

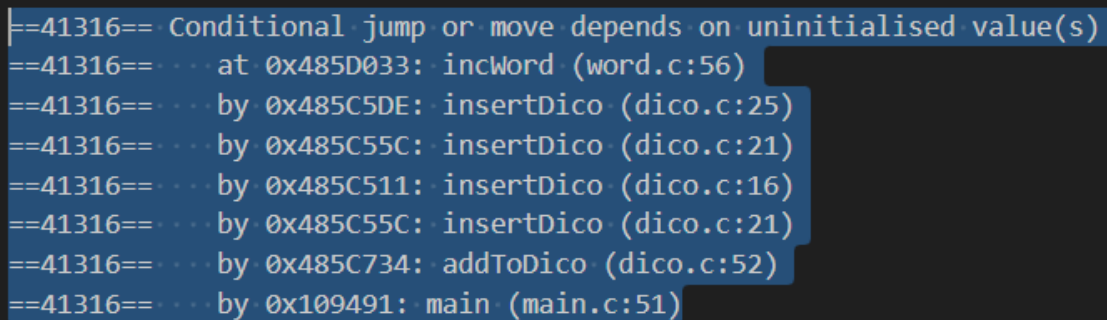
Rapport Projet: Gestion Mémoire

Le cours de cette semaine explore les principes fondamentaux de l'allocation et de l'utilisation de la mémoire. Il aborde les erreurs courantes liées à la mémoire, la stabilité, la sécurité des programmes et les outils permettant de les détecter et les corriger, tels que Valgrind et AddressSanitizer.

Notre travail porte alors sur la correction d'erreurs liées à la mémoire. Nous nous inspirons des outils utilisés en TP pour travailler sur notre projet.

Dans nos travaux pratiques, nous sommes arrivés à la conclusion que chaque outil a ses avantages et inconvénients. Ici, nous avons fait le choix d'utiliser Valgrind qui permet d'être plus précis sur quelles lignes du programme cause des erreurs de mémoire.

Après avoir lancé l'outil, nous nous sommes rendus compte qu'une grande partie des erreurs proviennent du fait que les variables ne sont pas initialisées:



```
==41316== Conditional jump or move depends on uninitialised value(s)
==41316==    at 0x485D033: incWord (word.c:56)
==41316==    by 0x485C5DE: insertDico (dico.c:25)
==41316==    by 0x485C55C: insertDico (dico.c:21)
==41316==    by 0x485C511: insertDico (dico.c:16)
==41316==    by 0x485C55C: insertDico (dico.c:21)
==41316==    by 0x485C734: addToDico (dico.c:52)
==41316==    by 0x109491: main (main.c:51)
```

Ceci est quelque chose qui est souvent survolé par les développeurs car la plupart des langages haut-niveau initialise les variables pour nous. Mais en C, une variable non initialisée n'est pas forcément égale à 0 ou à NULL ce qui peut provoquer des fuites de mémoires voire des comportements inattendus dans les différentes fonctions.

Le travail porte principalement sur l'élimination de ces problèmes en s'assurant d'initialiser toutes les variables correctement mais aussi de corriger toute mauvaise allocation.

En effet, dans le programme, plusieurs fois des pointeurs sont initialisés à l'aide d'allocation mémoire avant d'être assignés à un autre pointeur. C'est une pratique incorrecte car on alloue de la mémoire pour rien au final car le pointeur va désormais pointer vers une zone en mémoire déjà initialisée. Au final, on alloue de la mémoire que l'on perd à tout jamais et que le processus de notre programme empêche l'accès aux autres processus: On crée une fuite de mémoire.

```
void insertDico(dico** dictionary, mot_t* linkWord) {
    dico* newDictionary = (dico*) malloc(sizeof(dico));
    newDictionary = *dictionary;
```

MOCA - NTYAM Kévin, El Ouechrine

(Dangereux cette allocation pour changer la valeur du pointeur et perdre l'accès en mémoire. C'est une fuite de mémoire (i.e. memory leak))

La prochaine fois, nous travaillerons alors sur la mémoire et la performance du programme.