LIMACE version 1.6
Manuel utilisateur

ALAIN CROUZIL
alain.crouzil@irit.fr

1 Introduction

L'objectif de LIMACE (List, Image and Matrix C Extensions) est de fournir des outils de base afin de faciliter la manipulation de listes génériques, d'images (couleur, niveaux de gris ou noir et banc) et de matrices (d'éléments de type `int` ou `double`) en langage C ANSI.

La bibliothèque LIMACE se présente sous la forme du fichier d'en-tête `limace.h` et du fichier source `limace.c`. Les outils disponibles sont constitués de :

— trois type « privés » : `Image`, `Matrix` et `List` ;
— deux énumérations : `ImageType` et `MatrixType` permettant respectivement de spécifier le type d'images à manipuler et le type des éléments des matrices ;
— un ensemble de fonctions permettant de manipuler les images, les matrices et les listes.

La structure de programmation de LIMACE permet l'encapsulation des données, c'est-à-dire qu'elle ne permet pas l'accès à l'implémentation des objets manipulés ; leur accès n'est possible que par l'intermédiaire des fonctions disponibles dont les prototypes sont fournis dans le fichier `limace.h`. En outre, chaque prototype est accompagné de commentaires facilitant l'utilisation de la fonction correspondante.

LIMACE n'a pas l'ambition d'être une bibliothèque complète mais propose un ensemble minimal d'outils permettant de s'affranchir du développement de parties de code souvent difficiles à mettre au point. En outre, son implémentation actuelle se veut être un compromis entre rapidité d'exécution, occupation mémoire, facilité d'utilisation, lisibilité du code et sécurité d'utilisation, avec une pondération importante sur ce dernier point.

1.1 Avertissement

Les différentes versions de LIMACE ont été testées avec différents compilateurs C sous différents systèmes d'exploitation.

L'auteur ne pourra en aucun cas être tenu responsable d'éventuels dégâts causés directement ou indirectement par l'utilisation de LIMACE .

Toute utilisation non commerciale de LIMACE est autorisée. Avant toute utilisation commerciale, veuillez contacter au préalable son auteur (`alain.crouzil@irit.fr`). LIMACE peut être copiée et distribuée dans la mesure où les fichiers ne sont pas modifiés et que la présente documentation accompagne cette copie.

Les remarques et suggestions sont les bienvenues et peuvent être envoyées à `alain.crouzil@irit.fr`

1.2 Utilisation de LIMACE

Dans votre programme, il suffit de rajouter la ligne suivante :

```
#include "limace.h"
```

En ce qui concerne la compilation, il faut effectuer l'édition de liens entre le fichier objet de LIMACE et votre programme en n'oubliant pas de faire appel à la bibliothèque mathématique (`-lm` en fin de commande).

1.3 Gestion des erreurs

Dans la mesure du possible, les erreurs ont été gérées de la manière suivante : si une erreur est détectée par une fonction, un message décrivant le problème est affiché sur `stderr` et la fonction retourne une valeur permettant à l'appelant de prendre connaissance de la présence de cette erreur et donc de prendre les décisions adéquates. En aucun cas, les fonctions de LIMACE n'interrompent l'exécution du programme, à l'exception de la présence involontaire et regrettable d'un *bug*.

Si l'on ne souhaite pas que les messages d'erreurs soient affichés on peut utiliser la fonction d'en-tête :

```
int Verbose(void);
```

qui fonctionne comme un commutateur d'état. Par défaut, l'état a la valeur `ON` (les messages sont affichés). Pour ne plus avoir d'affichage de messages d'erreur, il suffit de faire appel à l'instruction `Verbose();` qui positionne l'état à la valeur `OFF` (aucun affichage). À chaque appel de la fonction on passe d'un état à l'autre. `Verbose` retourne le nouvel état (`ON` ou `OFF`).

2 Manipulation d'images

2.1 Déclaration d'une image

L'instruction :

`Image` *Identificateur*`;`

déclare une image de nom *Identificateur*, mais n'effectue aucune réservation d'espace mémoire pour cette image.

2.2 Création d'une image

La fonction d'en-tête :

```
Image ImAlloc(ImageType Type, int NbRow, int NbCol);
```

effectue la réservation de l'espace mémoire pour une image comportant `NbRow` lignes et `NbCol` colonnes. Le type de l'image doit être spécifié ; trois types sont possibles :

- `BitMap` : image noir et blanc ne comportant que des 0 et des 1 ; 0 → noir, 1 → blanc.
- `GrayLevel` : image de niveaux de gris compris entre 0 et 255.
- `Color` : image couleur à trois canaux RGB, chacun ayant des valeurs comprises entre 0 et 255.

`ImAlloc` retourne une valeur de type `Image` si tout s'est bien passé ou `NULL` sinon.

La fonction `ImCAlloc` fonctionne comme la fonction `ImAlloc` mais, en plus, tous les pixels de l'image sont initialisés à 0.

2.3 Type d'une image

La fonction d'en-tête :

```
ImageType ImType(Image Im);
```

retourne le type de l'image `Im`. Trois valeurs peuvent être retrounées : `BitMap`, `GrayLevel` ou `Color` (voir le paragraphe 2.2).

2.4 Dimensions d'une image

La fonction d'en-tête :

```
int ImNbRow(Image Im);
```

retourne le nombre de lignes de l'image `Im`. Si un problème est détecté, elle retourne 0.

La fonction d'en-tête :

```
int ImNbCol(Image Im);
```

retourne le nombre de colonnes de l'image `Im`. Si un problème est détecté, elle retourne 0.

2.5 Accès aux pixels d'une image

Il n'est pas possible d'accéder directement aux pixels d'une image. Il faut tout d'abord utiliser une des fonctions qui retournent un accès soit à la matrice des niveaux de gris (image de type `BitMap` ou `GrayLevel`), soit aux matrices des plans R, G ou B (image de type `Color`).

2.5.1 Images noir et blanc ou de niveaux de gris

La fonction d'en-tête :

```
unsigned char **ImGetI(Image Im);
```

retourne un accès à la matrice des niveaux de gris d'une image de type `BitMap` ou `GrayLevel`. Si un problème est détecté, la valeur `NULL` est retournée. Cette fonction retourne une adresse permettant l'accès aux niveaux de gris grâce à l'opérateur `[][]`.
Exemple :

```
int i,j;
Image Im; /* declare l'image Im */
unsigned char **I;, /* declare un acces à une matrice de unsigned char */

Im=ImAlloc(GrayLevel,100,200); /* cree une image de niveaux de gris comportant */
                               /* 100 lignes et 200 colonnes                   */
I=ImGetI(Im); /* recupere l'acces a la matrice des niveaux de gris */
for (i=0;i<100;i++)
  for (j=0;j<200;j++)
    I[i][j]=128; /* affecte le niveau de gris 128 a chaque pixel de l'image */
```

Attention : Une erreur consiste à utiliser `Im[i][j]` au lieu de `I[i][j]`.
Parcours rapide d'une image : si l'on désire parcourir rapidement une image en n'utilisant qu'un seul pointeur, on peut utiliser la technique suivante :

```
Image Im;
unsigned char *Debut,*Fin,*p;

Im=ImAlloc(GrayLevel,100,200);
Debut=*(ImGetI(Im)); /* recuperation de l'adresse de début de la zone */
                     /* memoire contenant les niveaux de gris         */
Fin=Debut+100*200; /* calcul de l'adresse de fin de la zone */
for (p=Debut;p<Fin;*p++=128); /* affecte le niveau de gris 128 a chaque */
                              /* pixel de l'image                       */
```

Attention : ce type de parcours doit être utilisé avec prudence à cause du risque de commettre une erreur et de son manque de lisibilité.

2.5.2 Images couleur

L'accès aux pixels d'une image couleur fonctionne de manière similaire à celui concernant les images noir et blanc ou de niveaux de gris. On accède séparément aux trois plans R, G et B correspondant aux composantes rouge, verte et bleue des images.

La fonction d'en-tête :

```
unsigned char **ImGetR(Image Im);
```

retourne un accès à la matrice du plan R. Si un problème est détecté, la valeur `NULL` est retournée.

La fonction d'en-tête :

```
unsigned char **ImGetG(Image Im);
```

retourne un accès à la matrice du plan G. Si un problème est détecté, la valeur `NULL` est retournée.

La fonction d'en-tête :

```
unsigned char **ImGetB(Image Im);
```

retourne un accès à la matrice du plan B. Si un problème est détecté, la valeur `NULL` est retournée.
Exemple :

```
int i,j;
Image Im; /* declaration de l'image */
unsigned char **R,**G,**B; /* declaration des acces aux matrices des 3 plans */

Im=ImAlloc(Color,100,200); /* creation d'une image couleur de 100 lignes et */
                           /* 200 colonnes                                    */
R=ImGetR(Im); /* recuperation des   */
G=ImGetG(Im); /* acces aux matrices */
B=ImGetB(Im); /* des 3 plans        */
for (i=0;i<100;i++)
  for (j=0;j<200;j++)
  {
    R[i][j]=255; /* parcours simultane des 3 plans : */
    G[i][j]=128; /* tous les pixels recoivent la     */
    B[i][j]=192; /* couleur rose                     */
  }
```

Remarque : Il est possible d'utiliser (avec prudence) la technique du parcours rapide sur chaque plan (voir page 3).

2.6 Copie d'une image

La fonction d'en-tête :

```
Image ImCopy(Image Im);
```

retourne une copie de l'image `Im` ou `NULL` si un problème est détecté. L'allocation mémoire est effectuée par `ImCopy` mais sa libération est à la charge de l'appelant (voir le paragraphe 2.8).

2.7 Entrées-sorties

Les entrées-sorties d'images sont effectuées avec les formats [1] PBM, PGM, PPM (souvent regroupés sous l'appellation PNM : Portable aNyMap) :

— PBM (Portable BitMap) pour les images noir et blanc (type `BitMap`);
— PGM (Portable GrayMap) pour les images de niveaux de gris (type `GrayLevel`);
— PPM (Portable PixelMap) pour les images couleur (type `Color`).

Pour chacun de ces formats, il existe une version binaire et une version texte (ASCII) qui entraîne des fichiers beaucoup plus gros mais qui présente l'avantage de pouvoir vérifier ou modifier facilement les valeurs des pixels. Mais dans les deux cas, l'en-tête est en mode texte.

La seule restriction concerne la taille de codage des pixels qui est limitée à :

— 8 bits pour les images de niveaux de gris (PGM), donc des valeurs pouvant aller de 0 à 255;
— 3×8 bits pour les images couleur (PPM), donc chaque plan a des valeurs pouvant aller de 0 à 255.

En théorie, ces formats prévoient la possibilité de codages plus longs qui sont rarement utilisés. C'est pourquoi, ils ne sont pas supportés par LIMAGE.

1. Ces formats sont connus de nombreux produits. Pour convertir des images, on pourra utiliser NetPbm (`ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM`) ou, pour PC, IrfanView (`http://www.irfanview.com`).

2.7.1 Lecture d'une image

La fonction d'en-tête :

```
Image ImRead(char FileName[]);
```

retourne une image lue dans le fichier de nom `FileName`. Si `FileName = ""` (la chaîne vide), l'image est lue sur `stdin`; cette possibilité peut être utilisée pour réaliser des filtres (voir le paragraphe 2.9). La réservation mémoire est réalisée par `ImRead`. La libération est à la charge de l'appelant. Si un problème est détecté, la valeur `NULL` est retournée et des messages sont affichés sur `stderr`.

2.7.2 Écriture d'une image

Deux fonctions sont disponibles pour écrire une image; elles correspondent aux deux versions, binaire et texte, des formats PBM, PGM et PPM.

La fonction d'en-tête :

```
void ImWrite(Image Im, char FileName[]);
```

écrit l'image `Im` dans le fichier de nom `FileName` au format binaire. Le choix entre les formats PBM, PGM et PPM est fait en fonction du type de `Im`. Si `FileName = ""` (la chaîne vide), l'image est écrite sur `stdout`.
Attention : sous DOS, `stdout` étant considéré comme un flux de texte, le résultat de l'écriture peut être incorrect. C'est pourquoi, il est fortement conseillé d'utiliser plutôt la fonction `ImWriteAsc` si l'on veut faire un filtre ou afficher les valeurs des pixels de l'image à l'écran.

La fonction d'en-tête :

```
void ImWriteAsc(Image Im, char FileName[]);
```

écrit l'image `Im` dans le fichier de nom `FileName` au format texte ASCII. Le choix entre les formats PBM, PGM et PPM est fait en fonction du type de `Im`. Si `FileName = ""` (la chaîne vide), l'image est écrite sur `stdout`.
Exemple :

```
Image Im,ImRes;
int NbLig,NbCol,i,j;
unsigned char **I,**IRes;

Im=ImRead("jolie.pgm"); /* lecture de l'image stockée dans le fichier jolie.pgm */
NbLig=ImNbRow(Im); /* recuperation des dimensions de l'image */
NbCol=ImNbCol(Im);
I=ImGetI(Im); /* recuperation d'un acces aux pixels de Im */
ImRes=ImAlloc(GrayLevel,NbLig,NbCol); /* creation de l'image resultat */
IRes=ImGetI(ImRes); /* recuperation d'un acces aux pixels de ImRes */
for (i=0;i<NbLig;i++)
  for (j=0;j<NbCol;j++)
    IRes[i][j]=255-I[i][j]; /* inversion des niveaux de gris de l'image */
ImWrite(ImRes,"eiloj.pgm"); /* ecriture de l'image resultat dans eiloj.pgm */
ImFree(&Im); ImFree(&ImRes); /* liberation memoire (voir ci -dessous) */
```

2.8 Libération de l'espace mémoire occupé par une image

La fonction d'en-tête :

```
void ImFree(Image *pIm);
```

libère l'espace mémoire occupé par l'image pointée par `pIm` et affecte la valeur `NULL` à `*pIm`.
Attention : contrairement à la fonction standard `free`, c'est l'adresse de l'image qui doit être passée et pas l'image (`ImFree(&Im);`).

2.9 Exemple de réalisation d'un filtre

```
#include "limace.h"

int main(void)
{
  Image Im1,Im2;
  unsigned char **I1,**I2;
  int i,j,NbLig,NbCol;

  Im1=ImRead(""); /* lecture de l'image sur stdin */
  if (Im1==NULL) return 1;
  if (ImType(Im1)!=GrayLevel) return 2;
  NbLig=ImNbRow(Im1); NbCol=ImNbCol(Im2);
  I1=ImGetI(Im1);
  Im2=ImAlloc(GrayLevel,NbLig,NbCol);
  if (Im2==NULL) return 3;
  I2=ImGetI(Im2);
  for (i=0;i<NbLig;i++)
    for (j=0;j<NbCol;j++)
      I2[i][j]=255-I1[i][j];
  ImWriteAsc(Im2,""); /* ecriture de l'image resultat sur stdout au format */
                      /* texte pour ne pas avoir de probleme sous DOS      */
  ImFree(&Im1); ImFree(&Im2); /* liberation memoire */
  return 0;
}
```

Si l'exécutable produit à partir de ce programme s'appelle `inversion`, sous Unix on pourra l'utiliser de la manière suivante :

```
cat jolie.pgm | inversion > eiloj.pgm
```

Cette technique permet de réaliser des enchaînements de commandes grâce à des tubes de communication (*pipe*).

3 Manipulation de matrices

3.1 Déclaration d'une matrice

L'instruction :

`Matrix` *Identificateur*`;`

déclare une matrice de nom *Identificateur*, mais n'effectue aucune réservation d'espace mémoire pour cette matrice.

3.2 Création d'une matrice

La fonction d'en-tête :

```
Matrix MatAlloc(MatrixType Type, int NbRow, int NbCol);
```

effectue la réservation de l'espace mémoire pour une matrice comportant `NbRow` lignes et `NbCol` colonnes. Le type de la matrice doit être spécifié ; deux types sont possibles :

- `Int` : matrice de `int`
- `Double` : matrice de `double`

`MatAlloc` retourne une valeur de type `Matrix` si tout s'est bien passé ou `NULL` sinon.

La fonction `MatCAlloc` fonctionne comme la fonction `MatAlloc` mais, en plus, tous les éléments de la matrice sont initialisés à 0.

3.3 Type d'une Matrice

La fonction d'en-tête :

```
MatrixType MatType(Matrix Mat);
```

retourne le type de la matrice `Mat`. Deux valeurs peuvent être retrounées : `Int` ou `Double` (voir le paragraphe 3.2).

3.4 Dimensions d'une matrice

La fonction d'en-tête :

```
int MatNbRow(Matrix Mat);
```

retourne le nombre de lignes de la matrice `Mat`. Si un problème est détecté, elle retourne 0.

La fonction d'en-tête :

```
int MatNbCol(Matrix Mat);
```

retourne le nombre de colonnes de la matrice `Mat`. Si un problème est détecté, elle retourne 0.

3.5 Accès aux éléments d'une matrice

Il n'est pas possible d'accéder directement aux éléments d'une matrice. Il faut tout d'abord utiliser une des deux fonctions qui retournent un accès aux éléments d'une matrice de `int` ou d'une matrice de `double`.

3.5.1 Matrices de `int`

La fonction d'en-tête :

```
int **MatGetInt(Matrix Mat);
```

retourne un accès aux éléments d'une matrice de type `Int`. Si un problème est détecté (type de la matrice incorrect, par exemple), la valeur `NULL` est retournée. Cette fonction retourne une adresse permettant l'accès aux éléments grâce à l'opérateur `[][]`.
Exemple :

```
int i,j;
Matrix Mat; /* declare la matrice Mat */
int **M;, /* declare un acces aux éléments d'une matrice de int */

Mat=MatAlloc(Int,10,20); /* cree une matrice de int comportant */
                         /* 10 lignes et 20 colonnes           */
M=MatGetInt(Mat); /* recupere l'acces aux elements de la matrice de int */
for (i=0;i<10;i++)
  for (j=0;j<20;j++)
    M[i][j]=12; /* affecte la valeur 12 a tous les elements de la matrice */
```

Attention : Une erreur consiste à utiliser `Mat[i][j]` au lieu de `M[i][j]`.
Parcours rapide d'une matrice de `int` *:* si l'on désire parcourir rapidement une matrice de `int` en n'utilisant qu'un seul pointeur, on peut utiliser la technique suivante :

```
Matrix Mat;
int *Debut,*Fin,*p;

Mat=MatAlloc(Int,10,20);
Debut=*(MatGetInt(Mat)); /* recuperation de l'adresse de début de la zone */
                         /* memoire contenant les elements de la matrice  */
Fin=Debut+10*20; /* calcul de l'adresse de fin de la zone */
for (p=Debut;p<Fin;*p+=12); /* affecte la valeur 12 a chaque */
                            /* element de la matrice        */
```

Attention : ce type de parcours doit être utilisé avec prudence à cause du risque de commettre une erreur et de son manque de lisibilité.

3.5.2 Matrices de `double`

L'accès aux éléments d'une matrice de `double` fonctionne exactement de la même manière que celui concernant les matrices de `int`. La seule différence réside dans la fonction d'accès (`MatGetDouble`) et le type retourné (`double **`).

La fonction d'en-tête :

```
double **MatGetDouble(Matrix Mat);
```

retourne un accès aux éléments d'une matrice de type `Double`. Si un problème est détecté (type de la matrice incorrect, par exemple), la valeur `NULL` est retournée. Cette fonction retourne une adresse permettant l'accès aux éléments grâce à l'opérateur `[][]`.
Exemple :

```
int i,j;
Matrix Mat; /* declare la matrice Mat */
double **M;, /* declare un acces aux éléments d'une matrice de double */

Mat=MatAlloc(Double,10,20); /* cree une matrice de double comportant */
                            /* 10 lignes et 20 colonnes             */
M=MatGetDouble(Mat); /* recupere l'acces aux elements de la matrice de double */
for (i=0;i<10;i++)
  for (j=0;j<20;j++)
    M[i][j]=3.14; /* affecte la valeur 3.14 a tous les elements de la matrice */
```

Attention : Une erreur consiste à utiliser `Mat[i][j]` au lieu de `M[i][j]`.
Parcours rapide d'une matrice `double` *:* si l'on désire parcourir rapidement une matrice de `double` en n'utilisant qu'un seul pointeur, on peut utiliser la technique suivante :

```
Matrix Mat;
double *Debut,*Fin,*p;

Mat=MatAlloc(Double,10,20);
Debut=*(MatGetDouble(Mat)); /* recuperation de l'adresse de début de la zone */
                            /* memoire contenant les elements de la matrice  */
Fin=Debut+10*20; /* calcul de l'adresse de fin de la zone */
for (p=Debut;p<Fin;*p++=3.14); /* affecte la valeur 3.14 a chaque */
                               /* element de la matrice        */
```

Attention : ce type de parcours doit être utilisé avec prudence à cause du risque de commettre une erreur et de son manque de lisibilité.

3.6 Copie d'une matrice

La fonction d'en-tête :

```
Matrix MatCopy(Matrix Mat);
```

retourne une copie de la matrice `Mat` ou `NULL` si un problème est détecté. L'allocation mémoire est effectuée par `MatCopy` mais sa libération est à la charge de l'appelant (voir le paragraphe 3.8).

3.7 Entrées-sorties

Les entrées-sorties de matrices sont effectuées en utilisant deux formats de fichiers : le format ASCII de base et le format *Matrix*. Dans les deux cas, il s'agit de fichiers de texte (ASCII).

— *Le format ASCII de base :* chaque ligne de la matrice correspond à une ligne du fichier et, au sein d'une même ligne, deux éléments sont séparés par un espace, une tabulation, une virgule, un point-virgule ou deux-points.
Exemple de fichier contenant une matrice de `int` *de 3 lignes et 4 colonnes :*

```
12 4 56 45
1 76 33 1
2 9 67 2
```

— *Le format Matrix :* le fichier contient sur la première ligne la chaîne de caractères "Matrix", le nombre de lignes et le nombre de colonnes de la matrice séparés par un espace sur la deuxième ligne et les lignes de la matrice sur les lignes suivantes. Sur une même ligne, deux éléments sont séparés par un espace.
Exemple de fichier contenant une matrice de `double` *de 2 lignes et 3 colonnes :*

```
Matrix
2 3
12.0 15.4 -3.5
3.45 .3 -3.98e-02
```

3.7.1 Lecture d'une matrice

La fonction d'en-tête :

```
Matrix MatReadAsc(char FileName[]);
```

retourne une matrice lue dans le fichier de nom `FileName`. Ce fichier peut être au format ASCII da base ou au format *Matrix*. Le type de la matrice est déterminé automatiquement en fonction du premier élément : si, dans le fichier, le premier élément de la matrice comporte un point, un `e` ou un `E`, la matrice est considérée de type `Double`, sinon elle est considérée de type `Int`. Si `FileName = ""` (la chaîne vide), la matrice est lue sur `stdin`; dans ce cas, le format attendu est *Matrix* ; cette possibilité peut être utilisée pour réaliser des filtres (on pourra s'inspirer du paragraphe 2.9 concernant les images). La réservation mémoire est réalisée par `MatReadAsc`. La libération est à la charge de l'appelant. Si un problème est détecté, la valeur `NULL` est retournée et des messages sont affichés sur `stderr`.

3.7.2 Écriture d'une matrice

La fonction d'en-tête :

```
void MatWriteAsc(Matrix Mat, char FileName[]);
```

écrit la matrice `Mat` dans le fichier de nom `FileName` au format *Matrix*. Si `FileName = ""` (la chaîne vide), l'image est écrite sur `stdout`.
Exemple :

```
Matrix Mat;
Mat=MatReadAsc("test.txt"); /* lecture de la matrice se trouvant dans text.txt */
if (Mat==NULL)
{
  fprintf(stderr,"Probleme de lecture du fichier test.txt\n");
  fprintf(stderr,"Liberation de la memoire...\n");
  MatFree(&Mat);
}
MatWriteAsc(Mat,""); /* affichage de la matrice sur stdout */
MatFree(&Mat); /* liberation de l'espace memoire occupe par Mat */
```

3.8 Libération de l'espace mémoire occupé par une matrice

La fonction d'en-tête :

```
void MatFree(Matrix *pMat);
```

libère l'espace mémoire occupé par la matrice pointée par `pMat` et affecte la valeur `NULL` à `*pMat`.
Attention : contrairement à la fonction standard `free`, c'est l'adresse de la matrice qui doit être passée et pas la matrice (`MatFree(&Mat);`).

4 Manipulation de listes

Merci de consulter les commentaires dans le fichier `limace.h`.