# Algorithmique et Base de la programmation

Durée : *1h30*
Documents autorisés : **AUCUN**

**Remarques** :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier sur votre copie.

## 1 Traitements d'image (8 points)

Une image bitmap en 256 niveaux de gris peut être utilisée à l'aide du type `ImageNB` et des fonctions et procédures suivantes :
- **fonction** imageNB **(**largeur,hauteur : **NaturelNonNul**, couleurFond : [0..255]**) :** ImageNB
- **fonction** largeur **(**im : ImageNB**) : NaturelNonNul**
- **fonction** hauteur **(**im : ImageNB**) : NaturelNonNul**
- **fonction** obtenirCouleur **(**im : ImageNB,x,y : **Naturel) :** [0..255]

  ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im)
- **procédure** fixerCouleur **(E/S** im : ImageNB,**E** x,y : **Naturel**, couleur : [0..255]**)**

  ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im)

En traitement d'image, un filtre est une fonction qui calcule une image destination à partir d'une image source. Chaque pixel de l'image destination est fonction des couleurs des pixels de l'image source et optionnellement des couleurs des pixels de l'image destination déjà calculées.

### Filtre moyenneur

Le principe du filtre moyenneur est que la couleur de chaque pixel $x, y$ de l'image destination est égale à la moyenne des couleurs des pixels appartenant à une fenêtre carrée centrée en $x, y$ de l'image source. La largeur de cette fenêtre carrée est obligatoirement impaire.

La figure 1 présente l'utilisation d'un filtre moyenneur avec une fenêtre de taille 5 (exemple issue de `http://perso.telecom-paristech.fr/~maitre/BETI//filtres_lin_nlin/filtres.html`).

(a) Image source (b) Image destination

FIGURE 1 – Utilisation d'un filtre moyenneur de taille 5

### Questions

1. Analyse : Proposez une analyse descendante du problème.
2. Conception préliminaire : Donnez les signatures des fonctions et procédures issues de votre analyse.
3. Conception détaillée : Donnez les algorithmes de toutes les fonctions et procédures de votre conception préliminaire.

**Solution proposée :**

1. Analyse :
   - filtreMoyenneur : ImageNB × **NaturelNonNul** ⇸ ImageNB
     - moyenneFenetree : ImageNB × **Naturel** × **Naturel** × **NaturelNonNul** ⇸ [0..255]
       - min : **Entier** × **Entier** → **Entier**
       - max : **Entier** × **Entier** → **Entier**
2. CP :

   **fonction** filtreMoyenneur **(**im : ImageNB, tailleFenetre : **NaturelNonNul) :** ImageNB

   ⌊**précondition(s)** impair(tailleFenetre)

   **fonction** moyenneFenetree **(**im : imageNB, x,y : **Naturel**, tailleFenetre : **NaturelNonNul) :** [0..255]

   ⌊**précondition(s)** x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)
3. CD :

```
fonction filtreMoyenneur (im : ImageNB, tailleFenetre : NaturelNonNul) : ImageNB
    ⌊précondition(s)  impair(tailleFenetre)
    Déclaration  i,j : Naturel
                 res : ImageNB
debut
    im ← imageNB(largeur(im),hauteur(im),0)
    pour i ←0 à largeur(im)-1 faire
        pour j ←0 à hauteur(im)-1 faire
            fixerCouleur(res,i,j,moyenneFenetree(im,x,y,tailleFenetre))
        finpour
    finpour
    retourner res
fin
fonction moyenneFenetree (im : imageNB, x,y : Naturel, tailleFenetre : NaturelNonNul) : [0..255]
    ⌊précondition(s)  x≤0 et x<largeur(im) et y≤0 et y<hauteur(im) et impair(tailleFenetre)
    Déclaration  i,j,somme,nbPixel : Naturel
debut
    somme ← 0
    nb ← 0
    pour i ←entierEnNaturel(max(0,x-taillefrenetre div 2)) à entierEnNaturel(min(largeur(im)-1,x+taillefrenetre
    div 2)) faire
        pour j ←entierEnNaturel(max(0,y-taillefrenetre div 2)) à entierEnNaturel(min(hauteur(im)-1,y+taillefrenetre
        div 2)) faire
            somme ← somme+obtenirCouleur(im,i,j)
            nb ← nb+1
        finpour
    finpour
    retourner somme div nb
fin
```

# 2 Un peu de récursivité (6 points)

### Rappels mathématiques

Les trois sommets d'un triangle équilatéral, dont l'un des côté est parallèle à l'axe des abscisses, de centre $x_c, y_c$ de base $b$ ont les coordonnées suivantes :

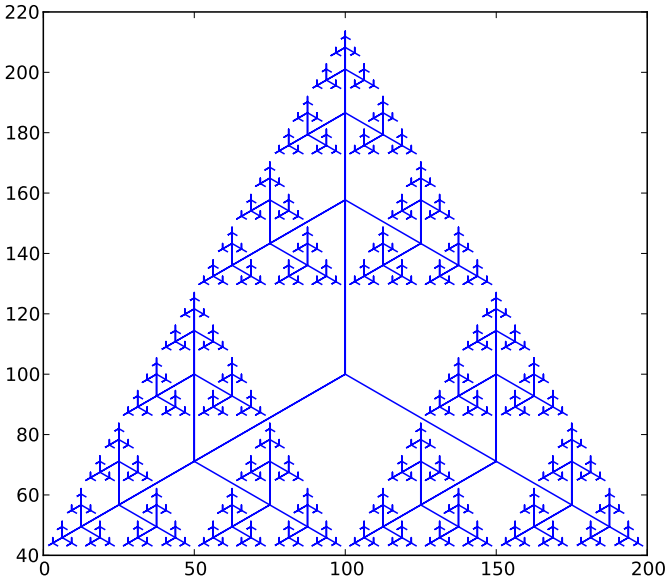


FIGURE 2 – Dessin récursif

- $x_c, y_c + 2 * h/3$
- $x_c - b/2, y_c - h/3$
- $x_c + b/2, y_c - h/3$

avec $h = \sqrt{b^2 - (b/2)^2}$

## Questions

Supposons que la procédure suivante permette de dessiner un segment sur un graphique (variable de type `Graphique`) :

- **procédure** ligne **(E/S** g : Graphique,**E** x1,y1,x2,y2 : **Reel)**

L'objectif est de concevoir une procédure `dessinerCroix` qui permet de dessiner sur un graphique des dessins récursifs tels que présentés par la figure 2. La signature de cette procédure est :

- **procédure** dessinerCroix **(E/S** g : Graphique,**E** xCentre,yCentre,base : **Reel**, niveauRecursion : **Naturel)**

1. Lors du premier appel de cette procédure, donnez la valeur des quatre derniers paramètres effectifs afin d'obtenir le graphique de la figure 2.
2. Donnez le corps de cette procédure.

**Solution proposée :**

1. dessinerCroix(g,100,100,200,5)
2. Algo

   **procédure** dessinerCroix **(E/S** g : Graphique,**E** xCentre,yCentre,base : **Reel**, niveauRecursion : **Naturel)**

   **Déclaration** x1,y1,x2,y2,x3,y3,h : **Reel**

   **debut**

   h ← racineCarree(b*b-(b/2)*(b/2))
   x1 ← xc
   y1 ← yc+2*h/3
   x2 ← xc-b/2
   y2 ← yc-h/3
   x3 ← xc+b/2
   y3 ← yc-h/3
   ligne(g,xc,yc,x1,y1)

```
        ligne(g,xc,yc,x2,y2)
        ligne(g,xc,yc,x3,y3)
        si n>0 alors
            dessinerCroix(g,x1,y1,b/2,n-1)
            dessinerCroix(g,x2,y2,b/2,n-1)
            dessinerCroix(g,x3,y3,b/2,n-1)
        finsi
    fin
```

# 3 Questions de C (4 points)

En Travaux Dirigés, nous avons vu l'algorithme suivant :

**procédure** calculerMinMax **(E** t : **Tableau**[1..MAX] **d'Entier** ; n : **Naturel**, **S** min,max : **Entier)**

⌊**précondition(s)** n≤MAX

**Déclaration** i : **Naturel**

**debut**
  min ← t[1]
  max ← t[1]
  **pour** i ←2 **à** n **faire**
    **si** t[i]<min **alors**
      min ← t[i]
    **finsi**
    **si** t[i]>max **alors**
      max ← t[i]
    **finsi**
  **finpour**
**fin**

Proposez la fonction C correspondante.

**Solution proposée :**

```
void calculerMinMax(int t[], int n, int* pmin, int* pmax){
  assert(n<MAX);

  int i;
  *pmin=t[0];
  *pmax=t[0];
  for(i=1;i<n;i++) {
    if (t[i]<*pmin)
      *pmin=t[i];
    if (t[i]>*pmax)
      *pmax=t[i];
  }
}
```