j] = m.DATA[i][j];
      }
   }
// destructeur
  ~IMAGE()
  {
    for (int i = 0; i < L; i++)
      delete []  DATA[i];

      delete [] DATA;
```

```
}

void lire()
{
   cout<<endl<<"Saisir les élements de l'image: ";
   for(int i = 0; i < L; i++)
   for(int j = 0; j < C; j++)
      {
       cout<< endl<<"Pixel("<<i<<","<<j<<")=";
       DATA[i][j].lire();
      }

}

void afficher()
{
   cout <<"Image :\n";
   for(int i = 0; i < L; i++)
   {
       for(int j = 0; j < C; j++)
       {
         DATA[i][j].afficher () ; cout<<"   ";
       }
       cout <<endl;
   }
}

IMAGE  operator- (const IMAGE & m)
{
    if(L!=m.L || C!=m.C) // optionnelle
     return  IMAGE(0,0);

    IMAGE r(L, C);
    for(int i = 0; i < L; i++)
      for(int j = 0; j < C; j++)
        {
          r.DATA[i][j] = DATA[i][j] - m.DATA[i][j];
        }

    return r;
 }



T operator() (const int y, const int x)
{
   return DATA[y][x];
}


 IMAGE & operator= (const IMAGE & m)
{


 // désaloué
 for (int i = 0; i < L; i++)
   delete []  DATA[i];

   delete [] DATA;
```

```
        L= m.L;
        C= m.C;

        DATA = new T *[L];

       for (int i = 0; i < L; i++)
       {
           DATA[i] = new T [C];

       }


         for(int i = 0; i < L; i++)
          for(int j = 0; j < C; j++)
            {
              DATA[i][j] =  m.DATA[i][j];
            }

       return *this;
   }

   template <typename s>  friend IMAGE <s>  operator+ (const IMAGE <s> &
m1, const IMAGE <s> & m2);

};

//friend
template <typename s>
IMAGE <s>   operator+ (const IMAGE <s> & m1, const IMAGE <s> & m2)
   {
       // if(m2.L!=m1.L || m2.C!=m1.C) // optionnelle
       //  return  IMAGE(0,0);
        IMAGE <s> r(m1.L, m2.C);
        for(int i = 0; i < m1.L; i++)
          for(int j = 0; j < m1.C; j++)
            {
              r.DATA[i][j] =m1.DATA[i][j] + m2.DATA[i][j];
            }

        return r;
    }
```

```
// exemple  main
int main()
{
    //Question 13 exemple
   IMAGE <PIXEL_NG> a(2,2),  b(2,2), k(2,2), m(2,2);
   PIXEL_NG p;
   a.lire();
   b.lire();

   k=a+b;
   m=a-b;

   p=a(1,1); // equivalent à p= a.DATA[1][1]

   k.afficher();

   //Question 14
   // Création de taleau T
   vector <PIXEL *> T;
   T.push_back(new PIXEL_NG());
   T.push_back(new PIXEL_RVB());
   T.push_back(new PIXEL_RVB());
   T.push_back(new PIXEL_NG());
   T.push_back(new PIXEL_RVB());

   // Saisie des valeurs des pixels dans T
   for(int i=0; i<5; i++)
    T[i]->lire();

   // Affichage à l'écran de contenu de T
   for(int i=0; i<5; i++)
    T[i]->afficher();

    return 0;
}
```