Séance 5 : Modularité et compilation

L2 Informatique – Université Côte d'Azur

Exercice 1 — Vérification gestion mémoire

Partie I : Préparation du répertoire

1. Dans le répertoire `tp5`, veillez créer un nouveau repertoire `exercice1/`. Créez un fichier `main.c` se contetant d'afficher « `Hello World !` » et un fichier `Makefile` comme vue dans le cours 5. Une fois que tout marche bien passez à la question suivante.

Partie II : Écriture de la bibliothèque

On veut s'assurer que notre gestion de la mémoire est correcte. Une première stratégie revient à définir une variable statique `NB_ALLOC` initialisée à `0`. On définira catte variable en dehors de toute fonction.

Lorsqu'une allocation est faite, on incrémente cette variable. On la décremente lors de l'appel de `free`. Pour le faire automatiquement sans se soucier de `NB_ALLOC`, on utilisera dans notre code principal les fonctions `mon_malloc` et `mon_free` à la place des fonctions `malloc` et `free`.

2. Dans un fichier `allocation_simple.c` écrire les deux fonctions suivantes. Ces fonctions se contenteront d'appeler les fonctions correspondantes de la bibliothèque `stdlib.h` en gérant la variable statique `NB_ALLOC`.
   (a) `void *mon_malloc(size_t size)`
   (b) `void mon_free(void *ptr)`
3. En créant un fichier `allocation_simple.h`, utilisez les deux fonctions précédentes dans votre fichier `main.c`.
4. Afin de tester votre bibliothèque, ajouter à `allocation_simple` une fonction `int bilan(void)` qui renvoie la valeur de la variable statique `NB_ALLOC` après avoir affiché un message de la forme « `Il reste 3 pointeurs à libérer.` ».
5. Que se passe-t-il si vous quittez votre fonction `main` en appelant `return bilan();`
6. Ajoutez une option `MODE_DEBUG` pour choisir à la compilation d'afficher ou non ces messages.
7. Complétez votre bibliothèque avec les fonctions :
   (a) `void *mon_calloc(size_t nmemb, size_t size)`
   (b) `void *mon_realloc(void *ptr, size_t size);`

Exercice 2 — Un gestionnaire de mémoire plus avancés

Même exercice que le précédent. Cette fois-ci, au lieu de simplement compter le nombre d'allocations et de libérations, on va stocker dans une liste chaînée tous les pointeurs alloués et non encore libérés. On stockera aussi la ligne et le fichier à laquelle l'allocation a été faite.

Attention, cette exercice est complexe et non guidé. C'est à vous de créer les fichiers nécessaires, vos propres bibliothèques, réfléchir à comment utiliser les macro, etc.

Voici, page suivante, un exemple de fichier `main.c` utilisant notre bibliothèque.

```c
#include <stdio.h>
#include <stdlib.h>
#include "allocation.h"

int main(void) {
  int *p;
  int *q;
  int *r;

  p = allocation_malloc(sizeof(int));
  allocation_bilan();

  q = allocation_malloc(sizeof(int));
  r = allocation_malloc(sizeof(int));
  allocation_bilan();

  allocation_free(q);
  allocation_bilan();

  allocation_free(r);
  allocation_bilan();

  allocation_free(p);
  allocation_bilan();

  return 0;
}
```

Et voici l'affichage correspondant.

stdin/stdout

```
 0 free/1 malloc
== Début table allocation ==
0x5646423982a0 ligne 11 « main.c »
== Fin table allocation ==

 0 free/3 malloc
== Début table allocation ==
0x564642398340 ligne 13 « main.c »
0x5646423982f0 ligne 12 « main.c »
0x5646423982a0 ligne 11 « main.c »
== Fin table allocation ==

 1 free/3 malloc
== Début table allocation ==
0x564642398340 ligne 13 « main.c »
0x5646423982a0 ligne 11 « main.c »
== Fin table allocation ==

 2 free/3 malloc
== Début table allocation ==
0x5646423982a0 ligne 11 « main.c »
== Fin table allocation ==

 3 free/3 malloc
== Début table allocation ==
== Fin table allocation ==
```