R 2.04

2022 - 2023

Communication et fonctionnement bas niveau

TD N°5

« Allocation dynamique de mémoire »

ANNE Jean-François
D'après le TD de F. BOURDON

Le but de ce TD est de se familiariser avec l'architecture bas niveau système et réseau.

« Allocation dynamique de mémoire »

Notions vues dans ce TD :

malloc(), calloc(), realloc(), free(), gprof.

Nombre de séance de 2h00 prévu pour faire ce TD : 1.

PS : Les parties correspondant à du travail à faire sont toutes en italiques ; le restant étant du complément au cours.

1°). Question n°1)

Ecrivez le prototype et le code correspondant aux fonctions « saisieXY() » et « milieu() ». La première fonction demande au clavier (utiliser « scanf ») deux réels. La deuxième fonction a besoin de deux couples (x, y) de réels et calcule le milieu du segment défini par ces points. Vous utiliserez le code retour de la fonction « milieu() » pour dire si le milieu a bien été calculé.

Ecrivez un programme principal qui utilise ces deux fonctions « saisieXY() » et « milieu() » pour calculer le milieu de segments de droites et affiche le résultat obtenu (coordonnées X/Y du point milieu).

Pour cela vous utiliserez la notion de structure pour manipuler la notion de "point". Un point P sera donc implanté avec une structure, elle-même composée de deux réels (ou entiers) x et y.

Pour faire cet exercice vous pouvez utiliser la méthode suivante :

1. Définir les interfaces des fonctions « saisieXY() » et « milieu() », en regardant quels sont les paramètres d'entrée et ceux d'entrée/sortie (pointeur).
2. Ecrire le programme principal (« main ») en incluant les appels aux fonctions définies en 1.
3. Regarder quelles sont les variables nécessaires pour ces appels et les réservations en mémoire qu'il faut faire.
4. Ecrire le contenu de ces fonctions.

2°). Question n°2)

Une fonction "C" ne peut retourner après son appel qu'un objet de type simple (entier, réel, caractère, ...) ou éventuellement un pointeur. Il est donc impossible de retourner un tableau.

Dans ce contexte comment utiliser le mot clé "return" dans la fonction "milieu" pour retourner le "point-milieu" (structure) calculé. L'idée serait de créer la structure "point-milieu" dans l'appel à la fonction, c'est-à-dire au plus proche de l'endroit où l'on s'en sert, et de renvoyer l'adresse de cette structure avec le "return". Le problème est que le programme appelant ("main") va récupérer une adresse mémoire sur la pile, qui sera libérée (sortie de la fonction). Il y aura un grand risque que la zone mémoire où se trouve cette structure soit utilisée pour d'autres besoins du processus, entraînant ainsi un problème d'accès concurrent à la mémoire.

Pour résoudre ce problème, regardez l'utilisation de la fonction « malloc( ) » qui permet d'allouer de la mémoire sur le tas (accessible par tous et pendant toute la durée de l'exécution du processus) et celle de la fonction « free( ) » qui libère cette zone de mémoire. En utilisant l'allocation dynamique de mémoire modifier le programme précédent (exercice n°1) pour renvoyer au programme appelant l'adresse de la structure, résultat du calcul dans la fonction « milieu() ».

3°). Question n°3)

Modifier le programme précédent en créant un fichier « donnees.txt » dans lequel vous placerez un premier entier qui indiquera le nombre de couples de points dont il faudra calculer le milieu de chacun d'eux. Vous placerez à la suite de ce premier nombre deux colonnes de valeurs, représentant les couples de points dont il faudra calculer les points milieu. Voici un exemple de fichier de données :

```
prompt> cat donnees.txt
5
24 56 66 74
44 63 45 98
17 16 38 83
88 69 32 81
3 98 55 18
prompt>
```

Votre programme devra appeler votre fonction « milieu() » dans une boucle afin de calculer le point milieu des différents points du fichier de données. Vous stockerez ces valeurs « milieu » dans un tableau construit dynamiquement. Avant de quitter le programme, vous afficherez les différents points milieu en face du couple de points correspondants. Etudiez l'optimisation de votre mémoire. Comparez l'usage d'une variable « static » dans la fonction « milieu() », avec celui d'une allocation dynamique dans cette même fonction « milieu() ».

4°). Question n°4)

Reprendre le code du programme « gest_fich.c » de la Question n°7) du TD9. Optimiser son code au niveau de la gestion de la mémoire en utilisant les fonctions « malloc() », « calloc() » et « free() ».

5°). Question n°5)

Ecrivez un programme qui alloue dynamiquement 100 Go (« malloc() ») par bloc de 1 Mo dans une boucle. Que se passe-t-il ? Qu'en concluez-vous ?

6°). Question n°6)

Reprenez le code de la Question n°5) et utilisez la fonction « memset() » dans une deuxième boucle pour initialiser la mémoire allouée avec le caractère « 1 ». Que se passe-t-il à l'exécution ? Expliquez. Vous pouvez lancer la commande « ps un » dans un autre terminal et observer, pour

le programme qui alloue la mémoire, les valeurs des colonnes « VSZ » et « RSS », qui correspondent respectivement à l'espace virtuel alloué et à l'espace physique réellement alloué en mémoire.

7°). Question n°7)

Remplacez dans le code du calcul de Fibonacci suivant, l'allocation statique du tableau des résultats, par une allocation dynamique de mémoire (« calloc() ») dans la fonction « calul_fibonacci() ».

```
#include <stdio.h>
#include <stdlib.h>
int calcul_fibonacci (int nombre_de_valeurs, int* tab)
{
int i;
if (nombre_de_valeurs > 0) {
tab[0] = 1;
if (nombre_de_valeurs > 1) {
tab[1] = 1;
for (i = 2; i < nombre_de_valeurs; i ++)
tab[i] = tab[i - 2] + tab[i - 1];
}
}
return 1;
}
int main (int argc, char * argv[])
{
int nb_valeurs;
int table[100];
int i, retour;
if ((argc != 2) || (sscanf(argv[1], "%d", & nb_valeurs) != 1)) {
fprintf(stderr, "Syntaxe : %s nombre_de_valeurs\n", argv[0]);
exit(EXIT_FAILURE);
}
retour = calcul_fibonacci(nb_valeurs, table);
if (retour == 1) {
for (i = 0; i < nb_valeurs; i ++)
fprintf(stdout, "%d\n", table[i]);
return EXIT_SUCCESS;
}
return -1;
}
```

8°). Question n°8)

Reprenez le code de la Question n°5) en utilisant la fonction « calloc() » au lieu de la fonction « malloc() ». Bien que cette fonction soit sensée initialiser à « 0 » la mémoire allouée dynamiquement, qu'observez-vous ?

9°). Question n°9)

A l'image de la Question n°6) vous pouvez compléter le code de la Question n°8) afin de mettre effectivement vous-même une valeur dans la mémoire allouée dynamiquement par « calloc() ». Cette fois qu'observez-vous ?

*Nous allons travailler sur la conjecture de Collatz, dont les calculs peuvent être faits de façon récursive. Cette conjecture (on connaît le résultat quels que soient les nombres utilisés, sans pouvoir le démontrer) travaille sur les nombres entiers positifs. Sa définition est assez simple. Il s'agit de faire un calcul récursif à partir de n'importe quel nombre entier positif. On applique récursivement à ce nombre « n » la fonction suivante : si « n » est pair, alors « n » devient « n/2 », sinon (n est impair) alors « n » devient « 3*n+1 ». Ce que l'on a observé sans jamais réussir à le démontrer, est qu'en appliquant cette fonction récursivement à n'importe quelle valeur entière strictement positive, on arrive toujours à « 1 ».*

10°). Question n°10)

Pour aborder cette question informatiquement nous allons construire un programme qui utilise un tableau à deux colonnes (matrice) qui sera alloué de façon statique (3000 lignes au départ) et comme une variable globale. Le programme principal (« main ») posera la question (« scanf() ») de la valeur de départ du calcul dans une boucle infinie, de façon à pouvoir enchaîner plusieurs calculs. On sortira de cette boucle infinie si la valeur initiale demandé vaut « -1 ». Pour chaque calcul, le programme principal appellera une fonction récursive (« fr() ») qui lancera les calculs sur le tableau global. La première colonne permettra de mémoriser la profondeur du calcul pour la valeur de départ (par exemple 3 pour une valeur initiale de 8, car 8 devient 4, puis 2, puis 1). La deuxième colonne mémorisera la valeur du nombre immédiatement suivant la valeur initiale dans le calcul (par exemple « 4 » pour « 8 » au départ). Dans la mesure où les différents calculs partagent le même tableau global, ce dernier se remplira au fur et à mesure des différents calculs. Pour pouvoir traiter de plus grands nombres, le tableau des résultats sera une matrice de « long int » et non de « int ».

Qu'observez-vous si vous lancer votre programme avec certaines valeurs initiales assez grandes ?

11°). Question n°11)

Pour corriger le problème de gestion de la mémoire rencontré à la question précédente, nous allons remplacer le tableau global des résultats par une allocation dynamique de mémoire au fur et à mesure des besoins du calcul. Le programme principal fera une première allocation dynamique avec un appel à la fonction « malloc() », dont il passera l'adresse à la fonction récursive. Cette dernière réallouera (fonction « realloc() ») cette zone dès que le calcul le demande, avant que le processus ne soit arrêté violemment pour débordement de mémoire. Vous pourrez tester votre programme avec les anciennes valeurs qui arrêtaient le programme de la Question n°10).

I. Webographie :

- https://bourdon.users.info.unicaen.fr/cours/IUT-1A/index.html
-