TP n°5 - Allocation dynamique sur le tas.

La gestion de la mémoire correspondant au segment du tas est :

dynamique : elle se fait au fil de l'exécution ;

manuelle : elle est à la charge du programmeur, de l'allocation à la libération des zones mémoires.

On rappelle ici les syntaxes en C des fonctions d'allocation et de libération de la mémoire dans le segment du **tas**. Ces fonctions se trouve dans la bibilothèque `stdlib`. Il faut donc, pour pouvoir les utiliser, rajouter la directive de prétraitement suivante :

```
# include <stdlib.h>
```

`malloc : void *malloc(size_t size)`

Cette fonction permet de réserver un certain nombre d'octets de mémoire **contigus** dans le tas et retourne l'adresse de la première case de la zone allouée. La taille de la zone mémoire à allouer (paramètre `size`) est donnée en octets. Il est possible d'utiliser la fonction `sizeof(`*`nom_du_type`*`)` pour avoir la taille d'un type en octets au format `size_t` et donc de calculer la taille de la zone à allouer en octets :

```
sizeof(type) * nombre_de_cases
```

`calloc : void *calloc(size_t n, size_t  size_type)`

Très semblable à `malloc`, sauf que l'on donne le nombre de cases et la taille en octet des cases séparément. De plus, cette fonction permet d'initialiser automatiquement tous les bits de la zone allouée à 0 ce qui n'est pas fait avec `malloc`.

`free : void free(void *ptr)`

Cette fonction permet de libérer une zone mémoire précédemment allouée et référencée par un pointeur `ptr`.

Exercice 1 (Allocation et libération).
Écrire un programme `allocation_dynamique.c` qui

- prend en entrée N valeurs entières données par l'utilisateur sur la ligne de commande
- les stocke dans un tableau d'entiers
- affiche le contenu du tableau grâce à la fonction `affiche_tableau` du précédent TP.

Toutes les instructions seront codées dans la fonction principale.
La fonction `affiche_tableau` du précédent TP sera recopiée au dessus de la fonction principale.
Voici un exemple d'exécution du programme :

```
golivier@ordiprof:~/doc/MPII/TP5$ ./allocation_dynamique 3 -5 9
Tableau alloué dynamiquement:
3 -5 9
golivier@ordiprof:~/doc/MPII/TP5$ ./allocation_dynamique 3 -5 9 7 2 -2 1 0 0
Tableau alloué dynamiquement:
3 -5 9 7 2 -2 1 0 0
```

Exercice 2 (Allocation et libération).
Écrire un programme `double_voyelles.c` qui :

- prend en entrée sur la ligne de commande un texte donné par l'utilisateur
- double toutes ses voyelles
- affiche la chaîne de caractères avec les voyelles dédoublées

On ne considérera que des textes en minuscules sans aucun accent.
Vous coderez toutes les instructions directement dans la fonction principale, en veillant toutefois à organiser proprement votre travail.
Vous pourrez utiliser la fonction `strlen` de la bibliothèque `string`.
Voici un exemple d'exécution du programme :

```
golivier@ordiprof:~/doc/MPII/TP5$ gcc double_voyelles.c -o double_voyelles -Wall
golivier@ordiprof:~/doc/MPII/TP5$ ./double_voyelles geraldine
Texte original: geraldine
Texte modifié : geeraaldiinee
golivier@ordiprof:~/doc/MPII/TP5$ ./double_voyelles "remy aime qu'on lui parle mal"
Texte original: remy aime qu'on lui parle mal
Texte modifié : reemyy aaiimee quu'oon luuii paarlee maal
golivier@ordiprof:~/doc/MPII/TP5$ ./double_voyelles
double_voyelles: double_voyelles.c:20: main: Assertion `argc == 2' failed.
Abandon (core dumped)
golivier@ordiprof:~/doc/MPII/TP5$
```

Pensez à programmer de manière défensive, à rendre votre code robuste grâce à la bibliothèque `assert`.

Exercice 3 (Allocation dans une fonction).
Copier votre fichier `double_voyelles.c` dans un autre fichier `double_voyelles_restruct.c`.
Puis, en ouvrant ce nouveau fichier, restructurez le code précédent pour que l'algorithme de dédoublement de voyelles soit codé à part dans une fonction `double_voyelles` qui prend en entrée le texte donné par l'utilisateur et retourne la taille de la nouvelle chaîne et le texte modifié.
Indication : passage par adresse
Voici un exemple d'exécution du programme :

```
golivier@ordiprof:~/doc/MPII/TP5$ gcc double_voyelles_restruct.c -o double_voyelles_restruct
golivier@ordiprof:~/doc/MPII/TP5$ ./double_voyelles_restruct "c'etait bien l'after hier?"
Texte original: c'etait bien l'after hier?, taille: 26 caractères
Texte modifié : c'eetaaiit biieen l'aafteer hiieer?, taille: 35 caractères
golivier@ordiprof:~/doc/MPII/TP5$
```

Exercice 4 (Linéarisation d'un tableau multi-dimensionnel).
En mathématiques, une matrice est un tableau de valeurs numériques à deux dimensions. La taille d'une matrice est un couple d'entiers naturels strictement positifs (N, M) correspondant respectivement au nombre de lignes et au nombre de colonnes de la matrice. On dit alors que la matrice est de taille $N \times M$. Un élément de la matrice est repéré par son double indice (i, j) où $1 \leq i \leq N$ et $1 \leq j \leq M$.

Écrire un programme `NOM_matrices.c` qui

- prend en entrée deux valeurs entières strictement positives N et M correspondant respectivement au nombre de lignes et au nombre de colonnes de la matrice ;
- alloue l'espace mémoire nécessaire au stockage d'une matrice $N \times M$ à valeurs entières ;
- calcule la matrice dont la valeur m_{ij} associée à l'indice (i, j) est donnée par la formule suivante :

$$m_{ij} = \begin{cases} i + j & \text{si } j \geq i \\ 0 & \text{sinon} \end{cases}$$

  Indication : Attention, au niveau informatique, l'indexation des tableaux commence à 0 !
- affiche cette matrice à l'écran sous la forme d'un tableau bidimensionnel.

Voici un exemple d'exécution du programme :

```
golivier@ordiprof:~/doc/MPII/TP5$ gcc matrices.c -o matrices -Wall
golivier@ordiprof:~/doc/MPII/TP5$ ./matrices 2 3
Matrice calculée de taille 2 par 3:
  0   1   2
  0   2   3

golivier@ordiprof:~/doc/MPII/TP5$ ./matrices 6 6
Matrice calculée de taille 6 par 6:
  0   1   2   3   4   5
  0   2   3   4   5   6
  0   0   4   5   6   7
  0   0   0   6   7   8
  0   0   0   0   8   9
  0   0   0   0   0  10

golivier@ordiprof:~/doc/MPII/TP5$ ./matrices 10 7
Matrice calculée de taille 10 par 7:
  0   1   2   3   4   5   6
  0   2   3   4   5   6   7
  0   0   4   5   6   7   8
  0   0   0   6   7   8   9
  0   0   0   0   8   9  10
  0   0   0   0   0  10  11
  0   0   0   0   0   0  12
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
```

- Structurez votre programme en le découpant en trois fonctions : `allouer_matrice`, `remplir_matrice` et `afficher_matrice`
- Stockez votre matrice dans un grand tableau uni-dimensionnel (cf cours séquence 4)
- La fonction d'affichage de la matrice doit être appelée dans le main (pour vous obliger à faire en sorte que la matrice survive à la fonction `allouer_matrice`)
- Testez votre code, par exemple sur les mêmes exemples que moi
- Pensez à bien regarder les corrigés des TPs précédents pour essayer de "coller" aux attendus et aux bonnes pratiques de programmation.
- Pour l'affichage, on pourra utiliser le format `%3d` au lieu de `%d` pour forcer l'affichage de la valeur sur 3 emplacements quelque soit le nombre de chiffres du nombre, et ainsi garantir un bel alignement des valeurs de votre tableau.