LES POINTEURS

1. Rappels.

1.1. Pointeurs simples

Un pointeur est une variable qui contient des adresses. Au même titre qu'une variable de type **`int`** contient un entier, le contenu d'un pointeur est une adresse.

On déclare un pointeur en faisant précéder son nom d'un astérisque.

```
type *nom;
```

Les variables n'ayant pas toutes la même longueur les pointeurs sont typés. Ainsi, il y a des pointeurs sur des **`int`**, des pointeurs sur des **`float`** etc.

Avec un pointeur il est possible d'afficher, de manipuler des adresses.

```
printf("%X", adresse) ; // affiche une adresse sous forme hexadécimale (base 16)
```

Il est possible d'ajouter des adresses. Si X est une adresse correspondant à une variable **`int`**, alors X+1 est l'adresse du prochain **`int`** possible.

1.2. Variable pointée.

Si P est un pointeur, on appelle variable pointée, la variable dont l'adresse est contenue dans P.

Il est possible d'obtenir l'adresse de la variable pointée en utilisant le pointeur P : c'est son contenu

Il est possible d'obtenir le contenu de la variable pointée en utilisant le pointeur P, il suffit d'utiliser l'opérateur d'indirection *. Le tableau ci-dessous résume l'utilisation d'un pointeur P déclaré comme suit :

```
int a, *P;
P = &a ; // P contient désormais l'adresse de la variable a.
```

Code	Signification	Exemple
`P`	Utilise le contenu de P, c'est à dire l'adresse de a	`scanf("%d",P) ;`
`*P`	Utilise le contenu de la variable pointée par P, c'est à dire, ici, a.	`printf("a vaut : %d\n",*P);`

1.3. Variables dimensionnées

Une variable dimensionnée est une collection de variables du même type, regroupées sous un même nom, auxquelles on accède à l'aide d'un numéro appelé indice. On place cet indice entre crochets droits.

```
type nom[ nombre_d_éléments ] ; // déclare une variable appelée nom du type spécifié.
```

Le nom d'une variable dimensionnée est l'adresse du premier élément de cette variable. Il s'agit donc d'un pointeur, mais il est ***impossible*** de modifier le lien **`nom = &nom[0];`** contrairement aux pointeurs simples.

À l'aide de l'addition des adresses, il est possible de calculer l'adresse de l'élément k d'une variable dimensionnée :

```
&nom[ k] <=> nom + k ;
```

Il est donc possible d'accéder au contenu de la variable d'indice k d'une variable dimensionnée à l'aide de l'opérateur d'indirection.

```
nom[k] <=> *(nom + k)
```

La parenthèse est obligatoire, l'indirection est prioritaire sur l'addition.

2. Exercice 1

Tout ce qui suit doit être écrit dans la fonction main.

★ Créez deux variables de type **`float`** a, b, ainsi que deux pointeurs p1 et p2. Faites pointer p1 vers a, p2 vers b.

A partir de maintenant vous n'accéderez plus aux variables a et b qu'à travers les pointeurs.

★ Entrez, à l'aide d'instructions **`scanf`** un contenu dans a et b.

★ Calculez b = b*a

★ Affichez le contenu de a et de b

★ Comparaison : si a est plus grand que b affichez "A est plus grand" ; "B est plus grand ou égal" sinon.

L'utilisation de pointeurs tel que vu dans cet exercice est possible. Toutefois, convenez-en, cela ne sert à rien, autant utiliser a et b !

Cet exercice n'a d'autre but que de vous faire manipuler les syntaxes pointeur : il "ne faut pas" déclarer de pointeurs dans main() sauf à gérer la mémoire dynamiquement.

3. Exercice 2

★ Ecrivez une fonction qui saisisse deux **`float`** à l'aide de pointeurs, le prototype sera

```
void saisie( float *a, float *b);
```

★ Ecrivez une fonction calcule le produit de deux **`float`**, et retourne le résultat à l'aide d'un pointeur. Le prototype sera d'abord :

```
void calcul1 (float *a, float *b); // résultat dans b
```

puis

```
void calcul2 ( float a, float b, float *c); // résultat dans c
```

★ A l'aide des trois fonctions écrites ci dessus, depuis la fonction main, saisissez deux nombres réels, puis calculez et affichez leur produit. Vous n'utiliserez que deux variables dans main. D'abord avec la fonction calcul1, puis calcul 2.

4. Exercice 3

★ Dans la fonction main, déclarez une variable dimensionnée appelée X pouvant contenir 50 **`float`**.

★ Ecrivez une fonction appelée *saisie*. Elle demandera le nombre de points que l'utilisateur veut saisir, puis procédera à la saisie des points. L'interface de la fonction sera

```
void saisie( float x[50], int *n);
```

✦ Saisissez les valeurs suivantes : { 5, 9, 4, -2 } à l'aide de *saisie*, puis affichez-les depuis la fonction main.

★ Même question avec cette fois l'interface suivante pour la fonction saisie :

```
void saisie( float *x, int *n);
```

★ Créez un type de variable pouvant contenir 50 **`float`** appelé datas.

✦ Remplacez dans la fonction main, la déclaration de X par **`datas X;`** Lancez le programme *sans modifier la fonction saisie*. Conclusion ?

✦ Modifiez le programme précédent pour que l'on saisisse 4 valeurs à partir de l'élément 9 (donc dans **`X[9], X[10], X[11], X[12]`**). Affichez les 15 premiers éléments de x. Conclusion ?

★ Complétez le programme précédent en écrivant une fonction qui affiche à l'écran, les n dernières valeurs en ordre inverse d'une variable dimensionnée ainsi que leur position, la fonction aura l'interface suivante :

```
void affiche( float *P, int n);
```

Indication : Si l'on appelle la fonction ainsi : **`affiche( &X[10], 3),`** cela provoquera l'affichage suivant :

```
 0   X[10] // comprendre par X[10] la valeur de cette variable.
-1   X[9]  // idem etc.
-2   X[8]
-3   X[7]
```

★ A l'aide de ces fonctions, saisissez les 5 valeurs suivantes à partir de la position 7 de X : { 8, -2, 4, 0, 3 }. Affichez alors ces valeurs dans l'ordre inverse de leur saisie à l'aide de la fonction affiche.