TP 7 : Les pointeurs

Objectif :

- Maîtriser les notions d'adresse et de pointeur

Les pointeurs

1. Rappel sur les tableaux

La déclaration d'un tableau est faite ainsi :

```
TYPE nomTableau[n]
```

Où :

- TYPE représente le type des données contenues dans le tableau ;
- nomTableau est le nom du tableau ;
- n est le nombre d'éléments du tableau.

Cette déclaration de tableau correspond à :

1. la réservation en mémoire d'une zone de n × (taille du TYPE) octets contigus pour le tableau ;
2. deux informations supplémentaires :
   a. l'adresse du premier octet de la zone réservée ;
   b. le type de l'objet.

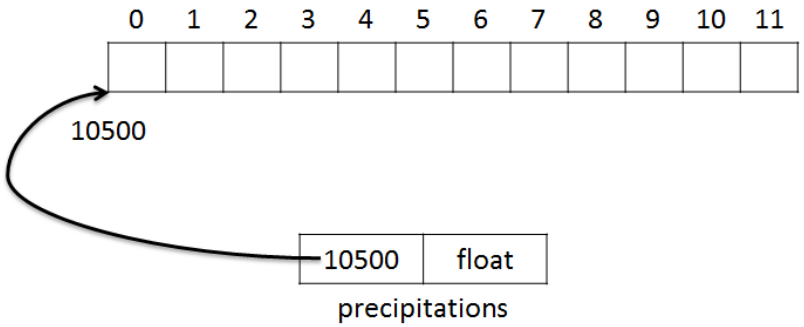
   Ces informations sons constantes : il n'est pas possible d'en modifier leurs valeurs (adresse et type).

Exemple :

La déclaration suivante :

```
float precipitations[12] ;
```

correspond à :



2. Adresses et pointeurs

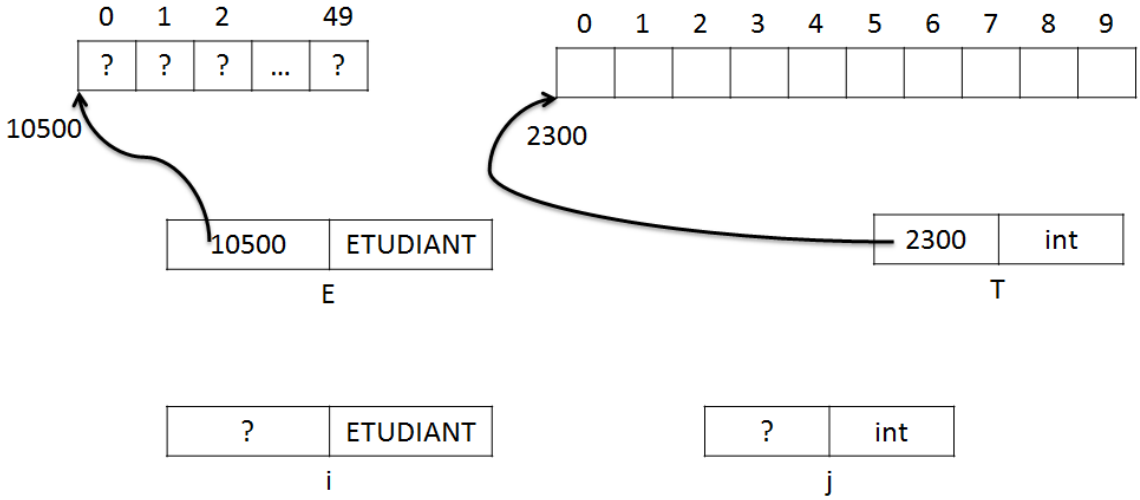
Chaque variable d'un programme C possède une adresse. Une adresse est un entier correspondant à l'emplacement du premier octet de la mémoire occupée par cette variable. Un pointeur est une variable contenant une adresse d'une variable d'un type donné.

Exemple :

Supposons que E soit un tableau d'étudiants, T un tableau d'entiers, i un pointeur vers un étudiant et j un pointeur vers un entier.
Les déclarations de ces variables se fait ainsi :

```
ETUDIANT E[50], *i ;
int T[10], *j ;
```

i est un pointeur vers un objet de type ETUDIANT (on suppose que ce type a été défini par ailleurs) ; il est donc susceptible de contenir l'adresse d'un objet de type ETUDIANT.
j est un pointeur vers un objet de type entier (i.e. est susceptible de contenir l'adresse d'un emplacement mémoire occupé par un entier ; on dit que j « pointe vers un entier »).



Les affectations suivantes ont du sens :

- i ← E : affecte le contenu de E à i (E et i ont le même type) ;
- j ← T : affecte le contenu de T à j (T et j ont le même type).

Les affectations suivantes n'ont pas de sens :

- i ← T : impossible car T et i n'ont pas le même type (le type d'un pointeur est constant) ;
- E ← i : impossible car E est constant et ne peut être changé.

3. Récupération de l'adresse d'un élément

En langage C, il existe une fonction prédéfinie qui, étant donné un objet, rend l'adresse de cet objet. Cette fonction s'appelle : &.

L'opérateur * permet de manipuler un objet pointé. Si k est un pointeur vers un entier, l'expression *k désigne l'entier pointé.

Un pointeur (de n'importe quel type) contenant la valeur NULL par convention « ne pointe sur rien » (aucune adresse ne lui est associée).

Exemple :

Soient x un entier et p un pointeur sur un entier. Ils sont déclarés et initialisés ainsi :

```
int x, *p ;
x = 5 ;
```

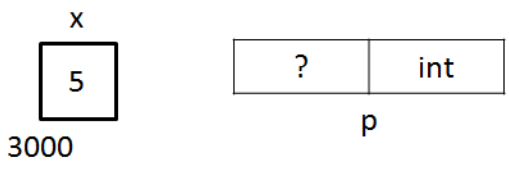


Figure 1 : Etat après la déclaration et l'affectation.

L'opération p ← &x affecte l'adresse de x à p. Après cette opération, *p aura pour valeur 5.

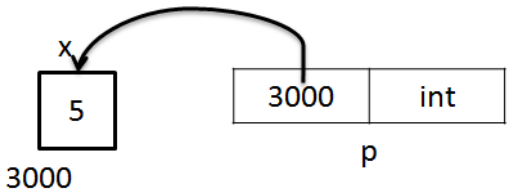


Figure 2 : Résultat de l'affection p ← &x.

Exemple :

```
int *k, l ; /* k est un pointeur, l est un entier : attention à cette
manière de déclarer ! */
l = 5 ;
k = &l ;
printf(" %d\n", l) ;
*k = 2;
printf("%d\n", l);
```

Le code ci-dessus affichera successivement les valeurs 5 et 2.

4. Arithmétique des pointeurs

Lorsque l'on déclare un tableau T, on définit en fait un doublet constant (adresse, type). La notation T[n], où n est un entier, désigne alors l'objet se trouvant à l'adresse :

adresse du doublet + n × taille du type de T.

Si i est un pointeur (adresse, type) et n un entier, alors la notation i + n désigne le doublet :

(*adresse du doublet + n × taille du type, type)*

Les expressions *(i+n) et i[n] désignent le même objet.

Exemple :

Pour un tableau E de 50 étudiants, E[4] désigne l'objet de type étudiant qui se trouve à l'adresse E + 4. Supposons que le tableau est stocké en mémoire à l'adresse 10500 et que la taille d'un objet de type Etudiant soit de 160.

E = (10500, Etudiant)

E + 4 = (11140, Etudiant)

L'expression *(E + 4) désigne également l'étudiant se trouvant à l'adresse 11140. Les deux expressions *(E + 4) et E[4] sont équivalentes.

Exemple :

Soient les déclarations et les opérations suivantes :

```
Etudiant E[50], *i, *j ;
i = E ;
j = i + 4 ;
```

Les expressions suivantes désignent toutes le même objet :

```
E[4], i[4], *j, *(E + 4) et *(i + 4)
```

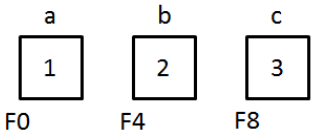
Les expressions suivantes désignent toutes un autre même objet :

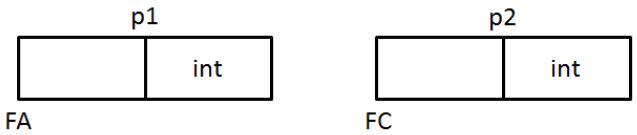
```
E[5], (i+1)[4], *(i + 5), *(j + 1), j[1] et (j – 2)[3]
```

Exercices

Exercice 1 : Manipulation des pointeurs

Soient trois entiers a, b et c et deux pointeurs de type entier p1 et p2.





1. Déterminez quelle est la valeur des différents éléments donnés dans le tableau pour chaque opération.

	a	b	c	&a	&b	&c	p1	p2	*p1	*p2
Initialisation										
p1 = &a										
p2 = &c										
***p1 = *p2**										
(*p2)++										
p1 = p2										
p2 = &b										
***p2 = *p1 – 2 * *p2**										
(*p2)--										
***p1 = *p2 - c**										
a = (2 + *p2) * *p1										

	a	b	c	&a	&b	&c	p1	p2	*p1	*p2
p2 = &c										
***p2 = *p1 / *p2**										
***p1 = a + b**										
a += *p1										
b = *p1 + *p2										
***p1 = 2 * a**										
a = *p2										
***p2 = *p1 - *p2**										
***p1 = 1 – c**										
***p2 += *p1 + a**										
p2 = p1 = &a										
p2++										
p1 += 2										
c = p2 == &c										
p1 = NULL										

2. Vérifiez que vos réponses sont correctes à partir du fichier *mainPointeur.c* et du débogueur.

Exercice 2 : Passage de paramètres par adresse

En C, seul le passage de paramètres par valeur est disponible. Le passage de paramètres résultats (i.e. paramètres dont la valeur est modifiée par une fonction) n'est pas supporté. Le seul moyen dont on dispose pour approcher ce mode de passage est d'utiliser comme paramètre effectif une adresse.

```
void incr(int *a) {
  *a = (*a) + 1 ;
}

int main() {
  int x = 5;
  incr(&x);
  printf("%d", x); /* ce programme affiche 6 */
  return 0 ;
}
```

Observer l'exécution de ce programme avec le débogueur. Ainsi, on comprend mieux la fonction *scanf* qui reçoit les adresses des variables qui se voient affecter les valeurs saisies au clavier.

Exercice 3 : Problème du drapeau hollandais

On veut trier un tableau dont les éléments ne prennent que dix valeurs (de 0 à 9) avec le critère suivant :

- tous les éléments dont la valeur est strictement inférieure à 3 sont au début du tableau ;
- tous les éléments dont la valeur est comprise entre 3 et 6 sont au milieu ;
- et tous les éléments dont la valeur est strictement supérieure à 6 sont à la fin.

Le programme affichera, par exemple :

Tableau initial :

5	8	1	4	3	9	2	7	3	8	1	4	5	3	8

Tableau trié :

1	2	1	3	4	4	5	3	3	5	9	7	8	8	8

Voici le pseudo-algorithme pour ce tri (source : Wikipédia):

```
Début
      b ← 1; w ← 1 ; r ← n
         tant que w ≤ r faire
                  si T[w] = blanc alors w ← w+1
                  sinon si T[w] = bleu alors
                           {échanger T[b] et T[w] ;
                           b ← b+1 ; w ← w+1 }
                  sinon // T[w] = rouge
                           {échanger T[w] avec T[r] ;
                           r ← r-1}
         fintantque ;
Fin
```

Ecrire le programme qui permet de trier un tableau suivant l'algorithme proposé en utilisant les pointeurs.