Exercice 1 : Lecture de fichier sur le *heap*

On souhaite lire un fichier dont la taille n'est pas connue avant l'exécution du programme. Un espace mémoire doit alors être réservé sur le *heap*. Écrire la fonction `load` permettant de charger l'ensemble du fichier en mémoire dans la variable `data`.

```c
int main(int argc, char*argv[]) {
  assert(argc > 1);
  FILE *fp = fopen(argv[1], "r");
  assert(fp != NULL);
  char *data;
  load(&data, fp);
  fclose(fp);
}
```

```c
void load(char **data, FILE*fp) {
  assert(fp != NULL);
  fseek(fp, 0, SEEK_END);
  size_t size = ftell(fp);
  fseek(fp, 0, SEEK_SET);
  *data = malloc(size);
  assert(*data != NULL);
  fread(*data, 1, size, fp);
}
```

Exercice 2 : Membre flexible

Une structure avec un membre flexible est défini comme suit :

```c
typedef struct point {
  int x; int y;
} Point;

typedef struct record {
  int id;
  Point points[];
} Record;
```

(a) Quelle est la taille de la structure record en bytes ?

> Le membre flexible n'est pas inclus dans la structure, cette dernière est donc la taille de l'entier 32-bit, soit 4 bytes.

(b) Déclarez la variable `r` puis allouer l'espace mémoire nécessaire pour y stocker 10 points.

```c
Record *r = malloc(sizeof(Record) + sizeof(Point) * 10);
assert(r != NULL);
```

(c) Redimensionnez la variable `r` pour y stocker jusqu'à 100 points.

```
void *tmp = realloc(r, sizeof(Record) + sizeof(Point) * 100);
assert(tmp != NULL);
r = tmp;
```

(d) À la fin de l'exercice, il est nécessaire de libérer l'espace mémoire utilisé. Comment libérez-vous l'espace alloué ?

```
free(r);
```

Exercice 3 : Redimensionnement

Une fonction s'occupe de redimensionner un set de données pour pouvoir y stocker `n` valeurs. Écrire la fonction `resize`.

(a) Dans le premier cas le set de donnée est défini comme suit :

```
typedef struct measurements {
  size_t elements; // Nombre d'éléments dans le set
  size_t capacity; // Capacité en éléments de l'espace alloué
  int *data;
} Measurements;
```

```
int resize(Measurements *meas, size_t n) {
  if (n < meas->elements)
    return 1;
  if (meas->data == NULL) {
    meas->data = malloc(sizeof(Measurements.data[0]) * n);
    if (meas->data == NULL) return 2;
    meas->capacity = n;
    return 0;
  }
  meas->capacity = n;
  void *tmp = realloc(meas->data, sizeof(Measurements.data[0]) *
↪  meas->capacity);
  if (tmp == NULL) return 3;
  meas->data = tmp;
  return 0;
}
```

(b) Dans le second cas le set de donnée est défini à l'aide d'un membre flexible :

```
typedef struct measurements {
  size_t elements; // Nombre d'éléments dans le set
  size_t capacity; // Capacité en éléments de l'espace alloué
  int data[];
} Measurements;
```

```
int resize(Measurements **meas, size_t n) {
  if ((*meas) == NULL) {
    (*meas) = malloc(sizeof(Measurements) + n *
↪   sizeof(Measurements.data[0]));
    if ((*meas) == NULL) return 2;
    (*meas)->capacity = n;
    (*meas)->elements = 0;
    return 0;
  }
  if (n < (*meas)->elements)
    return 1;
  (*meas)->capacity = n;
  void *tmp = realloc((*meas),
    sizeof(Measurements) + n * sizeof(Measurements.data[0]));
  if (tmp == NULL) return 3;
  (*meas) = tmp;
  return 0;
}
```

Exercice 4 : Sudoku

Dans un jeu de Sudoku, on souhaite allouer l'espace nécessaire pour y stocker une grille dont la taille (nombre de colonnes) est déterminée à l'exécution du programme.

La grille est stockée dans un pointeur sur un tableau à deux dimensions, dont chaque dimension est la taille de la variable `columns`.

(a) Écrire la structure de donnée et allouer cette grille en initialisant toutes les valeurs à zéro.

```
const int columns = 10;
assert(columns < UINT8_MAX);
uint8_t (*grid)[columns] = calloc(columns, sizeof(uint8_t[columns]));
assert(grid != NULL);
```

(b) Affichez la grille sur la sortie standard.

```
for(int i = 0; i < columns; i++) {
    for(int j = 0; j < columns; j++) {
        printf("%d ", (*grid)[i][j]);
    }
    printf("\n");
}
```