Pointeurs

ENSIIE FISA IAP 2019-2020

Si vous avez déjà fait les exercices suivants pendant le premier TP, vous pouvez les passer.

Exercice 1 — *Echange*

Dans un programme `echange.c`, créez une fonction qui prend en entrée 3 entiers a, b et c. À l'issue de la fonction,

— b a la valeur que a avait au début de la fonction

— c a la valeur que b avait au début de la fonction

— a a la valeur que c avait au début de la fonction

Exercice 2 — *Initialisation, Fin*

Dans un programme `initfin.c`, créez une fonction qui crée et renvoie un pointeur sur un entier égal à 0, puis une autre fonction qui prend un pointeur en entrée et libère la mémoire associée à ce pointeur.

Exercice 3 — *Min et max*

Dans un programme `minmax.c`, créez une fonction qui prend en entrée un entier n, un tableau `tab` contenant `n` entiers et 2 pointeurs sur entiers `min` et `max`. Cette fonction doit mettre à l'adresse pointée par `min` et `max` respectivement la plus petite et la plus grande valeur des entiers contenu dans `tab`. La fonction ne renvoie rien.

Exercice 4 — *Liste de log*

Dans un programme `loglist.c`, créez une fonction qui prend en entrée un entier n et crée et renvoie un pointeur p sur n flottant et initialise `p[i]` avec $\log(i)$.

Exercice 5 — *Pointeur sur pointeur*

Comment feriez vous pour représentez le type pointeur sur pointeur sur entier ?

Dans un programme `pointeursType.c`, créer une fonction qui prend en entrée un entier n et un pointeur l sur n entiers. Renvoyez en sortie un pointeur sur pointeur d'entiers p représentant deux listes d'entiers. La première liste contient le nombre d'entiers pairs pointés par l, suivi de la liste de ces entiers. La seconde liste contient le nombre d'entiers impairs pointés par l, suivi de la liste de ces entiers.

Exercice 6 — *Structure*

Dans un programme `date.c`, créez une structure `date`. Une date est définie par le jour, le mois et l'année.

Créez une fonction qui prend en entrée une date et l'affiche.

Créez une fonction qui prend en entrée un pointeur sur `struct date` et qui incrémente la date de un jour. Par soucis de simplicité, on ignorera les années bissextiles.

On utilisera la notation `->` : soit un type `struct s` possédant l'attribut `a`, si `x` est une variable de type `struct s`, on peut accéder à l'attribut `a` de `x` avec `x.a`. Soit maintenant un pointeur `p` sur `x`, alors on peut accéder à `x.a` avec `(*p).a`. Il existe une notation pour améliorer la lisibilité : `p->a`.

Exercice 7 — *Retour au tuto*

1. Le code n'est pas correct, pourquoi ?

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

int* adress(int x){
int* px = (int*) malloc(sizeof(int));
px = &x;
return px;
}

int main(void){
int x = 2;
int* px2 = adress(x);
printf("%p", px2);
free(px2);
}
```

2. Le code n'est pas correct, pourquoi ?

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

char* hello(){
char* px = "hello";
return px;
}

int main(void){
char* s = hello();
printf("%s", s);
free(s);
}
```

Exercice 8 — *Vecteurs*

1. (a) Ecrivez une structure capable de représenter un vecteur de $\mathbb{R}^n$, quel que soit n.
   (b) Ecrivez une fonction qui renvoie une vecteur de taille n initialisé à 0.
   (c) Ecrivez une fonction qui augmente la dimension du vecteur à une valeur n et le complète avec des 0.
   (d) Ecrivez une fonction qui modifie la i^e coordonnée d'un vecteur. Si i est plus grand que la dimension du vecteur, la fonction modifie le vecteur pour augmenter sa dimension.
   (e) Ecrivez une fonction qui affiche un vecteur.
   (f) Ecrivez une fonction qui renvoie la norme d'un vecteur.
   (g) Ecrivez une fonction qui normalise un vecteur.
   (h) Ecrivez une fonction qui copie un vecteur.
   (i) Ecrivez une fonction qui calcule la somme de deux vecteurs, s'ils ne font pas la même taille, la dimension du vecteur le plus petit est augmentée.
   (j) Ecrivez une fonction qui effectue le même calcul que la question précédente mais remplace le premier des deux vecteurs par le résultat.
   (k) Ecrivez une fonction qui calcule le produit d'un vecteur par un réel.

(l) Ecrivez une fonction qui effectue le même calcul que la question précédente mais remplace le vecteur par le résultat.

(m) Ecrivez une fonction qui calcule le produit scalaire de deux vecteurs, s'ils ne font pas la même taille, la dimension du vecteur le plus petit est augmentée.

(n) Ecrivez une fonction qui connaissant un vecteur de dimension 2 et un réel α, pivote ce vecteur d'un angle α.

2. On veut faire une application graphique très simple en utilisant les vecteurs de la question précédente.
   Notre application va afficher en console une tableau de $N \times N$ symboles `.` ou `X`, avec N impair. (Vous pouvez choisir N en fonction de la taille de votre console).
   Pour l'exemple, on suppose que $N = 11$. Au début, on voit en console

```
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>   machin:~$
```

   Une tortue se déplace sur la grille. Elle dispose d'un stylo qu'elle peut poser pour écrire ou lever pour ne pas écrire. La tortue se déplace ; si elle se déplace avec le stylo posé, elle écrit des `X` partout où elle passe.
   Au début la tortue a le stylo levé et est placé au centre de la grille. Si elle pose le stylo, on verra

```
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    .....X.....
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>   machin:~$
```

   Si la tortue se déplace ensuite 3 crans à droite, on verra

```
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>    .....XXXX..
>    ...........
>    ...........
>    ...........
>    ...........
>    ...........
>   machin:~$
```

Si la tortue lève le stylo, se déplace 3 cases en haut et baisse le stylo, on verra

```
>    ..........
>    ..........
>    ..........
>    ........x..
>    ..........
>    .....XXXX..
>    ..........
>    ..........
>    ..........
>    ..........
>    ..........
>    machin:~$
```

La tortue se déplace selon le système suivant :

- on peut tourner la tortue ; initialement elle regarde en haut
- on peut avancer la tortue selon une distance donnée, elle avance dans la direction vers laquelle elle est tournée.

Donc dans l'exemple ci dessus, les commandes à envoyer sont

- Tourner à 90 degrés à droite,
- Poser le stylo
- Avancer de 3
- Tourner à 90 degrés à gauche
- Lever le stylo
- Avancer de 2
- Tourner à 90 degrés à gauche
- Poser le stylo

(a) Ecrivez une structure représentant la tortue.

(b) Ecrivez un programme qui initialise un tableau `T` de `char` de taille `N x N` avec le symbole `.` partout.

(c) Ecrivez une fonction qui connaissant la tortue et le tableau `T`, pose le stylo de la tortue

(d) Ecrivez une fonction qui connaissant la tortue, lève le stylo de la tortue

(e) Ecrivez une fonction qui connaissant la tortue, le tableau `T` et une distance, déplace la tortue selon cette distance, dans la direction de la tortue. Si la position d'arrivée de la tortue n'est pas entière, arrondissez à l'entier le plus proche.

(f) Ecrivez une fonction qui connaissant la tortue et un angle α, tourne la tortue d'un angle α.

Exercice 9 — *TD Tableaux : Tableau dynamique*

1. (a) Dans un fichier `tdyn.c`, créez une structure `struct stdyn` contenant un pointeur sur entier `t` et deux entiers `size` et `max`. Le pointeur `t` représente un tableau d'entiers à taille variable. Les entiers `size` et `max` représentent respectivement la taille effective de `t` et le nombre d'entiers vers lequel `t` pointe (autrement dit la taille maximum de `t`).

   Dans le TD sur les tableaux, ce type représente les tableaux de l'exercice 2. `t` est le tableau t ; `size` est n et `max` est N.

   En utilisant `typedef`, définissez le type `tdyn` égal au type `struct stdyn`.

   (b) Ecrivez une fonction `init` qui renvoie un `tdyn` où `t` est un tableau vide pointant vers 1 entier.

   (c) Codez une fonction `insert` qui, connaissant un pointeur sur `tdyn td` et un entier `i` insère `i` dans `td`.

2. On souhaite faire une application graphique où il est possible d'annuler ou de refaire ses opérations.

   (a) Copiez `tdyn.c` dans un fichier `histo.c`.

   (b) Modifiez la structure `struct stdyn` pour que `t` soit un pointeur sur tableaux 2D de taille `10x10` entiers. Modifiez les fonctions `init` et `insert` en conséquence.

   (c) Ecrivez une fonction `print` qui, connaissant un `tdyn td` affiche le dernier tableau pointé par `td`; sauf si `td` est vide.

   (d) Ecrivez une fonction `edit` qui, connaissant un pointeur sur `tdyn td` et 3 entiers `i, j, k` insert dans `td` une copie du tableau `td.t` où l'entier aux coordonnées `i` et `j` est remplacé par k.

   (e) Ecrivez une fonction `ask` qui, connaissant un pointeur sur `tdyn td` demande à l'utilisateur 3 entiers et appelle la fonction `edit` avec `td` et ces 3 entiers.

   (f) On veut écrire un historique des appels à la fonction `edit` avec une fonction d'annulation `ctrlz` et une fonction d'annulation d'une annulation `ctrly`. Un appel à `ctrlz` annule le dernier appel a `edit`. Un second appel annule le deuxième dernier appel et ainsi de suite. Un appel à `ctrly` annule le dernier appel à `ctrlz`, comme s'il refaisait le dernier appel à `edit` qui a été annulé. Enfin, un appel à edit doit effacer tout le contenu de l'historique qui a été annulé.

   Un appel à `ctrlz` ou `ctrly` alors qu'il n'y a aucun `edit` ou aucune `annulation` à annuler est sans effet.

   Voici un exemple.

```
edit(td, 0, 0, 1); // 1 dans la case (0, 0)
edit(td, 0, 0, 2); // 2 dans la case (0, 0)
edit(td, 0, 0, 3); // 3 dans la case (0, 0)
edit(td, 0, 0, 4); // 4 dans la case (0, 0)
edit(td, 0, 0, 5); // 5 dans la case (0, 0)
edit(td, 0, 0, 6); // 6 dans la case (0, 0)
ctrlz(td);  // 5 dans la case (0, 0)
ctrlz(td);  // 4 dans la case (0, 0)
ctrlz(td);  // 3 dans la case (0, 0)
edit(td, 0, 0, 8); // 8 dans la case (0, 0)
ctrlz(td);  // 3 dans la case (0, 0)
ctrlz(td);  // 2 dans la case (0, 0)
ctrlz(td);  // 1 dans la case (0, 0)
ctrlz(td);  // Sans effet
ctrly(td);  // 1 dans la case (0, 0)
ctrly(td);  // 2 dans la case (0, 0)
ctrly(td);  // 3 dans la case (0, 0)
ctrlz(td);  // 2 dans la case (0, 0)
ctrlz(td);  // 1 dans la case (0, 0)
ctrly(td);  // 2 dans la case (0, 0)
ctrly(td);  // 3 dans la case (0, 0)
ctrly(td);  // 8 dans la case (0, 0)
ctrly(td);  // Sans effet

```

   Implantez les fonctions `ctrlz`, `ctrly` et `edit` en conséquence. Vous pouvez modifier la structure `tdyn` si cela vous aide.

   (g) Ecrivez enfin une fonction qui demande en boucle à l'utilisateur s'il souhaite insérer, annuler sa dernière insertion ou refaire sa dernière insertion annulée et qui agit en conséquence. La fonction affichera le tableau après chaque action.