MEMO SUR LES POINTEURS

I Les opérateurs & et * sont-ils réciproques ?

Rappel : les opérateurs unaires & et * sont associatifs à droite.

1) quelque soit x de type quelconque : int, float, int *, float * et même int **, float **
*&x <==> *(&x) <==> x

2) quelque soit p pointeur de type quelconque : int *, float * et même int **, float **
&*p <==> &(*p) <==> p

Attention : si x n'est pas un pointeur, l'écriture &*x n'a aucun sens, car *x n'est pas une opération licite.

3) **conclusion : Pour tout pointeur p**, on a :
*&p <==> *(&p) <==> p d'après 1)
&*p <==> &(*p) <==>p d'après 2)

donc *&p <==> &*p <==> p

Les opérateurs & et * sont réciproques dans l'ensemble des pointeurs.

Illustration :

1) int x = 5; *&x <==> *(&x) <==> x

2) int * p = &x; *&p <==> *(&p) <==> p
&*p <==> &(*p) <==> &x <==> p

on a :
p <==> &x
*p <==> x

3) int ** pp = &p; *&pp <==> *(&pp) <==> pp
&*pp <==> &(*pp) <==> &p <==> pp

on a :
pp <==> &p
*pp <==> p
**pp <==> x

Associativité droite pour l'opérateur unaire *, on a donc :
**pp <==> *(*pp) <==> *p <==> x

II Les opérateurs ++ et – : pré et post incrémentation et décrémentation

Rappel :
x = 5
y = 3
++x + y alors x vaut 6 et l'expression vaut 9
puis
x + y++ alors l'expression vaut toujours 9 et y vaut 4
puis
x + y vaut 10 car x vaut 6 et y vaut 4

Il est possible d'utiliser les opérateurs ++ et – sur les pointeurs.
++ : le pointeur avance d'un nombre d'octets égal à celui du type de la donnée pointée.
-- : le pointeur recule d'un nombre d'octets égal à celui du type de la donnée pointée.

Exemple :
int x = 12;
int * p = &x;
p++ avance de sizeof(int) octets

III Utilité des pointeurs

1) en paramètres d'une fonction afin de modifier les arguments transmis
exemple : scanf("%d", **&x**); x est modifié via une saisie clavier

2) éviter de passer en paramètre une variable de grande taille
soit X un type quelconque de grande taille (beaucoup d'octets : 300 par exemple)
sur une architecture 32 bit, un pointeur occupe 32 bits/4 octets en mémoire

Règle : **tous les pointeurs ont la même taille**

```
void f(X *);          4 octets en écriture/Write

void g(const X *);    4 octets en lecture seule/Readonly

void h(const X);      300 octets en lecture seule/Readonly
```

conséquence : g est plus rapide que h lors du passage de paramètres

```
{
        X x;
        f(&x);        // x modifié
        g(&x);        // x n'est pas modifié
         h(x);        // x n'est pas modifié
}
```

IV La parfaite complémentarité des pointeurs et des tableaux

```
int a[10];
int * pa = NULL;
p = &a[0];
```

le nom du tableau a est un pointeur constant
ERREURS DE COMPILATION : le pointeur n'est pas modifiable

```
a++;
a = a + 3;
a += 5;
a = a - 8;
```

pa est un pointeur variable
COMPILATION OK : le pointeur est modifiable

```
pa++;          pa pointe sur la case d' indice 1 du tableau
pa = pa + 3;   pa pointe sur la case d' indice 4 du tableau
pa += 5;       pa pointe sur la case d' indice 9 du tableau
pa = pa - 8;   pa pointe sur la case d' indice 1 du tableau

pa = pa + 12;  pa pointe sur la case d' indice 13 du tableau
```

Les indices autorisés sont dans l'intervalle [0, 9]
DEBORDEMENT DU TABLEAU SANS AUCUN WARNING DU COMPILATEUR !!!!
LE POINTEUR ATTEINT UNE ZONE INTERDITE, NON RESERVEE
***pa PLANTAGE PROBABLE A L'EXECUTION**
LE COMPILE NE DETECTE AUCUNE INCOHERENCE.

Conclusion :
La manipulation des pointeurs demande une rigueur absolue et est sous la totale responsabilité du programmeur.

V La notation [] avec les pointeurs

1) Repère absolu constant : l'indice 0 du tableau, le pointeur constant

Pour tout indice i : si pa = &a[0] alors p[i] <==> tab[i]

2) Repère relatif variable : l'indice correspondant à la position du pointeur variable pa

exemple :
si pa vaut &a[4] , pa pointe sur la case d'indice 4 du tableau
alors *pa vaut a[4], par définition d'un pointeur

on peut considérer que 4 est l'origine d'un repère relatif à la nouvelle position du pointeur pa
par suite :

pa[0] <==> a[4]
pa[1] <==> a[5]
pa[5] <==> a[9]
pa[-1] <==> a[3]
pa[-3] <==> a[1]
pa[-4] <==> a[0]
pa[-7] <==> a[-3] ***DEBORDEMENT***
pa[8] <==> a[12] ***DEBORDEMENT***