TP1 : Fuites Mémoire et Pointeurs fous

Buts :
— montrer les pièges des pointeurs fous et des fuites de mémoire en C
— donner des pistes pour les corriger

# 1 Installation

1. Récupérez l'archive `TP1_Fuites.tar.gz` à partir :
   — de ma page *Enseignements* à `http://www-info.univ-lemans.fr/~jacob/enseignement.html` dans la rubrique *Programmation C*
   — du serveur du département informatique à `/info/tmp/AnnexesTPL2SPI/TP_Fuites/TP_Fuites.tar.gz`
2. Décompressez la et désarchivez la par : `tar xvfz TP1_fuites.tar.gz`

Vous devez normalement obtenir un répertoire `TP_Fuites` dans lequel il faut aller pour faire ce TP.

# 2 Exercice 1

1. Allez dans le répertoire `Exercice1`
2. Compilez le programme `prog1` en tapant la commande `make -f Makefile all`
3. Exécutez `prog1`. Normalement, celui-ci ne fonctionne pas.
4. Corrigez le programme `prog1.c` pour qu'il fonctionne en indiquant dans des commentaires la raison de l'erreur et comment vous l'avez résolue.

# 3 Exercice 2

Compilez les sources de l'exercice 2 :

1. Allez dans le répertoire `Exercice2`
2. Compilez les programmes et les modules en tapant la commande `make -f Makefile all`
3. Vérifiez en l'exécutant que le programme `prog2` comporte des fuites de mémoire. Le but de cet exercice est de les détecter et de les corriger.

Voici des propositions de méthodes pour corriger ces fuites de mémoire :

1. Complétez les corps des fonctions "*_detruire" (en regardant les sources, on s'apperçoit qu'il n'y a aucun `free` et c'est donc là qu'il faut les mettre)
2. Compter le nombre de références sur chaque objet. Pour cela, modifiez les modules `obj*.[ch]` pour qu'ils gèrent ces compteurs : chaque objet créé incrémente un compteur, chaque objet détruit décrémente ce compteur. Normalement, à la fin du programme, les compteurs de tous les objets doivent être égaux à zéro. Si ce n'est pas le cas alors vérifiez que le nombre de créations est égal au nombre de destructions des objets
3. Vérifiez par `valgrin` s'il ne reste pas des fuites de mémoire (si les fonctions "*_detruire" sont bien programmées). Pour cela, il faut faire, si on est dans le répertoire Exercice2 :

   ```
   ../valgrind-3.7.0/bin/valgrind --leak-check=full <nom du programme>
   ```

   Si dans le résumé à la fin du rapport de `valgrin` il y a

   ```
   ==95358== LEAK SUMMARY:
   ==95358==    definitely lost: 0 bytes in 0 blocks
   ==95358==    indirectly lost: 0 bytes in 0 blocks
   ==95358==      possibly lost: 0 bytes in 0 blocks
   ```

   alors il n'y a pas de fuite mémoire.

Si, en revanche il y a

```
40 bytes in 1 blocks are definitely lost in loss record 1 of 6
==95367==    at 0x1227F: malloc (vg_replace_malloc.c:266)
==95367==    by 0x25B0: obj1s_creer (in ./test_obj1s)
==95367==    by 0x2192: main (in ./test_obj1s)
==95367==
```

cela veut dire que la mémoire réservée (40 ø) par le `malloc` de la fonction `obj1s_creer` n'est jamais désallouée (les 40 ø sont définitivement perdus). Si on a :

```
==95367== 40 bytes in 1 blocks are definitely lost in loss record 2 of 6
==95367==    at 0x1227F: malloc (vg_replace_malloc.c:266)
==95367==    by 0x25B0: obj1s_creer (in ./test_obj1s)
==95367==    by 0x2890: obj1s_copier (in ./test_obj1s)
==95367==    by 0x2272: main (in ./test_obj1s)
```

alors il y a une fuite mémoire car un `malloc` de la fonction `obj1s_creer` appelée par `obj1s_copier` n'a pas de `free` correspondant.
Dans le résumé les compteurs d'octets perdus sont différents de zéro :

```
LEAK SUMMARY:
==95367==    definitely lost: 80 bytes in 2 blocks
==95367==    indirectly lost: 0 bytes in 0 blocks
==95367==      possibly lost: 0 bytes in 0 blocks
```

Si l'endroit où on perd la référence sur la mémoire (donc la fuite) n'est pas évidente alors on peut procéder comme suit :

— À un endroit critique du programme on met un `exit(0)`, on compile puis on lance le programme avec `valgrin`
— S'il n'y a pas de fuite alors c'est qu'elle se produit plus loin sinon c'est qu'elle se produit avant le `exit`
— en fonction du résultat déplacer le `exit`

Ainsi on peut identifier la ligne qui produit la fuite mémoire (un `free` ou une mauvaise affectation de pointeur par exemple).

Le but de cet exercice est de faire exécuter prog2 en le vérifiant avec `valgrin` jusqu'à ce qu'il n'y ait plus de fuite mémoire.