IMT Lille Douai
École Mines-Télécom
IMT-Université de Lille

Luc Fabresse
luc.fabresse@mines-douai.fr
DIA, Majeure ISIC

Des matrices

Écrire un programme qui permet de réaliser des opérations sur des matrices à 2 dimensions. On pose que les numéros de lignes/colonnes des matrices commencent à 0. Les matrices seront manipulées à travers les types suivants :

```c
typedef double Element;
struct MatriceStruct{
  Element **valeurs; //ligne * colonne
  int nbreLignes;
  int nbreColonnes;
};
typedef struct MatriceStruct Matrice;
```

Reprenez ces types dans votre programme puis définissez les fonctions suivantes :

1. `Matrice* creer(Element valeurInitiale, int nbreLignes, int nbreColonnes)`
   Créé (allocation mémoire) une matrice dont tous les éléments sont égaux à la valeur donnée en paramètre. Retourne un pointeur sur la nouvelle matrice.
2. `void detruire(Matrice* matriceP)`
   détruit (libération de la mémoire) la matrice référencée par le pointeur `matriceP`.
3. `void affiche(Matrice* matriceP)`
   affiche la matrice référencée par le pointeur `matriceP`.
4. `bool multiplicationPossible(Matrice* matriceAP, Matrice* matriceBP)`
   retourne `true` si la multiplication est possible et `false` sinon (cf. `stdbool.h`) dans le cas contraire. Rappel : la multiplication est possible si le nombre de colonnes de `matriceAP` est égal au nombre de lignes de `matriceBP`.
5. `Matrice* multiplier(Matrice* matriceAP, Matrice* matriceBP)`
   réalise la multiplication de deux matrices. Retourne un pointeur sur la matrice résultat ou NULL si impossible. Rappel : Soit A, B et C des matrices telles que $C = A.B$.
   $$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} . \begin{pmatrix} b_{11} & \dots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \dots & b_{pq} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1q} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mq} \end{pmatrix} \text{ où } \begin{cases} n = p \\ c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj} \end{cases}$$
6. Écrire une fonction permettant d'additionner deux matrices

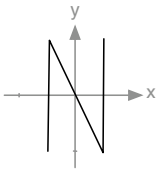
Utilisation des matrices pour dessiner !

Note : Réutiliser autant que possible les fonctions définies précédemment ainsi que le TP précédent (`libBMP`).

Un point P du plan sera représenté par une matrice colonne de dimension 3 : $P = (x, y, 1)$.

Pour représenter un polygone, il suffira d'ajouter autant de colonnes que de points nécessaires. Par exemple, la lettre N peut être entièrement définie par la donnée d'une matrice $\mathcal{N}$ de taille 4×3 contenant les coordonnées des 4 extrémités des segments (dans l'ordre de l'écriture, c'est-à-dire en commençant par le point en bas à gauche).

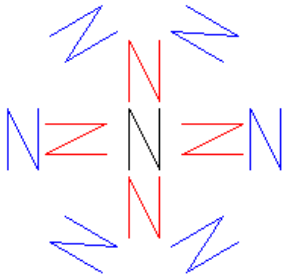
$$\mathcal{N} = \begin{pmatrix} -10.0 & -10.0 & 10.0 & 10.0 \\ -20.0 & 20.0 & -20.0 & 20.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \end{pmatrix}$$

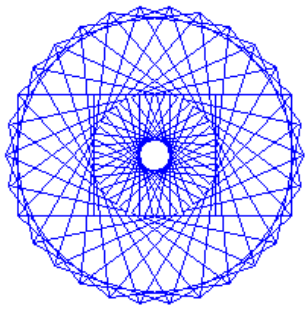


1. Ecrire une fonction `traceMatrice` qui étant donnés une image, une couleur et une matrice (contenant un nombre de points *quelconque*), trace en couleur dans l'image le polygone défini par la matrice. Attention : le repère d'une image et le repère utilisé pour exprimer les coordonnées dans une matrice sont différents ! Les coordonnées contenues dans une matrice, supposent que le point *(0,0)* est le centre du repère alors que dans une image, le point *(0,0)* est le coin supérieur gauche. De plus, l'orientation de l'axe des `Y` est inversé. Pour obtenir un dessin centrée dans l'image, il faut donc faire un changement de repère en tenant compte aussi des dimensions de l'image que l'on peut obtenir via les fonctions `GetWidth` et `GetHeight`.
2. Ecrire un programme principal permettant dessiner en noir la matrice $\mathcal{N}$ dans une image de 200×200. Vous devez obtenir le résultat présenté sur la figure ci-dessus centré dans l'image.
3. Écrire une fonction : `Matrice* translation(Matrice* figure,double x,double y)` qui translate la figure avec le vecteur $v(v_x, v_y)$. Une translation sera représentée par la matrice : $T_v = \begin{pmatrix} 1.0 & 0 & v_x \\ 0 & 1.0 & v_y \\ 0 & 0 & 1.0 \end{pmatrix}$
   Afficher en bleu la lettre N translatée par le vecteur (20.0,-20.0), résultat de l'opération : $N' = T_{(20,-20)}N$
4. Écrire une fonction : `Matrice* homotethie(Matrice* figure,double vx,double vy)`
   sachant que la matrice d'homotéthie de vecteur v et de centre $\mathcal{O}$ est : $H_v = \begin{pmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & 1.0 \end{pmatrix}$
   Effectuer le produit $N' = H_{(0.75,1.25)}N$.
5. Écrire une fonction : `Matrice* rotation(Matrice* figure,float angle)`
   sachant que la matrice de rotation d'angle θ et de centre $\mathcal{O}$ est : $R_\theta = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Pour avoir les fonctions `cos` et `sin`, vous utiliserez la bibliothèque `math.h`. Vous devrez faire `#include <math.h>` dans votre fichier et ajouter le paramètre `-lm` lors de l'édition des liens.

6. Écrire des programmes permettant d'obtenir les résultats suivants :





7. Question de réflexion : que pensez-vous des 3 fonctions précédentes en terme d'efficacité, surtout lorsque plusieurs transformations géométriques sont appliquées successivement ? Que faudrait-il modifier (en premier lieu) pour améliorer la rapidité de calcul ?