On souhaite disposer de l'opération d'addition de telles matrices : l'addition de deux matrices n'est possible que si ces matrices ont le même nombre de colonnes et le même nombre de lignes.

Une variable du type `matrix_t` vaut NULL si elle ne représente pas une matrice valide (par exemple, si on tente de construire une matrice avec une dimension négative ou si on tente d'additionner 2 matrices de tailles différentes).

Question.

1. Donnez les types décrits ci-dessus.
2. Donnez la définition d'une fonction de prototype

   ```
   matrix_t makenullmatrix ();
   ```

   qui construit une matrice zéro ;
3. Donnez la définition d'une fonction de prototype

   ```
   matrix_t makematrix (int, int);
   ```

   qui construit une matrice dont le nombre de lignes est passé en premier paramètre et le nombre de colonnes en second (si un de ces entiers est négatif ou nul, on retourne `NULL`). Cette fonction réserve de l'espace pour les coefficients mais n'affecte pas cet espace.
4. Donnez la définition d'une fonction de prototype

   ```
   void killmatrix (matrix_t);
   ```

   qui désalloue une matrice.
5. Donnez la définition d'une fonction de prototype

   ```
   matrix_t addmatrices (matrix_t,matrix_t);
   ```

   qui retourne la matrice résultant de l'addition des matrices passées en paramètres. Si ces matrices sont de dimensions différentes, cette fonction retourne `NULL`. Si chaque coefficient de la somme de ces matrices est zéro, on retourne une matrice zéro.

3 Mécanismes d'affectation dynamique de mémoire

On suppose disposer d'une zone mémoire :

```
#define NULL 0
#define BUFFERSIZE 65536
static char buffer[BUFFERSIZE] ;
```

On souhaite utiliser des portions de manière dynamique i.e. pouvoir disposer — quand c'est possible — d'un pointeur de type quelconque sur une portion de cet espace mémoire (sans possibilité de recouvrement avec d'autres portions pointées par d'autres pointeurs).

Pour ce faire, on se propose d'écrire les fonctions de prototype :

- `void *monalloc(unsigned int);` cette fonction retourne un pointeur de type `void` sur un espace mémoire de taille passée en paramètre à la fonction. Elle retourne `NULL` s'il est impossible de disposer de la quantité de mémoire demandée ;
- `void monfree(void *);` cette fonction libère l'espace mémoire associé au pointeur passé en paramètre.

Aucune autre fonction ou partie du code n'a à accéder au tableau `buffer` : ce dernier n'est manipulé que par l'intermédiaire des fonctions `monalloc` et `monfree`.