Mini Projet 1 : Outils de débogue, tableaux et complexité

Exercise 1

Q 1.1

La fonction permet de remplir un tableau de 10 entiers. Lors de son exécution, on constate un `segmentation fault(core dumped)`

A la sortie de la boucle `for`, `i` vaut `-1` or `i` n'est pas signée.

Pour résoudre le problème, on remplit notre tableau dans l'ordre croissant, pour être sûr que la variable `i` ne soit pas négative de sorte à commencer par le premier indice `0`.

Q 1.2

```
=thread-group-added,id="i1"
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
Warning: Debuggee TargetArchitecture not detected, assuming x86_64.
=cmd-param-changed,param="pagination",value="off"
Stopped due to shared library event (no libraries added or removed)
Loaded '/lib64/ld-linux-x86-64.so.2'. Symbols loaded.

Breakpoint 1, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:10
10    tab = (int *)malloc(len * sizeof(int));
Loaded '/lib/x86_64-linux-gnu/libc.so.6'. Symbols loaded.

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;
Execute debugger commands using "-exec <command>", for example "-exec info registers" will
list registers in use (when GDB is the debugger)

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;
```

```
Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Breakpoint 2, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;

Program received signal SIGSEGV, Segmentation fault.
0x0000000008001174 in main () at /home/zhenyue/data-structure-
in006/code_exercice1/tme1_exo1p1.c:13
13      tab[i] = i;
Kill the program being debugged? (y or n) [answered Y; input not from terminal]
[Inferior 1 (process 9100) killed]
The program '/home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p1' has exited with
code 0 (0x00000000).
```

Q 1.3

On modifie la ligne 12 `for (i = 0; i < len; i++)`

Q 1.4

La fonction crée une structure Adresse nommée « maison » dont les éléments qui la compose sont : le numéro, la rue ainsi que le code postal.

La fonction est censée afficher :

Adresse courante : 12 rue manoeuvre 15670 France

A l'exécution, on constate un `segmentation fault(core dumped)`

Q 1.5

`new->rue=0x0`

Nous n'avons pas alloué de mémoire pour la `rue`, donc nous ne pouvons pas le copier.

Le programme est sorti à la ligne 16.

La chaine de caractère qui doit stocker le nom de la rue n'a pas été alloué statistiquement ou dynamiquement. La mémoire allouée doit être fixe c'est pour cela qu'on décide d'utiliser `strdup` qui copie la chaine mais alloue l'espace nécessaire au stockage en mémoire de la copie.

On oubli pas de libérer la mémoire alloué destiné à stocker le nom de la rue。

Pour éviter d'oublier de libérer la mémoire, nous allouons la mémoire manuellement au lieu d'utiliser des `strdup`

```
Loaded '/lib64/ld-linux-x86-64.so.2'. Symbols loaded.

Breakpoint 1, main () at /home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p2.c:22
22    Adresse* maison = creer_adresse(12, "manoeuvre", 15670);
Loaded '/lib/x86_64-linux-gnu/libc.so.6'. Symbols loaded.

Breakpoint 2, creer_adresse (n=12, r=0x8002008 "manoeuvre", c=15670) at /home/zhenyue/data-
structure-in006/code_exercice1/tme1_exo1p2.c:15
15    strcpy(new→rue, r);
Execute debugger commands using "-exec <command>", for example "-exec info registers" will
list registers in use (when GDB is the debugger)

Program received signal SIGSEGV, Segmentation fault.
__strcpy_sse2_unaligned () at ../sysdeps/x86_64/multiarch/strcpy-sse2-unaligned.S:668

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
The program '/home/zhenyue/data-structure-in006/code_exercice1/tme1_exo1p2' has exited with
code 0 (0x00000000).
```

Q 1.6

La fonction crée une structure `Tableau t` composé d'un tableau d'entier, une taille définie, ainsi qu'un repère position qui va permettre de placer nos entiers au bon indice à chaque appel de `ajouterElement`.

La fonction affiche sans aucune interruption :

```
t→position = 5
[ 5 18 99999 -452 4587 ]
```

Mais en regardant de plus près avec `valgrind`, 3 allocations et 2 libérations de mémoires ont été effectuées et il y a une fuite mémoire de 400 bytes. On remarque que le tableau de la structure n'a pas été libéré.

```
==10143== Memcheck, a memory error detector
==10143== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10143== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10143== Command: ./tme1_exo1p3
==10143==
==10143== error calling PR_SET_PTRACER, vgdb might block
```

```
t→position = 5
[ 5 18 99999 -452 4587 ]
==10143==
==10143== HEAP SUMMARY:
==10143==     in use at exit: 400 bytes in 1 blocks
==10143==   total heap usage: 3 allocs, 2 frees, 1,440 bytes allocated
==10143==
==10143== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==10143==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==10143==    by 0x1091DE: initTableau (in /home/zhenyue/data-structure-
in006/code_exercice1/tme1_exo1p3)
==10143==    by 0x109288: main (in /home/zhenyue/data-structure-
in006/code_exercice1/tme1_exo1p3)
==10143==
==10143== LEAK SUMMARY:
==10143==    definitely lost: 400 bytes in 1 blocks
==10143==    indirectly lost: 0 bytes in 0 blocks
==10143==      possibly lost: 0 bytes in 0 blocks
==10143==    still reachable: 0 bytes in 0 blocks
==10143==         suppressed: 0 bytes in 0 blocks
==10143==
==10143== For counts of detected and suppressed errors, rerun with: -v
==10143== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Q 1.9

On rajoute donc une ligne `free(t->tab)` avant de libérer la structure.

```
==10269== Memcheck, a memory error detector
==10269== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10269== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10269== Command: ./tme1_exo1p3
==10269==
==10269== error calling PR_SET_PTRACER, vgdb might block
t→position = 5
[ 5 18 99999 -452 4587 ]
==10269==
==10269== HEAP SUMMARY:
==10269==     in use at exit: 0 bytes in 0 blocks
==10269==   total heap usage: 3 allocs, 3 frees, 1,440 bytes allocated
==10269==
==10269== All heap blocks were freed -- no leaks are possible
==10269==
==10269== For counts of detected and suppressed errors, rerun with: -v
==10269== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Exercise 2

Partie 1

Le deuxième algorithme utilise le

$$2n \times \sum_{i=1}^{n} t_i^2 - 2 \times (\sum_{i=1}^{n} t_i)^2$$

La courbe résultante est indiquée dans le fichier `ex02/01_courbes_vitesse.pdf`.

Au vu des courbes résultantes, on peut remarquer que le premier algorithme est plus conséquent en temps d'exécution car l'algorithme contient deux boucles for imbriquées, de complexité de n^2 tandis que le deuxième algorithme n'a qu'une seul boucle et de complexité n, ce qui a son sens.

Partie 2

Toutes les fonctions sont dans `matrix.c`.

Les deux main programmes sont `matrix_diff.c` et `matrix_produit.c`

J'ai toujours utilisé un tableau 1 dimensionnel pour stocker la matrice.

J'utilise `m[i*n+j]` ou `*(m+i*n+j)` au lieu de `m[i][j]`.

Plus précisément, comme indiqué dans la classe, la multiplication est plus rapide dans le système que la lecture de la mémoire.

Et il est plus facile d'allouer de l'espace mémoire et de le libérer pour la matrice.

Je remplace également la matrice triangulaire par un tableau unidimensionnel de taille `n*(n+1)/2`, où `m[n*i-i*(i+1)/2+j]` représente la matrice triangulaire supérieure et `m[i*(i+1)/2+j]` représente la matrice triangulaire inférieure.

Cela permet d'éviter la nécessité de générer des tableaux complexes à deux dimensions, et il n'est pas nécessaire de créer des structures différentes pour les matrices triangulaires supérieure et inférieure.

matrice ont des valeurs différentes

La courbe résultante est indiquée dans le fichier `ex02/02_courbes_vitesse.pdf`.

Le premier algorithme utilise 4 niveaux de boucles, tandis que le second utilise l'espace pour le temps, n'utilise que 2 niveaux de boucles, et est plus susceptible de trouver des valeurs doubles à l'avance et à la sortie.

matrice produit

La courbe résultante est indiquée dans le fichier `ex02/matrice_produit_courbes_vitesse.pdf`.

Dans le premier algorithme, la moitié des valeurs 0 sont stockées dans la matrice.

Et le second algorithme saute ces valeurs 0, optimisant ainsi l'espace et le temps.