TD XIV

POINTEURS

1 Manipulation de pointeurs

Exercice XIV.1

On considère le code suivant :

```c
#include <stdio.h>

int n = 3;
int p;

int f(int n){
    int* p = &n;
    int x = n + *p;
    return x + 1;
}

int g(int x, int y){
    int z = f(x);
    return z + f(y);
}

int main(void){
    p = 4;
    int result = g(n, p);
    printf("result = %d\n", result);
    return 0;
}
```

Aller sur le site https://pythontutor.com/c.html (vous pouvez taper *C tutor* sur Google, ça devrait être le premier résultat) et visualiser l'exécution pas à pas du programme. Essayer de bien comprendre ce qui se passe.

Exercice XIV.2

Dans chacun des cas suivants :

- déterminer si le programme est « faux » (lecture d'une variable non initialisée, déréférencement d'un pointeur invalide, erreur de type...) ;
- écrire sur papier (sans exécuter le programme) ce qu'on obtiendrait sur *C Tutor* (évolution du schéma mémoire et affichage produit) ;
- copier le code sur *C Tutor* et vérifier.

1.

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

void print_bool(bool b){
    if (b){
        printf("true\n");
    } else {
        printf("false\n");
    }
}

int main(void){
    double pi = 3.14;
    double e;
    double* p = NULL;
    p = &e;
    *p = pi;
    print_bool(e == pi);
    pi = 4.5;
    print_bool(e == pi);
}
```

2.

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

int x = 7;
int y = 12;
int* p;

int f(int x){
    printf("x = %d\n", x);
    printf("y = %d\n", y);
    printf("*p = %d\n", *p);
    int y = 1;
    printf("y = %d\n", y);
    x = x + y;
    return x;

}

int main(void){
    int z;
    p = &x;
    z = f(x + 1);
    printf("z = %d\n", z);
    return 0;
}
```

Exercice XIV.3

Écrire une fonction de prototype :

```
void extrema(int t[], int taille, int* min, int* max)
```

Les préconditions sont :

- la longueur de `t` vaut `taille`, et elle est strictement positive ;
- `min` et `max` sont des pointeurs valides.

La fonction affectera le minimum de `t` à l'objet pointé par `min`, et le maximum à celui pointé par `max`.

Exercice XIV.4

On considère la fonction suivante :

```
void mystere(int* x, int* y){
    *x = *x - *y;
    *y = *x + *y;
    *x = *y - *x;
}
```

1. Quel affichage obtiendrait-on avec le code suivant ?

```
int x = 3;
int y = 4;
mystere(&x, &y);
printf("x = %d\n", x);
printf("y = %d\n", y);
```

2. De manière générale, quel est l'effet de la fonction `mystere` ? On justifiera.
3. Quel affichage obtiendrait-on avec le code suivant ?

```
int x2 = 3;
int y2 = 3;
mystere(&x2, &y2);
printf("x2 = %d\n", x2);
printf("y2 = %d\n", y2);
```

4. Quel affichage obtiendrait-on avec le code suivant ?

```
int x3 = 3;
mystere(&x3, &x3);
printf("x3 = %d\n", x3);
```

Quel est le problème dans la démonstration faite plus haut ?

2 Fonction `scanf`

La fonction (ou *les fonctions* en fait, il en existe toute une série) `printf` permet de *lire* des données sous un format spécifié, tout comme la fonction `printf` permet d'*écrire* des données formatées.

Le principe de base est assez simple : on passe comme premier argument une chaîne de format, contenant un certain nombre de « champs » (`%d`, `%f` et autres), et ensuite une série de pointeurs. Il faut un pointeur par champ, et il faut que les types des pointeurs correspondent aux types des champs, dans

l'ordre d'apparition dans la chaîne de format.

Exercice XIV.5

Le programme suivant permet de lire sur l'entrée standard :

- un entier n ;
- puis ensuite, n fois, un entier p et un flottant x.

Après chaque lecture de (p, x), le programme affiche la somme $p + x$ sur la sortie standard.

```c
#include <stdio.h>

int main(void){
    int n = 0;
    float x = 0.;
    int p = 0;
    scanf("%d", &n);
    for (int i = 0; i < n; i++){
        scanf("%d %f", &p, &x);
        printf("%f\n", p + x);
    }
    return 0;
}
```

1. Compiler et exécuter ce programme.
2. Essayer quelques variations, en particulier sur le nombre d'espaces entre les différentes entrées, les retours à la ligne...
3. Essayer également d'exécuter ce programme en redirigeant sa sortie standard vers un fichier (`./programme >sortie.txt`). Cela permet de mieux faire la différence entre les entrées et les sorties.
4. Il est aussi possible de *rediriger l'entrée standard* d'un programme. Cela signifie que les valeurs seront lues dans le fichier spécifié au lieu de devoir être rentrées au clavier en direct. Créer un fichier `entree.txt` contenant les lignes suivantes :

   ```
   3
   1 3.14


   7 1 2 1.4
   ```

   Exécuter ensuite la ligne de commande suivante, et observer la sortie obtenue.

   ```
   $ ./programme <entree.txt
   ```

5. Rien n'empêche de rediriger l'entrée et la sortie (c'est même assez courant). Tester par exemple la commande suivante :

   ```
   $ ./programme <entree.txt >sortie.txt
   ```

Il y a un certain nombre de subtilités, qu'il n'est pas du tout indispensable de maîtriser :

- les champs numériques (comme `%d` qui lit un `int`, `%f` qui lit un `float`, `%lf` qui lit un `double`...) « mangent » les caractères d'espacement (espaces, retours à la ligne...). Par exemple, si l'entrée est `" \n\n 1234 \n3.14"` et que deux variables `int x` et `float y` ont été préalablement définies :
  - après un appel `scanf("%d", &x)`, `x` vaut `1234` et l'entrée restante est `" \n3.14"` ;
  - si l'on fait ensuite un appel `scanf("%f", &y)`, `y` vaut `3.14` et l'entrée a été entièrement consommée ;

  - on aurait pu remplacer ces deux appels successifs par un seul appel `scanf("%d %f", &x, &y)`, qui aurait eu le même effet.
- Un caractère d'espacement dans la chaîne de format est interprété comme un nombre quelconque de caractères d'espacement quelconques.
- `scanf` renvoie un entier pour indiquer ce qui s'est passé.
  - Si tout va bien, cet entier sera égal au nombre de paramètres supplémentaires passés (en plus de la chaîne de format). Un appel `scanf("%d %f", &x, &y)`, par exemple, est censé renvoyer 2.
  - Sinon, cet entier sera égal au nombre de champs pour lesquels on a réussi à extraire une valeur. Si l'entrée valait `"1234toto 3.14"`, l'appel `scanf("%d %f", &x, &y)` affectera `1234` à `x` et renverra 1, et l'entrée restante sera `"toto 3.14"`.
  - L'entier renvoyé sera `EOF` (ce qui signifie *End Of File*, et est une constante prédéfinie strictement négative) si on est arrivé à la fin de l'entrée et qu'on n'a rien réussi à extraire.

Exercice XIV.6

Écrire un programme acceptant en entrée :

- deux entiers n et p sur la première ligne ;
- puis n lignes contenant chacun p flottants.

Ce programme devra renvoyer en sortie n lignes, contenant chacune la somme des p nombres flottants présents sur la ligne correspondante de l'entrée.

Entrée	Sortie

`2 3`
`1.2 1.3 1.4`
`2.3 2.4 1.5`
	`3.9`
	`6.2`

Remarque
On ne s'intéressera pas au format précis d'écriture des nombres flottants en sortie (nombre de chiffres après la virgule). On peut le spécifier dans la chaîne de format de `printf`, mais ce n'est pas important pour l'instant.

3 Petits problèmes

Ces problèmes sont adaptés du site *France-IOI*. Je vous invite à vous y inscrire et à travailler dessus de temps en temps (si vous n'avez plus de TP à finir...). Les niveaux 1 et 2 n'ont pas un énorme intérêt : commencez par débloquer le niveau 3 (et demandez de l'aide si vous n'y arrivez pas, ce n'est pas évident).

Les deux premiers problèmes ci-dessous font partie de ceux à traiter pour débloquer le niveau 3 : ils ne présentent pas de difficulté algorithmique particulière, mais demandent de comprendre précisément ce qui est demandé et d'être soigneux dans son code. Le troisième problème est un peu plus délicat : une solution excessivement naïve ne s'exécutera pas en un temps satisfaisant.

Exercice XIV.7

La grande bibliothèque de la ville a actuellement des difficultés à gérer son stock car elle est très fréquentée en ce moment et ne dispose pas d'un nombre suffisant d'employés. Vous décidez de l'aider en écrivant un programme informatique permettant de soulager le travail des employés.

La bibliothèque possède $\texttt{nbLivres}$ livres indexés de 0 à $\texttt{nbLivres} - 1$. Chaque jour, un certain nombre de clients demandent à emprunter des livres pour une certaine durée. Si le livre est disponible, la requête du client est satisfaite, sinon le client repart sans livre.

Votre programme doit d'abord lire sur une première ligne deux entiers : $\texttt{nbLivres} \leqslant 1000$ et $\texttt{nbJours}$. Pour chacun des jours, votre programme lira un entier $\texttt{nbClients}$ sur une ligne puis $\texttt{nbClients}$ lignes de deux entiers. Le premier entier correspond à l'indice du livre et le second la durée correspondante. Il affichera ensuite, sur des lignes séparées, pour chaque client un 1 si le livre peut être prêté et un 0 dans le cas contraire.

On remarquera que si un client emprunte un livre le jour $\texttt{iJour}$ pendant une durée $\texttt{duree}$ alors celui-ci ne sera de nouveau disponible qu'au jour $\texttt{iJour} + \texttt{duree}$. De plus, si plusieurs personnes demandent le même livre pendant une journée, seule la première a une chance d'être satisfaite.

Entrée

```
2 4
2
0 3
1 3
1
0 3
1
1 4
2
0 2
0 5
```

Sortie

```
1
1
0
0
1
0
```

Exercice XIV.8

On considère une suite finie de nombres réels $x_0, \ldots, x_{N-1}$, et l'on considère une opération de lissage, où l'on remplace chaque valeur (sauf la première et la dernière) par la moyenne de la valeur suivante et de la valeur précédente. Par exemple, la suite $1, 3, 4, 5$ serait remplacée par $1, 2.5, 4, 5$, puis par $1, 2.5, 3.75, 5$ si l'on itère le procédé.
On demande de résoudre le problème suivant :

Entrées

- la première ligne de l'entrée contient un entier N ;
- la deuxième ligne de l'entrée contient un flottant Δ_{max} ;
- chacune des N lignes suivantes contient un flottant x_i.

Sortie il faut afficher un entier : le nombre d'étapes nécessaires pour que la différence entre deux valeurs successives de la suite ne dépasse pas Δ_{max}.

Garanties

- $1 \leqslant N \leqslant 100$
- $0 \leqslant \Delta_{max} \leqslant 100$
- $-100 \leqslant x_i \leqslant 100$ pour $0 \leqslant i < N$
- On garantit également que la propriété recherchée sera obtenue en au plus 5 000 étapes de lissage.

Exercice XIV.9

On considère une série $x_1, \ldots, x_P$ d'entiers de l'intervalle $[1 \ldots N]$. Pour $i \in [1 \ldots p]$, on définit $\texttt{ppa}(i)$ (*plus petit absent*) comme le plus petit entier $x \in [1 \ldots N]$ tel que $x \notin \{x_1, \ldots, x_i\}$, ou -1 si tous les éléments de $[1 \ldots N]$ appartiennent à $x_0, \ldots, x_i$.
On demande de résoudre le problème suivant :

Entrées

- La première ligne de l'entrée contient les deux entiers N et P, séparés par une espace.
- Les P lignes suivantes contiennent les entiers $x_1, \ldots, x_P$ à raison d'un entier par ligne.

Sortie Il faut afficher P lignes : la i-ème ligne contiendra $\texttt{ppa}(i)$.

Garanties

- $1 \leqslant N \leqslant 10^9$
- $1 \leqslant P \leqslant 250\,000$