INFO0946 : Introduction à la Programmation
Challenge 5 (Pointeurs & Allocation Dynamique)

Benoit Donnet, Simon Liénardy

1 Énoncé des Problèmes

Une fois n'est pas coutume, il vous est demandé, dans ce dernier challenge, de résoudre deux problèmes. Le premier (Sec. 1.1) porte sur les pointeurs et l'évaluation d'expressions. Le seconds (Sec. 1.2) porte sur l'allocation dynamique de mémoire.

Les deux parties de ce challenge comptent chacune pour 50% de la note du challenge.

1.1 Pointeurs

Soit l'état de la mémoire suivant :

	220	17
	216	200
	212	72
	208	4
	204	16
`tab[0]` :	200	15
`data` :	16	73
	12	204
	8	??
`x` :	4	12

La première colonne donne le nom des variables, la deuxième donne des adresses mémoires (nous les exprimerons dans ce challenge en **base 10**) et, enfin, la troisième donne la valeur stockée à cette adresse (`??` signifie que la valeur est indéterminée). De plus, voici comment ont été déclarées les variables (on suppose que les entiers (`int`) sont représentés sur 4 octets [1]) :

```
1 int x, data, tab[] = {15,16,4,72,200,17}, *p = tab + 4;
```

Dans ce challenge, vous devez indiquer la valeur des **expressions** suivantes. Considérez qu'entre chaque expression, la mémoire est réinitialisée telle que présentée dans le schéma. Si un accès mémoire est demandé à une adresse hors de l'intervalle $[4, 220]$, mentionnez l'erreur de segmentation par la valeur `SF` [2]. Les adresses sont aussi représentées sur 4 octets. Ne tenez pas compte de l'avertissement éventuel que produirait un compilateur récent lors du *déréférencement* d'une valeur entière [3].

1. les `short` sur 2, les `long sur 8`
2. Pour `Segmentation Fault`.
3. Tant que l'adresse est dans $[4, 220]$, on considère que l'opération est permise.

1. `x`
2. `*x`
3. `**x`
4. `***x`
5. `&x`
6. `*&x`
7. `&*p`
8. `tab - x`
9. `tab - p`
10. `++*p`
11. `*p++`
12. `*++p`
13. `(*p)++`
14. `p + x`
15. `data + x`
16. `&x + data`
17. `(short *) p + data`
18. `*(tab[4] - *p)`
19. `p++ + ++x`
20. `*&data & x * data`

Lors de votre soumission, vous devrez indiquer les valeurs de ces expressions, en fonction de l'état de la mémoire décrit ci-dessus. La façon de formuler vos réponses est indiquée dans le fichier servant de canevas à votre soumission, fichier disponible sur la page web du cours. [4]

4. https://www.ecampus.ulg.ac.be, Sec. Challenges.

1.2 Allocation Dynamique

Soit le fichier header `chenil.h` qui représente, partiellement, un élevage de chiens :

Extrait de code 1 – Fichier `chenil.h`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// Défintion du type pour une date
typedef struct{
    unsigned short jour;
    unsigned short mois;
    unsigned short annee;
}Date;

// Définition d'un chien
typedef struct{
    char *nom;
    char sexe; //'F' ou 'M'
    char vaccine; //0 pas vacciné, 1 vacciné
}Chien;

// Définition d'une portée de chiens
typedef struct{
    Chien *pere;
    Chien *mere;
    Chien *chiots; //tableau de chiens
    unsigned short nb_chiots;
    Date date_naissance;
}Portee;

/*
 * Charge, depuis un fichier, {nb_chiots} chiots et les place dans un tableau.
 *
 * @pré: {nom_fichier} valide, {chiots} est un tableau de chiens valide
 * @post: 1 si {chiots} a bien été rempli depuis le fichier dont le nom est
 *             contenu dans {nom_fichier}.
 *       -1 sinon.
 */
int charger_chiots(char *nom_fichier, Chien *chiots, unsigned short nb_chiots);

/*
 * Crée un chien ayant un nom, un sexe et une information quant à sa
 * vaccination.
 *
 * @pré: {nom} valide, {sexe} ∈ {'M','F'}, vacciné ∈ [0,1]
 * @post: Un pointeur vers un chien valide.
 *        NULL en cas d'erreur.
 */
Chien *cree_chien(char *nom, char sexe, char vaccine);

/*
 * Crée une portée de chiots en fonction d'un père et d'une mère et charge,
 * depuis un fichier, les différents chiots.
 *
 * @pré: {nom_pere} est un nom de chien valide, {nom_mere} est un nom de chien
 *       valide. Le père et la mère ont été vaccinés, {nb_chiots} > 0,
 *       {nom_fichier} est valide.
 * @post: un pointeur vers une portée valide. Les chiots de la portée ont été
 *        chargés depuis le fichier {nom_fichier}.   NULL en cas d'erreur
 */
Portee *cree_portee(char *nom_pere, char *nom_mere, unsigned short nb_chiots,
                    char *nom_fichier, Date date_naissance);
```

Les spécifications pour chaque fonction vous sont données et doivent vous aider à comprendre comment utiliser une fonction/procédure et le retour attendu après exécution de la fonction/procédure.

Le fichier `chenil.c` est le module implémentant `chenil.h`. Il se présente comme suit :

Extrait de code 2 – Fichier `chenil.c`

```c
#include "chenil.h"

int charger_chiots(char *nom_fichier, Chien *chiots, unsigned short nb_chiots){

   //CODE NON DONNÉ

   return 1;
}

Chien *cree_chien(char *nom, char sexe, char vaccine){
   Chien *n_chien;

   //CODE NON DONNÉ

   return n_chien;
}

Portee *cree_portee(char *nom_pere, char *nom_mere, unsigned short nb_chiots,
                    char *nom_fichier, Date date_naissance){

   //votre code ici
}
```

Nous vous demandons d'implémenter le fonction `cree_portee()`. Dans le corps de la fonction, vous pouvez définir toutes les variables que vous jugez nécessaires. Soyez néanmoins précis sur les types de ces variables et n'utilisez pas le caractère underscore (« _ ») dans vos noms de variable. En outre, nous vous demandons, dans votre implémentation, d'appliquer les principes de la **programmation défensive**. Vous pouvez (comprendre : « devez ») utiliser les autres fonctions décrites dans `chenil.h`. En ce qui concerne les fonctions de `stdlib.h`, n'utilisez pas `calloc` et `realloc`.

Lors de votre soumission, vous devrez fournir votre code complétant la fonction indiquée. La façon d'écrire votre code est indiquée dans le fichier servant de canevas à votre soumission, fichier disponible sur la page web du cours. [5]

2 Tester votre solution

Comme annoncé plus haut, les codes de `charger_chiots` et `cree_chien` ne vous sont pas fournis. Pour tester votre solution, libre à vous d'écrire rapidement une implémentation simple pour ces fonctions. Par exemple, votre fonction `charger_chiots` n'a pas nécessairement besoin de manipuler un fichier et le contenu de ce fichier peut vous être inconnu. Faites simplement en sorte de respecter son interface, et surtout, testez tous les cas possibles (si la fonction renvoie 1, -1 ; si une précondition n'est pas remplie, etc.) [6]. N'oubliez pas de bien compiler chez vous votre code avec `-Wall`.

3 Agenda

Votre devoir doit être soumis pour le **vendredi 20/12, 18h00**, au plus tard. Pour rappel, vous disposez de maximum trois essais.

4 Soumettre une Solution

Ces consignes ne sont pas différentes de celles des devoirs précédents, hormis le nom du devoir qui change (`challenge5.txt`). Elles sont toutefois rappelées pour mémoire.

5. https://www.ecampus.ulg.ac.be, Sec. Challenges.
6. Par exemple, en fonction de ce vous pourriez entrer au clavier à ce moment-là. Le `scanf` sera donc dans **votre** code de `charger_chiots` que vous gardez **pour vous**. CAFÉ procède autrement pour les tests, ne perturbez pas son comportement en ajoutant des `printf` et des `scanf` dans le corps de `cree_portee`.

Pour tous les challenges, un fichier servant de canevas pour la soumission du challenge est disponible sur la page web du cours [7]. Le nom du fichier est `challengeX.txt` où `X` est remplacé par le numéro du challenge. Le squelette pour ce challenge 5 est donc contenu dans le fichier `challenge5.txt`. Par la suite, libre à vous de modifier le nom du fichier que vous soumettez, cela n'a pas d'importance.

Tous les challenges doivent être compressés en une archive « `.zip` ». Voici comment procéder sur les systèmes d'exploitation les plus courants.

Sous Windows Il suffit de cliquer sur le fichier à l'aide du bouton droit de la souris, sélectionner « Envoyer vers... » et sélectionner ensuite « Dossier compressé ».

Sous Linux (Ubuntu, Fedora, Linux Mint, ...) Il suffit de cliquer sur le fichier à l'aide du bouton droit de la souris, sélectionner « Compresser... ». Veillez bien à sélectionner « `.zip` » dans la liste des extensions possibles pour le fichier.

Sous OS X Cliquez sur le fichier en maintenant la touche Contrôle enfoncée (ou cliquez avec 2 doigts), sélectionnez « Compresser ».

Dans tous les cas Ne soumettez pas de fichier `.tar.gz`, `.7z`, `.rar` ou autre ! C'est bien un fichier `.zip` qui est attendu. Le nom de l'archive importe peu, tant que c'est une archive `zip` valide, dont le nom se termine bien par « `.zip` » **et ne comporte pas de caractères spéciaux comme des espaces, des parenthèses, etc.**

Si vous commettez un erreur dans la soumission, comme par exemple :
- Donner un mauvais nom à l'archive ;
- Soumettre deux fois d'affilée en cliquant trop rapidement ;
- Mal placer les réponses dans le fichier `challenge5.txt`
- ...

C'est dommage pour vous [8]. Redoublez d'attention la prochaine fois ! Pour autant, la plateforme de soumission et le soucis d'équité entre tous les étudiants ne permettent pas de vous octroyer une nouvelle soumission.

7. http://www.ecampus.ulg.ac.be, Sec. Challenges.
8. Nous partageons votre peine.