Dans ce contrôle TP vous allez implémenter quelques structures de données usuelles permettant de stocker des entiers. Vous respecterez **scrupuleusement** les instructions, en particulier les déclarations de types et les prototypes de fonctions.

Vous rendrez l'intégralité des exercices dans un seul fichier .c dans lequel vous indiquerez vos nom, prénom et groupe dans un commentaire situé au début du fichier. Ce commentaire **DOIT** être la première chose que l'on trouvera dans le fichier !

Votre code **doit** compiler sans erreur avec les options de compilation `-Wall -Werror -ansi -pedantic`

Exercice 1 : Pile *(7 points)*

On se propose ici d'implémenter une pile. Dans une pile, le dernier élément stocké sera le premier élément récupéré. Notre pile n'aura pas de limite de taille autre que celle imposée par le système et vous aurez donc à utiliser `malloc` et `realloc`. La structure de donnée représentant la pile est la suivante :

```
struct pile
{
  int *valeurs;
  int sommet_de_pile;
  int taille_max;
};
```

Q 1. Implémentez la fonction

```
int pile_init(struct pile *p, int taille_initiale);
```

qui initialise la structure de donnée passée en paramètre. Une fois l'initialisation effectuée, le pointeur `valeurs` pointera vers une zone mémoire allouée par malloc de `taille_initiale` entiers. Le champ `sommet_de_pile` vaudra -1 et `taille_max` vaudra `taille_initiale`.

La fonction devra retourner 0 si l'initialisation s'est bien déroulée et -1 en cas d'erreur.

Q 2. Implémentez la fonction

```
void pile_push(struct pile *p, int valeur);
```

qui empile une valeur dans la pile pointée par `p`. On considère ici que la pile a été préalablement initialisée avec la fonction précédente. Empiler une valeur consiste à incrémenter `somme_de_pile` et à stocker la valeur dans `valeurs[sommet_de_pile]`. Si la pile est pleine avant d'insérer une nouvelle valeur, vous augmenterez la taille de la pile à l'aide de la fonction `realloc`. Cette augmentation devra être d'au minimum 16 entiers.

Q 3. Implémentez la fonction

```
int pile_pop(struct pile *p);
```

qui dépile la prochaine valeur de la pile. Si la pile est vide, la fonction retournera 0.

Exercice 2 : File *(7 points)*

Cette fois, nous allons implémenter une file. A l'inverse d'une pile, dans une file le premier élément inséré et le premier élément récupéré. La structure de données est maintenant la suivante :

```
struct file
{
  int *valeurs;
  int premier;
  int dernier;
  int taille_totale;
};
```

Q 1. Implémentez la fonction :

```
int file_init(struct file *f);
```

qui initialise la file. Comme précédemment, la fonction retournera 0 en cas de succès et -1 en cas d'échec.

Q 2. Implémentez la fonction :

```
void file_put(struct file *f, int valeur);
```

qui insère un nouvel entier dans la file. Comme précédemment, vous devrez utiliser `malloc` et `realloc` pour que la taille de la file ne soit limitée que par le système.

Q 3. Implémentez la fonction :

```
int file_get(struct file *f);
```

qui retourne l'élément suivant dans la file. Si la file est vide, la fonction retournera 0.

Pour cet exercice, un bonus sera accordé si l'implémentation est efficace en terme de mémoire (c'est à dire si votre solution s'arrange pour ne gâcher que peu de cases mémoires non utilisées).