Méthode : Déchiffrer une erreur en C

L'objectif de ce document est d'aider à lire les messages d'erreurs donnés par un compilateur C ou un sanitizer. Ces messages peuvent être des messages d'erreur ou de warning, être générés à la compilation ou à l'exécution : les éléments importants à trouver et comprendre sont les mêmes dans tous les cas.

Rappel : un warning n'est pas "moins grave" qu'une erreur : il signale une erreur qui n'a pas interrompu la compilation mais une erreur (de syntaxe ou de conception) tout de même. Il ne faut pas s'en satisfaire.

Rappel : une compilation et une exécution sans erreur ni warning ne garantit pas la correction du code.

Pour comprendre un message d'erreur, le principe générique est de chercher les informations suivantes :

- La nature de l'erreur (encadrée en vert dans les exemples).
- La ligne probable causant l'erreur (encadrée en rouge dans les exemples).
- La fonction probable causant l'erreur (encadrée en bleu dans les exemples).

Le terme "probable" indique la possibilité que l'endroit à modifier ne soit pas toujours à la ligne indiquée dans la fonction indiquée (par exemple, une fonction correcte `f1` mais utilisée de manière incorrecte dans `f2` peut être incriminée par une erreur alors que le problème est à corriger dans `f2`) mais c'est a minima une première piste à investiguer. Ces informations occupent parfois une part réduite du message d'erreur, la difficulté est donc de les trouver au milieu des informations qui ne vous sont pas utiles.

Sélection de messages d'erreurs expliqués

Deux messages simples obtenus à la compilation

Mettons en application le principe susmentionné dans un cas simple :

```
arbre.c: In function 'cherche':
arbre.c:165:36: error: lvalue required as left operand of assignment
  165 |         if (compare(valeur(a),val) = 0)
      |                                    ^
```

- La nature de l'erreur est souvent indiquée après un mot "error" écrit dans une couleur visible (idem pour un warning). Ici, lire l'erreur nous dit qu'on essaie de faire une affectation alors que le membre de gauche n'est pas une lvalue, mais en fait le souci vu le contexte est qu'on a confondu == et =.
- La ligne où intervient l'erreur est indiquée juste avant. Ici, l'erreur est en ligne 165 (ce qui est d'ailleurs confirmé la ligne d'en dessous où le contenu de cette ligne est rappelé) et en colonne 36.
- La fonction où intervient l'erreur est la fonction `cherche` : c'est explicitement indiqué dans ce cas.

On obtient exactement les mêmes informations dans le cas d'un warning :

```
TPDM4.c: In function 'determiniser':
TPDM4.c:253:11: warning: unused variable 'finaux' [-Wunused-variable]
  253 |     bool* finaux = (bool*)malloc(sizeof(bool)*(res->taille_Q));
      |           ^~~~~~
```

Ici, le problème se situe dans la fonction `determiniser`, en ligne 253 et colonne 11 et est qu'une variable, à savoir `finaux`, n'est pas utilisée dans la fonction alors qu'elle y est déclarée et initialisée, ce qui est suspect.

En cas d'erreurs et warnings multiples, les blocs d'erreurs tels que les précédents se suivent dans la console et il faut alors délimiter, analyser et corriger chacune.

Une erreur de lecture

Observons à présent le message suivant pour illustrer un concept générique : à l'éxecution, lorsqu'une fonction appelle une fonction qui appelle une fonction qui appelle ... qui appelle une fonction provoquant un comportement problématique, le message d'erreur explicite l'historique de remontée de l'erreur.

```
arbre.c:41:13: runtime error: member access within null pointer of type 'struct noeud'
AddressSanitizer:DEADLYSIGNAL
=================================================================
==609784==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000010 (pc 0x5563589baa68 bp
==609784==The signal is caused by a READ memory access.
==609784==Hint: address points to the zero page.
    #0 0x5563589baa67 in gauche /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c 41
    #1 0x5563589bacab in libere /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:70
    #2 0x5563589bacb3 in libere /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:70
    #3 0x5563589bacb3 in libere /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:70
    #4 0x5563589bacb3 in libere /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:70
    #5 0x5563589bd397 in main /home/aude/Documents/MPI/C_Ocaml/Tests/magasin.c:57
    #6 0x7fbde47c4082 in __libc_start_main ../csu/libc-start.c:308
    #7 0x5563589ba36d in _start (/home/aude/Documents/MPI/C_Ocaml/Tests/magasin+0x436d)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:41 in gauche
==609784==ABORTING
```

Comme dans les cas précédents, la nature de l'erreur est indiquée à la suite d'un "error". Contrairement aux cas précédents, on a un peu plus d'informations via les deux phrases encadrées en vert qui nous permettent de deviner que le problème est très probablement qu'on a essayé de faire une lecture à une adresse invalide et que cette adresse est invalide parce que le pointeur `NULL` est impliqué.

Les lignes suivantes indiquent les fonctions impliquées dans le problème, le fichier dans lequel chacune se trouve et pour chacune la ligne qui a causé le problème. L'ordre d'appel à ces fonctions se retrace en les lisant de bas en haut. Ici par exemple on retrace l'historique suivant de bas en haut :

1. Des trucs obscurs sur lesquels vous n'avez pas la main se sont passés.
2. Il y a eu un appel à la fonction `main` qui se trouve dans le fichier magasin.c (vous avez même accès au chemin absolu de ce fichier). En ligne 57, le `main` a fait un appel à ...
3. ... la fonction `libere` qui se trouve dans le fichier arbre.c. A priori, ça veut donc dire qu'on à écrit un module arbre auquel on fait appel dans le code source principal contenant le `main`. Vu les lignes suivantes cette fonction s'est appelée elle même récursivement et en ligne 70 a fait un appel à ...
4. ... la fonction `gauche` qui se trouve aussi dans arbre.c et dont la ligne 41 est problématique.

La conséquence de cette analyse c'est que les coupables présumés sont à chercher en ligne 57 du `main`, en ligne 70 de `libere` ou en ligne 41 de `gauche`. A priori, on commence par corriger les bases donc la première chose à faire est d'aller observer plus attentivement `gauche`.

Remarque : toutes ces informations sont en fait récapitulées à la fin du message d'erreur dans le résumé (SUMMARY), qui porte bien son nom et qui ici est plutôt explicite !

Double libération de mémoire

Observons à présent (les blocs grisés servent à préserver l'anonymat de la personne ayant écrit le code) :

```
=================================================================
==603374==ERROR: AddressSanitizer: attempting double-free on 0x602000004830 in thread T0:
    #0 0x7fb439b6b40f in __interceptor_free ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:122
    #1 0x55dc357b5be1 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15b
    #2 0x55dc357b5bd5 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15b
    #3 0x55dc357b5e1e in etats_accessibles (/home/aude/Documents/Administratif_classe/Notes/TPDM4 rendus/              /
    #4 0x55dc357b762b in main (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/              /TP4+0x1762b)
    #5 0x7fb438dd4082 in __libc_start_main ../csu/libc-start.c:308
    #6 0x55dc357af40d in _start (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/              /TP4+0xf40d)

0x602000004830 is located 0 bytes inside of 6-byte region [0x602000004830,0x602000004836)
freed by thread T0 here:
    #0 0x7fb439b6b40f in __interceptor_free ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:122
    #1 0x55dc357b5ca4 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15c
    #2 0x55dc357b5bd5 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15b
    #3 0x55dc357b5bd5 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/          _     /TP4+0x15b
    #4 0x55dc357b5e1e in etats_accessibles (/home/aude/Documents/Administratif_classe/Notes/TPDM4 rendus/              /
    #5 0x55dc357b762b in main (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/              /TP4+0x1762b)
    #6 0x7fb438dd4082 in __libc_start_main ../csu/libc-start.c:308

previously allocated by thread T0 here:
    #0 0x7fb439b6b808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55dc357b3c79 in delta_etats (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x
    #2 0x55dc357b5bb9 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15b
    #3 0x55dc357b5bd5 in parcours (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/                /TP4+0x15b
    #4 0x55dc357b5e1e in etats_accessibles (/home/aude/Documents/Administratif_classe/Notes/TPDM4 rendus/              /
    #5 0x55dc357b762b in main (/home/aude/Documents/Administratif_classe/Notes/TPDM4_rendus/              'TP4+0x1762b)
    #6 0x7fb438dd4082 in __libc_start_main ../csu/libc-start.c:308

SUMMARY: AddressSanitizer: double-free ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:122 in __interceptor_free
==603374==ABORTING
```

À nouveau la nature de l'erreur est indiquée non loin d'un mot "error" et est rappelée dans le résumé en fin de message : on a libéré plusieurs fois le même endroit de la mémoire sur le tas.

Comme dans l'exemple précédent on peut retracer l'historique du problème : le `main` a fait un appel à `etats_accessibles` qui a appelé `parcours` qui a appelé indirectement une fonction que vous n'avez pas implémentée personnellement et qui vu la nature de l'erreur est probablement un bout du code de `free`.

Ici, l'erreur provient d'une ligne (la 122) d'un code que vous n'avez pas écrit et bien sûr ce n'est pas lui qu'il faut modifier mais le votre : il n'utilise pas les fonctions natives correctement. La fonction sur laquelle se pencher a priori est donc `parcours`. Vous n'avez ici pas d'information précise sur la ligne à incriminer (et pas uniquement parce que j'ai tronqué, il se trouve que dans le message d'origine elles n'étaient que partiellement indiquées) mais vu la nature de l'erreur chercher les occurrences de `free` dans cette fonction devrait vous mettre sur la voie.

Remarque : dans ce cas particulier, les lignes qui suivent le premier bloc peuvent aider. En effet, les deux délimitations en rose vous indiquent par quelle fonction la mémoire que vous avez essayé de libérer deux fois a été allouée et par quelle fonction elle a déjà été libérée.

Une fuite mémoire

Un dernier exemple classique :

```
=================================================================
==613521==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 32 byte(s) in 1 object(s) allocated from:
    #0 0x7f4eb5996808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55a0d68d1832 in cons /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:25
    #2 0x55a0d68d2ae9 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:217
    #3 0x55a0d68d2c60 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:224
    #4 0x55a0d68d2d20 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:228
    #5 0x55a0d68d40f8 in main /home/aude/Documents/MPI/C_Ocaml/Tests/magasin.c:32
    #6 0x7f4eb4bff082 in __libc_start_main ../csu/libc-start.c:308

Direct leak of 32 byte(s) in 1 object(s) allocated from:
    #0 0x7f4eb5996808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55a0d68d1832 in cons /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:25
    #2 0x55a0d68d2ae9 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:217
    #3 0x55a0d68d2c60 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:224
    #4 0x55a0d68d2d20 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:228
    #5 0x55a0d68d4112 in main /home/aude/Documents/MPI/C_Ocaml/Tests/magasin.c:33
    #6 0x7f4eb4bff082 in __libc_start_main ../csu/libc-start.c:308

Direct leak of 32 byte(s) in 1 object(s) allocated from:
    #0 0x7f4eb5996808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55a0d68d1832 in cons /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:25
    #2 0x55a0d68d2ae9 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:217
    #3 0x55a0d68d2c60 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:224
    #4 0x55a0d68d2d20 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:228
    #5 0x55a0d68d2d20 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:228
    #6 0x55a0d68d4146 in main /home/aude/Documents/MPI/C_Ocaml/Tests/magasin.c:35
    #7 0x7f4eb4bff082 in __libc_start_main ../csu/libc-start.c:308

Direct leak of 32 byte(s) in 1 object(s) allocated from:
    #0 0x7f4eb5996808 in __interceptor_malloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55a0d68d1832 in cons /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:25
    #2 0x55a0d68d2ae9 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:217
    #3 0x55a0d68d2c60 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:224
    #4 0x55a0d68d2d20 in insere_2 /home/aude/Documents/MPI/C_Ocaml/Tests/arbre.c:228
    #5 0x55a0d68d40d8 in main /home/aude/Documents/MPI/C_Ocaml/Tests/magasin.c:31
    #6 0x7f4eb4bff082 in __libc_start_main ../csu/libc-start.c:308

SUMMARY: AddressSanitizer: 128 byte(s) leaked in 4 allocation(s).
```

Là encore la nature de l'erreur se trouve dès qu'on a trouvé le mot "error" ; on peut la deviner aussi grâce au résumé final même s'il est moins aidant que dans le troisième exemple : de la mémoire allouée sur le tas n'a pas été libérée. L'erreur est un peu plus longue mais on se rend vite compte que les 4 blocs sont très similaires et qu'en fait chacun retrace un chemin ayant mené à une partie des fuites mémoires constatées. On peut donc commencer par résoudre le problème soulevé par le premier bloc.

Pour cela, on procède comme dans les exemples précédents : la ligne 32 du `main` a lancé des appels à `insere_2` et indirectement à `cons` ce qui a alloué de la mémoire qui n'a pas été libérée. On commencera donc à observer plus en détail les fonctions `insere_2` et `cons` aux lignes indiqués par le message d'erreur (encadrées en rouge) pour débugguer.