Étude du préprocesseur...

▸ **Exercice 1**. Donner le résultat de l'exécution du programme suivant :

```
#define A       -1
#define B(A)    2*A+A
#define C       3A

void main(void)
{ int i = 1;
  printf("premier printf\n");

#define printf(A)       printf("A et B sont ici\n");
  printf("C'est fini\n");
  printf("C\n");
#undef printf

  printf("A %d\n",A);

#ifdef C
  #undef C
  #define C       "%d\n"
#else
  printf("Attention, C n'etait pas defini\n");
#endif

  printf("C"C,B(i+1));
}
```

Étude de la gestion d'une pile par tableau...

On désire dans cette partie "simuler une pile" à l'aide d'un tableau. Pour ce faire, on a besoin de définir un type `pile` regroupant :

1. un tableau `data` qui stockera les données,
2. un "pointeur de pile" de nom `sommet` qui sera dans ce cas précis un entier représentant le numéro de la case du tableau dénotant le sommet de la pile.

▸ **Exercice 2**. Définir le type *C* permettant de manipuler une pile par cette technique.

▸ **Exercice 3**. Écrire les fonctions usuelles sur les piles :

1. `Empty(p)` teste si la pile `p` est vide ou non,
2. `push(x, p)` empile l'élément `x` sur la pile `p`,
3. `pop(p)` dépile —et renvoie— le sommet de la pile `p`,

Les fonctions d'allocation dynamique de mémoire

La bibliothèque standard propose essentiellement trois fonctions d'allocation dynamique de mémoire. On donne ci-dessous les prototypes de ces fonctions, ainsi que leurs principales utilisations :

La fonction `malloc()` renvoie une zone mémoire dont la taille `size` est passée en argument; la fonction renvoie `NULL` si l'allocation est impossible.

```
void * malloc(size_t size);
```

La fonction `calloc()` est basée sur le même principe, mais on lui passe en argument le nombre d'objets `nitems` de taille `size` : la zone mémoire demandée a donc une taille de `nitems`×`size`. Par ailleurs, la zone renvoyée est initialisée par des zéros.

```
void * calloc(size_t nitems, size_t size);
```

La fonction `realloc()` est un peu spéciale dans le sens où elle permet de réallouer une zone mémoire, c'est à dire d'en changer sa dimension. Si vous allouez un tableau de 10 entiers, et qu'au cours de l'exécution de votre programme, vous avez besoin de stocker 20 entiers, il faut réallouer la zone : vous n'avez pas à libérer l'ancienne zone et la recopie des données de l'ancienne zone vers la nouvelle est réalisée par le système. A noter que l'adresse de la nouvelle zone n'est pas forcément différente de l'adresse de l'ancienne zone.

```
void * realloc(void * block, size_t size);
```

Enfin, la fonction `free()` permet de libérer une zone allouée par votre programme.

```
void free(void * block);
```

Tout `malloc()`, `realloc()` ou `calloc()` doit être associé à un appel à la fonction `free()`. D'autre part, vous n'avez pas à libérer des objets que vous n'avez pas alloués —objets alloués par le système par exemple...—.

▸ **Exercice 4**. Cet exercice consiste à afficher les arguments de la ligne de commande. On demande un programme ne comportant qu'une fonction `main`, qui réalise cet affichage. Si votre programme s'appelle `aff`, on aura par exemple le résultat suivant :

```
$ aff ceci est le test numero 1
aff ceci est le test numero 1
$
```

ou encore,

```
$ aff aff marche bien
aff aff marche bien
$
```

On rappelle que l'on accède aux arguments de la ligne de commande en précisant que la fonction `main()` reçoit des arguments.

```
void main(int argc, char *argv[])
{

}
```

Le fait d'utiliser cette définition signifie que le paramètre `argc` contiendra le nombre d'arguments de la ligne de commande —en comptant la commande elle-même—, et le paramètre `argv` est un tableau de `argc` chaînes de caractères qui correspondent dans l'ordre aux arguments de la ligne de commande. En particulier, si on réalise un `printf("%s\n",argv[0])`, on affichera le nom du programme lui-même, ce qui sous-entend qu'un programme a accès au nom par lequel il a été lancé !

▸ **Exercice 5**. On demande, comme application de cette partie, de dupliquer les arguments de la ligne de commande, puis de libérer ce que vous avez créé. Le but de l'exercice est donc de créer une "structure", réplique exacte de la "structure" stockant les arguments de la ligne de commande.