TP3 : Pointeurs, tableaux, chaînes de caractères

MP2I Lycée Pierre de Fermat

Consignes

Vous déposerez votre rendu sur cahier de prépa, avant ***mardi 11/10 à 22h00***.

Vous devez commenter ***toutes*** vos fonctions, et mettre des assertions lorsque c'est nécessaire.

Votre archive de rendu doit contenir un fichier de réponse, et doit aussi contenir un sous-dossier par exercice. Chaque sous-dossier contiendra le ou les fichiers C demandés dans l'énoncé de l'exercice. Vous devez rendre uniquement le code et supprimer les fichiers exécutables.

Votre fichier de réponse doit être un fichier ***".txt"***. Si vous êtes sur Mac, veillez à bien me rendre un fichier texte simple et pas un fichier .docx car je ne peux pas les lire. Cherchez sur internet comment faire des fichiers .txt sur Mac si vous ne savez pas le faire.

Vous êtes encouragés à travailler à plusieurs, mais veillez à ***ne pas rendre du code copié*** sur vos camarades. Si vous n'arrivez pas à finir un exercice, passez à la suite et notez le dans vos réponses.

*
* *

Vous pouvez télécharger sur cahier de prépa une archive contenant des fichiers C pour ce TP, afin de ne pas avoir à recopier les codes donnés dans le sujet.

Pointeurs

Un pointeur est une adresse mémoire, et prend des valeurs entre 0 et $2^{64} - 1$. En C, la valeur NULL représente le ***pointeur nul***. Cette valeur particulière est utilisée pour encoder "un pointeur vers rien". Par exemple, certaines fonctions de la librairie standard C renvoie un pointeur, si elles renvoient NULL c'est qu'il y a eu une erreur. En pratique, NULL vaut 0 sur presque toutes les machines.

Lorsque l'on essaie d'accéder à une zone de la mémoire à laquelle on n'a pas le droit, par exemple parce qu'elle ne correspond pas à une variable déclarée, le programme s'arrête instantanément et affiche une ***segmentation fault***, ou ***erreur de segmentation***. Vous risquez de voir cette erreur très souvent au cours de ce TP. Il est important de segmenter vos programmes en ***fonctions*** et de munir ces fonctions d'***assertions*** car cela vous aidera à mieux localiser les bugs.

Exercice 1.

Essayez de compiler et d'exécuter le code suivant. Que se passe-t'il ? Pourquoi ?

```c
#include <stdio.h>

int main()
{
  int* p = NULL;
  printf("La valeur pointée par p est %d\n", *p);

    return 0;
}
```

Dans la suite, lorsqu'une fonction prend en argument un pointeur, et qu'elle tente de déréférencer ce pointeur, on doit s'assurer au début de la fonction que le pointeur n'est pas nul, avec une assertion.

Exercice 2.

Question 1. Ouvrez le fichier exo_pointeurs.c présent dans l'archive. Sans le compiler, juste en le lisant et en l'exécutant à la main, qu'affiche ce programme ? *Vous pourrez simuler l'exécution en représentant graphiquement la mémoire par un grand tableau, et les pointeurs par des flèches entre les cases, pour vous aider.*

Question 2. Exécutez le code et vérifiez qu'il affiche bien ce que vous avez prévu.

Question 3. Que serait un nom adéquat pour la fonction f ? Rédigez un commentaire pour cette fonction, et rajoutez des assertions si nécessaire.

Les pointeurs peuvent permettre aux fonctions de renvoyer plusieurs valeurs. L'idée est de définir la fonction avec des paramètres supplémentaires `t_1* res_1`, `t_2* res_2`, ..., `t_k* res_k`, et de stocker dans chaque `res_i` une valeur.

Question 4. Écrivez une fonction permettant de récupérer la somme, le produit et le maximum de deux entiers, et testez la sur plusieurs valeurs.

Tableaux

En C, on peut déclarer un tableau de la manière suivante :

```c
int tab[50];
```

Cette ligne de code a pour effet de réserver 50 cases mémoires de 4 octets, contigües dans la mémoire. La variable tab a le type `int*`. Lorsque l'on crée une fonction prenant un tableau comme argument, on peut soit utiliser le type `t*`, où `t` est le type des éléments du tableau, soit utiliser la syntaxe suivante :

```c
int ma_fonction(int tab[]);
```

On peut accéder au 1er élément d'un tableau tab avec *tab, au 2e élément avec *(tab + 1), etc... jusqu'au 50e élément avec *(tab + 1). On n'utilise que rarement cette syntaxe en pratique, et on écrira plutôt tab[0], tab[1], ..., tab[49].

On peut initialiser un tableau directement avec des valeurs :

```
1 int tab[3] = {12, 63, 268};
```

En C, il n'y a aucun moyen d'obtenir la taille d'un tableau. Ainsi, lorsqu'une fonction manipule un tableau, on doit fournir la taille du tableau en paramètre. De plus, on ne peut pour l'instant créer que des tableaux de taille constante (i.e. écrire int tab[50];) et pas de taille variable (i.e. écrire int tab[n];.

Pour l'instant, on ne peut pas faire de fonction qui ***renvoie*** un tableau : essayez, vous verrez que le compilateur soulève un avertissement.

Préprocesseur, macro #define Dans vos programmes, vous serez amenés à manipuler des tableaux d'une certaine taille. Par exemple, si vous manipulez un tableau de taille 100, vous devrez écrire 100 à de nombreux endroits du programme. Pour changer 100 en 1000, il faudra changer CHAQUE instance. Afin d'éviter cela, on utilise la directive ***#define*** qui n'est pas une instruction, mais permet de donner un nom à un bout de code. Par exemple, si au début de votre programme, vous écrivez :

```
1 #define N 1000
```

Avec cette ligne, dans le reste du programme, à chaque fois que vous écrirez N, le compilateur verra les caractères "1000". Attention N n'est pas une variable ! C'est un nom que vous donnez à une suite de caractères. Par exemple, vous pouvez écrire int tab[N]; sans problème.

Lors de la compilation, une première phase consiste à remplacer toutes les macros créées par #define par leur valeur. Pour cette raison, #define est appelée une ***directive préprocesseur***. Toutes les directives préprocesseur commencent par # (par exemple, les inclusions de librairies).

Exercice 3.

Dans l'archive fournie avec le TP, vous trouverez le fichier ***eratosthene.c*** implémentant l'algorithme du ***crible d'Ératosthène***. Lisez et exécutez le code pour comprendre comment il marche, et comment on utilise les tableaux et les macros.

Notez que dans ce code, on utilise un tableau de taille 10^6, mais que l'on utilise que les n premières valeurs, où n est rentré par l'utilisateur. On simule donc un tableau de taille n avec un tableau de taille 10^6 lorsque $n < 10^6$. Pour l'instant cette méthode nous permettra d'avoir des programmes interactifs, où l'utilisateur peut choisir la taille des tableaux, mais c'est très peu efficace car on réserve beaucoup de mémoire même pour $n = 10$. On verra au cours suivant une manière plus adaptée de réserver de la mémoire dans ce type de situation.

Exercice 4.

Question 1. Écrivez un programme qui crée un tableau de taille 6, contenant les flottants suivants $5, -3.6, 28, 9.2, 17, 0.02$.

Question 2. Modifiez votre programme pour qu'il affiche les éléments du tableau, un élément par ligne.

Question 3. Écrivez un autre programme, qui crée un tableau de taille $n <= 100000$, avec n rentré par l'utilisateur, où l'élément d'indice i est i^2.

Les tableaux étant des pointeurs, une fonction peut modifier le contenu d'un tableau qu'on lui passe en argument.

Exercice 5.

Question 1. Écrivez une fonction prenant en entrée un tableau d'entiers t de taille n, mettant chaque élément du tableau au carré. N'oubliez pas que la taille du tableau doit être un paramètre de la fonction !

Question 2. Écrivez une fonction prenant en entrée un tableau d'entiers t de taille n, et deux entiers a et b, et initialisant chaque élément de t par un entier aléatoire entre a et b. Gardez cette fonction sous le coude pour tester de futurs programmes.

Chaîne de caractère

Le type char sert en réalité à représenter non seulement des entiers signés 8 bits, mais aussi des caractères (comme son nom l'indique), en associant à chaque symbole u n nombre entier entre 0 et 127. Par exemple :

— Les lettres majuscules 'A' - 'Z' sont représentées par les entiers 65 - 90 (soit 0x41 - 0x5a)

— Les lettres minuscules 'a' - 'z' sont représentées par les entiers 97 - 122 (soit 0x61 - 0x7a)

— Les chiffres '0' - '9' sont représentés par les entiers 48 - 57 (soit 0x30 - 0x39)

Ainsi, en C, lorsque l'on écrit 'a', c'est ***exactement*** comme si l'on écrivait 97 ou 0x41. Attention, il faut utiliser des guillemets ***simples*** pour les caractères, et des guillemets ***doubles*** pour les chaînes de caractères. En C, 'a' et "a" ne sont pas équivalents !

Exercice 6.

Écrivez un programme qui demande un entier à l'utilisateur, et affiche le caractère correspondant. Vérifiez que vous retrouvez bien les valeurs annoncées au dessus. En C, le format %c permet d'afficher les caractères.

Un texte est donc représenté par une ***chaîne de caractères***, c'est à dire un tableau de caractères. Toutes les chaînes de caractère finissent par le caractère 0x00, appelé caractère nul. En C, on peut le noter '\0'. Ainsi, contrairement au cas général des tableaux, il est possible de connaître la longueur d'une chaine de caractères, en la lisant jusqu'à rencontrer le caractère nul. En revanche, la longueur d'une chaîne de caractères n'est pas nécessairement égale à la taille mémoire qu'on lui a réservé. Si l'on déclare une chaîne de 500 caractères, et que l'on écrit '\0' dans la case 3, la longueur de la chaîne est 3, alors que la taille mémoire réservée est 500.

En C, le format "%s" sert à afficher une chaîne de caractères, et a pour effet d'afficher tous les caractères dans l'ordre jusqu'à rencontrer un '\0'. Si la chaîne de caractère ne possède pas de caractère nul (par exemple parce qu'on l'a supprimé à la main), alors on va dépasser la mémoire de la chaîne de caractère et lire dans la mémoire de l'ordinateur jusqu'à trouver un octet valant 0.

Lorsque l'on utilise le format "%s" avec la fonction scanf, le programme va lire les caractères dans le terminal jusqu'à rencontrer un saut de ligne ou un espace.

Exercice 7.

Question 1. Le programme scan_string.c dans l'archive du TP demande une entrée à l'utilisateur, et écrit dans le terminal ce que l'utilisateur a entré. Lisez le code, exécutez-le, et testez d'écrire des mots, des phrases, etc... pour comprendre comment le programme fonctionne. Essayez d'écrire un mot de plus de 20 lettres : que se passe-t'il ? Expliquez en quoi ce comportement peut être exploité pour faire dysfonctionner un programme.

```c
#include <stdio.h>

/* Répète ce que l'utilisateur écrit tant que le programme
 * n'est pas interrompu*/
int main()
{
  char buf[20]; // sert à stocker la chaîne lue
  while (1){
    scanf("%s", buf);
    printf("Vous avez écrit: %s\n", buf);
  }
  return 0;
}
```

Question 2. Cherchez sur Internet la table ASCII, table de correspondance entre les entiers 0-127 et les caractères. Quels entiers correspondent aux caractères suivants ? Donnez vos réponses en décimal, binaire et hexadécimal.

1. `'?'`
2. `'.'`
3. `'\n'` (le caractère de retour à la ligne, aussi appelé ***Line Feed*** ou ***LF***)

Question 3. Le code suivant est dans le fichier string.c. Lisez-le et, sans l'exécuter, écrivez ce qu'il devrait afficher. Exécutez le programme pour vérifier.

```c
#include <stdio.h>

int main(){
  char str[20];
  str[4] = 'T';
  str[1] = 'A';
  str[3] = 'L';
  str[0] = 'S';
  str[2] = 'U';
  str[5] = '\0';
  str[6] = '!';
  str[7] = '!';

  printf("%s\n", str);
  return 0;
}
```

Question 4. Compilez et exécutez le fichier coucou_bonjour.c dans l'archive fournie avec le TP. Décrivez son comportement, et formulez une hypothèse sur pourquoi il se comporte de la sorte :

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
  char str1[8] = "BONJOUR"; // 7 lettres + 1 caractère nul = 8
  char str2[7] = "COUCOU"; // 6 lettres + 1 caractère nul = 7
  str1[7] = ' ';
  str2[6] = ' ';

  printf("%s\n", str1);
  printf("%s\n", str2);

  return 0;
}
```

Exercice 8.

Créez un unique fichier C pour cet exercice. Nous allons recoder quelques fonctions de la bibliothèque standard de C sur les chaines de caractères, <strings.h>. En conséquence, si vous savez déjà utiliser cette librairie, vous n'avez pas le droit de l'utiliser ici !

Pour chaque fonction, rajoutez dans le main plusieurs tests, en ajoutant de l'affichage permettant de vérifier que les tests fonctionnent correctement.

Question 1. Écrivez une fonction `int strlen(char* str)` qui calcule la longueur d'une chaîne de caractères. Par exemple, `strlen("bonjour")` renvoie 7, `strlen("")` renvoie 0, et `strlen("bonjour\0!!")` renvoie 7.

Question 2. Écrivez une fonction `void epeler(char* str)` qui affiche chaque caractère de `str` sur une ligne distincte. Vous n'avez pas le droit d'utiliser la fonction longueur !

Question 3. Écrivez une fonction `void strcpy(char* dst, char* src)` qui copie la chaîne de caractères source, src, et la recopie dans la chaîne destination, dst. On suppose que la taille du tableau dst est assez grande pour contenir la chaîne contenue dans le tableau src.

Question 4. Écrivez une fonction `void strcat(char* dst, char* src)`, qui concatène la chaîne src à la fin de la chaîne dst. La concaténation est le fait de coller deux chaînes ensembles. Par exemple :

```c
char str1[50] = "Bonjour";
char str2[20] = " tout le monde !";
strcat(str1, str2);
```

Après l'exécution de ce code, str1 contiendra "Bonjour tout le monde !" (suivi du caractère nul pour terminer la chaîne).

Question 5. Écrivez une fonction `int strcat(char* str1, char* str2)` qui compare un à un les caractères de `str1` et de `str2`, et renvoie :

- 0 si les deux chaînes sont égales
- 1 si le premier indice i où les chaînes contiennent un caractère différent est tel que $str1[i] > str2[i]$
- -1 si le premier indice i où les chaînes contiennent un caractère différent est tel que $str1[i] < str2[i]$

Cette fonction doit pouvoir traiter des chaînes de longueur différentes.

Tableau 2D

On peut créer des tableaux de tout type. En particulier, on peut créer des tableaux de pointeurs, et donc des tableaux de tableaux. On peut donc représenter ainsi des matrices, c'est à dire des grilles de nombres. Par exemple pour déclarer un tableau 2D avec 4 lignes et 3 colonnes, on écrit :

```
1 int tab[4][3];
```

Ainsi, `tab` est un pointeur vers une suite de 4 pointeurs :

`tab[0]`, `tab[1]`, `tab[2]` et `tab[3]`.

Chaque `tab[i]` est un pointeur vers une liste de 3 entiers :

`tab[i][0]`, `tab[i][1]`, `tab[i][2]`

Visuellement, on représente les tableaux 2D de la manière suivante :

tab[0][0]	tab[0][1]	...	tab[0][1]	...	tab[0][m−1]
tab[1][0]	tab[1][1]	...	tab[1][j]	...	tab[1][m−1]
⋮	⋮	⋱	⋮		⋮
tab[i][0]	tab[i][1]	...	tab[i][j]	...	tab[i][m−1]
⋮	⋮		⋮	⋱	⋮
tab[n−1][0]	tab[n−1][1]	...	tab[n−1][j]	...	tab[n−1][m−1]

On peut généraliser toutes ces idées à 3, 4, ... dimensions.

Lorsque l'on déclare une fonction prenant en paramètre un tableau 2D de taille $N \times M$, on écrit :

```
1 void f(int tab[N][M]){
2   ...
3 }
```

Si l'on utilise un tableau 2D de taille $N \times M$ mais que l'on utilise que les n premières lignes et les m premières colonnes, il faut encore une fois fournir n et m aux fonctions :

```
1 void f(int tab[N][M], int n, int m){
2   ...
3 }
```

Exercice 9.

Question 1. Codez une fonction qui prend en entrée un tableau 2D de taille $n \times m$ (n'oubliez pas de fournir n et m en argument !) et qui remplit le tableau en donnant à la case d'indice (i, j) la valeur $i + j$.

Question 2. Dans le même fichier, codez une fonction qui affiche un tableau 2D, en séparant d'un espace les éléments d'une même ligne. Testez la sur le tableau généré à la question précédente.

Question 3. Écrivez un nouveau programme, et écrivez une fonction qui, étant donné un tableau $N \times N$ (avec N une macro définie au début du programme), le remplit en le numérotant dans un ordre de spirale. Par exemple pour $N = 6$ le tableau devra contenir :

0	1	2	3	4	5
19	20	21	22	23	6
18	31	32	33	24	7
17	30	35	34	25	8
16	29	28	27	26	9
15	14	13	12	11	10

Indication : Pour vous aider, vous pouvez écrire une fonction :

```
int anneau(int** tab, int i0, int j0, int valeur_init, int taille)
```

Cette fonction fait un tour de spirale, en écrivant dans `tab` les nombres de `valeur_init` à `valeur_init` + taille(taille − 1) − 1 dans un anneau carré de côté `taille` dont le coin gauche est en (i_0, j_0). Enfin, la fonction renvoie la prochaine valeur à écrire.

Par exemple, si `tab` est un tableau de taille 6×6 rempli de zéros, alors après avoir appelé `anneau(tab, 1, 1, 20, 4)`, `tab` contient :

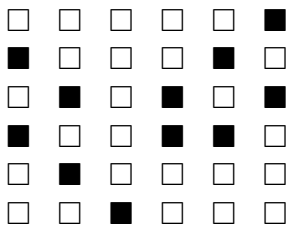
0	0	0	0	0	0
0	20	21	22	23	0
0	31	0	0	24	0
0	30	0	0	25	0
0	29	28	27	26	0
0	0	0	0	0	0

et l'appel de fonction a renvoyé 32.

Question 4. Écrivez une nouvelle fonction dans le même programme, remplissant un tableau en spirale en partant du milieu (vous pouvez utiliser la fonction précédente). La numérotation devra commencer à 1. Par exemple, pour $N = 6$, le tableau devra contenir :

36	35	34	33	32	31
17	16	15	14	13	30
18	5	4	3	12	29
19	6	1	2	11	28
20	7	8	9	10	27
21	22	23	24	25	26

Utilisons les différentes fonctions vues jusqu'ici pour écrire un programme dessinant une ***spirale d'Ulam*** de côté N. La spirale d'Ulam est obtenue en considérant une spirale comme obtenue à la question d'avant, et en coloriant les cases correspondant à des nombres premiers en noir, et les autres en blanc. Par exemple pour $N = 6$, la spirale d'Ulam est :



Question 5. Écrivez un nouveau programme qui affiche la spirale d'Ulam de côté N, avec N une macro définie dans le programme.

En C, pour tenter de rendre l'affichage joli, on affichera "X " (X puis un espace) pour un nombre premier et " " (deux espaces) pour les autres nombres.

Vous pouvez réutiliser le code du crible d'Ératosthène ainsi que les fonctions que vous avez programmées plus haut.

Exercice 10.

"Cent mille milliards de poèmes" est un ouvrage / une expérience de Raymond Queneau, un écrivain surréaliste du XX^{e} siècle membre de l'Oulipo (un groupe de recherche sur la création littéraire). L'ouvrage est un sonnet (14 vers), où chaque vers possède 10 versions différentes interchangeables. Ainsi, il y a 10^{14} combinaisons possibles, c'est à dire cent mille milliards.

Le fichier queneau.c contient un tableau char$*$ vers [14][10] avec les 10 versions de chacun des 14 vers. Complétez ce fichier pour écrire un programme affichant un des 10^{14} poèmes au hasard.

Exercice libre

Exercice 11.

Écrivez un programme utilisant les différentes notions vues dans ce TP. Vous devez avoir plusieurs fonctions et votre programme doit être adapté à un utilisateur, c'est à dire qu'il doit afficher des messages informatifs (par exemple quand il demande une entrée, qu'il affiche un résultat, etc...). Le programme doit évidemment être commenté de manière claire, et n'oubliez pas les assertions lorsque c'est nécessaire.

Si vous n'avez pas d'idées, vous pouvez piocher dans la liste suivante :

— Un programme qui génère des mots aléatoires de n lettres, ou de n syllabes

— Une implémentation du jeu de la vie de Conway (voir sur Wikipédia si vous ne connaissez pas)

— Un jeu de labyrinthe