ZENG Fanxiang

FRANCOISE Raphaël

**TME 1 – Structure de données**

**Exercice 1 :**

**Partie 1**

* 1. Le programme est censé remplir le tableau en commençant par (longueur du tableau -1) jusqu'à 0. Il n’y a pas d’erreur à la compilation. Cependant nous rencontrons une *segmenation fault* à l’exécution du programme.
  2. Après l’itération où i vaut 0, i vaut 294967295. Nous nous attendions cependant à ce que i soit égale à -1. On essaie donc d’accéder à tab[294967295]. Cette *segmentation fault* s’explique par le fait que nous avons essayé d’accéder à une case qui n’existe pas dans le tableau.
  3. Pour résoudre cette erreur il suffit d’enlever le *unsigned* devant la variable i, pour qu’elle puisse devenir négative et que l’on puisse donc sortir de la boucle.

**Partie 2**

* 1. Le programme est censé créer une adresse et afficher les différentes composantes de la structure fraichement créer (nom, rue, code\_postal) à l’aide d’un *printf*. Lors de la compilation il n’y a pas de soucis. Cependant nous avons une *segmenation fault* à l’exécution du programme.
  2. *print new->rue* nous donne une adresse correspondant à 0x0.

Nous observons ligne 16 une *segmentation fault*. Cette erreur survient au lancement de la fonction *strcpy*.

La chaine *new->rue* n’a pas été alloué avant d’être appelée dans *strcpy.* Or cela doit être fait. Pour résoudre le problème, nous pouvons donc allouer la chaine new->rue de manière dynamique avec un *malloc(sizeof(char)\*n))*, *n* pouvant être choisi de façon arbitraire (ex : 200 pour être large).

**Partie 3**

* 1. Le programme est censé créer une structure de type *Tableau\** et initialiser le tableau de cette structure à 100 cases (de manière dynamique). Nous ajoutons ensuite des valeurs aux 5 premières cases. Puis nous affichons les n premières cases (de tab[0] à tab[(t->position)-1]) du tableau contenue dans cette structure.

La compilation et l’exécution on l’air de bien se passer. L’affichage attendu est celui obtenu.

* 1. Il y a un problème de fuite de mémoire.
  2. En exécutant le programme avec Valgrind nous remarquons une fuite de mémoire de 400 bytes. Cette fuite provient donc de *t->tab* car un *int* correspond à 4 bytes et ici la taille de *t->tab* est de 100 donc 400 bytes.
  3. Pour résoudre ce problème il faut libérer *t->tab* avec la commande *free(t->tab)* et ensuite libérer *t* avec *free(t).*

**Exercice 2 :**

**Partie 1**

2.1)Nous avons choisi le passage par référence car cela permet de modifier directement le tableau dans le *main* sans passer par une copie. Passer par une copie du tableau est beaucoup plus couteux et donc moins intéressant.

2.2) Dans la 2ème partie de la question il faut développer et réduire la somme que nous avons eu précédemment pour obtenir quelque chose de la forme (xi \* yi) et ensuite utiliser la formule donnée dans l’énoncé. Après avoir fait cela nous obtenons quelque chose du type : 2 \* ((n \* S(tab[i])) – (S(tab[i]))^2). Avec S qui correspond à la somme de i=1 à n, n étant la taille du tableau.

2.3) Nous pouvons voir que le temps mis par le CPU est inferieur pour la fonction de complexité O(n) que pour la fonction de complexité O(n^2). Le temps passe en générale de 3-4s à 1-2s avec la nouvelle version.

**Partie 2**

2.4) Voir code.

2.5.1) Voir code.

2.5.2) Voir code.

2.5.3)

2.6.1) Voir code.

2.6.2) Voir code.

2.6.3) Le premier algorithme est de complexité O(n^2) et le second de compléxité O(n^2).

2.6.4)