

Pointeurs

Structures & Pointeurs

Ce document a pour objectif de vous familiariser avec les pointeurs, c'est-à-dire avec la mémoire et les adresses en mémoire.

Définition informelle d'un pointeur¹ : « *En programmation informatique, un pointeur est un objet qui contient l'adresse mémoire d'une donnée ou d'une fonction* ».

Dans les différents exercices précédemment faits, vous avez régulièrement utilisés les tableaux et des index pour sélectionner une case à l'intérieur pour accéder au contenu. Les pointeurs fonctionnent exactement comme les index : il s'agit de l'adresse d'une case dans un tableau externe à toutes les fonctions et variables manipulées jusqu'à présent.

Les pointeurs travaillent en *mémoire*, c'est-à-dire dans un espace de stockage indépendant des variables et du contexte des fonctions. Lorsqu'une fonction est appelée, les paramètres sont copiés vers le contexte de la fonction et l'espace dédié aux variables est créé. Lorsqu'une fonction est terminée (par un *return*) ou une procédure (on termine l'exécution de la dernière instruction de la procédure), l'espace réservé pour les variables et les paramètres est libéré. Ainsi, les variables et paramètres ne peuvent exister que lors de l'exécution de la fonction/procédure qui les exécute.

Dans l'exemple suivant, les variables et paramètres n'existent que dans leur contexte. Seul le paramètre *y* fait en réalité référence à la variable *j* de la fonction *FonctionA* (*y* renvoie vers *j*).

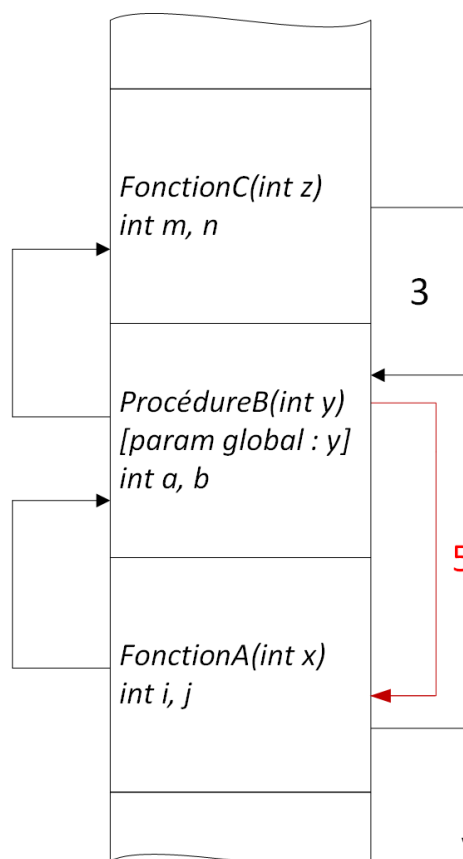
```

FonctionC : entier
  parametres locaux
    entier z
  variables
    entier m, n
m = 1
n = 2
retourner(m + n + z) // 3 + z

ProcédureB : entier
  parametres globaux
    entier y
  variables
    entier a, b
a = FonctionC(0)      // a = 3
b = 2
y = a + b              // j = y = 5

FonctionA : entier
  parametres locaux
    entier x
  variables
    entier i, j
FonctionB(j)           // (j = 5)
i = j + x              // i = 5 + x
retourner(i)           // 5 + x

```



1. Wikipedia : Pointeur


Dans les ordinateurs, la *mémoire* est l'espace où peuvent être stockées des données lors de l'exécution des programmes. On sépare généralement la *mémoire* (l'espace où l'on stocke les données dont l'accès doit être immédiat) de la *mémoire de masse* (l'espace où l'on stocke les données dont l'accès peut être différé et/ou dont le volume est trop gros pour entrer en mémoire vive).

Pour illustrer, dans une bibliothèque, il est nécessaire d'avoir accès instantanément à la liste des livres appartenant à la bibliothèque, lesquels sont disponibles en rayon, et dans quel rayon les trouver, ou éventuellement lesquels sont disponibles en réserve. Dans ce cas, l'index doit être en *mémoire vive*, et les livres en *mémoire de masse* : on cherche d'abord si le livre est disponible et où il se trouve, avant de chercher parmi tous les rayons et les livres existants.

Toujours dans l'exemple de la bibliothèque, le livre serait la donnée visée, et sa côte (le numéro de référence indiquant le rayon où le trouver) serait son adresse, donc un pointeur faisant référence à ce livre.

La mémoire est donc l'espace où l'on stocke les données, et un pointeur est littéralement une adresse (ou référence) vers une case de cet espace. Dans l'exemple suivant, on a donc la mémoire représentée comme un grand tableau, et ici, on a un pointeur avec l'adresse 44 qui fait référence à une case contenant la valeur 91.

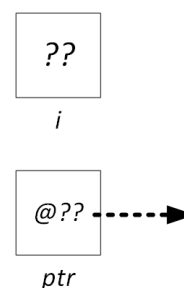
Pointeur vers
l'adresse 44
contenant 91



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|----|----|----|----|----|----|----|----|----|
| 30 | 42 | 18 | 36 | 15 | 90 | 60 | 75 | 42 | 38 | 14 |
| 40 | 12 | 34 | 56 | 78 | 91 | 01 | 11 | 21 | 31 | 41 |
| 50 | 51 | 20 | 95 | 43 | 22 | 79 | 00 | 14 | 24 | 37 |
| 60 | 84 | 62 | 35 | 12 | 34 | 44 | 52 | 64 | 32 | 16 |

Pour déclarer une variable comme étant un pointeur, on utilise l'opérateur * après le nom du type. Par exemple pour un pointeur vers un entier, on va déclarer :

```
variables
int    i
int    *ptr
```



Les déclarations se lisent de la droite vers la gauche. On peut donc lire ces deux variables comme étant : *i est un entier, ptr est un pointeur vers un entier*.

Pour *déréférencer* un pointeur, c'est-à-dire pour accéder à la valeur dans la case visée, on utilise également l'opérateur *, mais d'une autre manière.

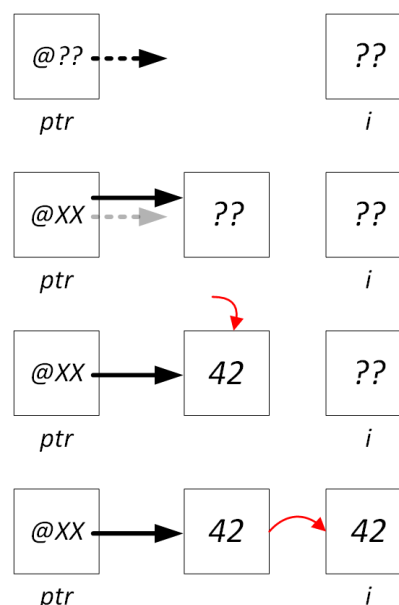
variables

```

int    i
int    *ptr

ptr = malloc(sizeof (int))
(*ptr) = 42
i = (*ptr)

```



Ici, on demande au système de nous donner une case mémoire pouvant accueillir un entier. Le système se débrouille pour trouver un espace de taille suffisante, puis, il renvoie l'adresse de cette case (*ptr* récupère donc l'adresse de cette case).

Ensuite, on indique que l'on déréfère la variable *ptr* pour y mettre 42. C'est-à-dire que l'on indique que l'on résout l'adresse de *ptr* pour accéder à la case visée, puis, on y met la valeur 42.

La dernière ligne indique que l'on déréfère *ptr* (on résout l'adresse de *ptr*) pour récupérer la valeur au bout et la mettre dans la variable *i*. Maintenant, *i* contient donc 42.

On peut également *faire référence* à une variable ou autre. Pour cela, on va utiliser l'opérateur `&`.

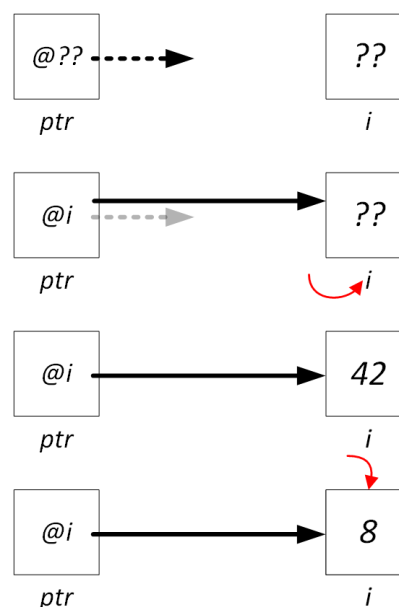
variables

```

int    i
int    *ptr

ptr = &i
i = 42
(*ptr) = 8

```

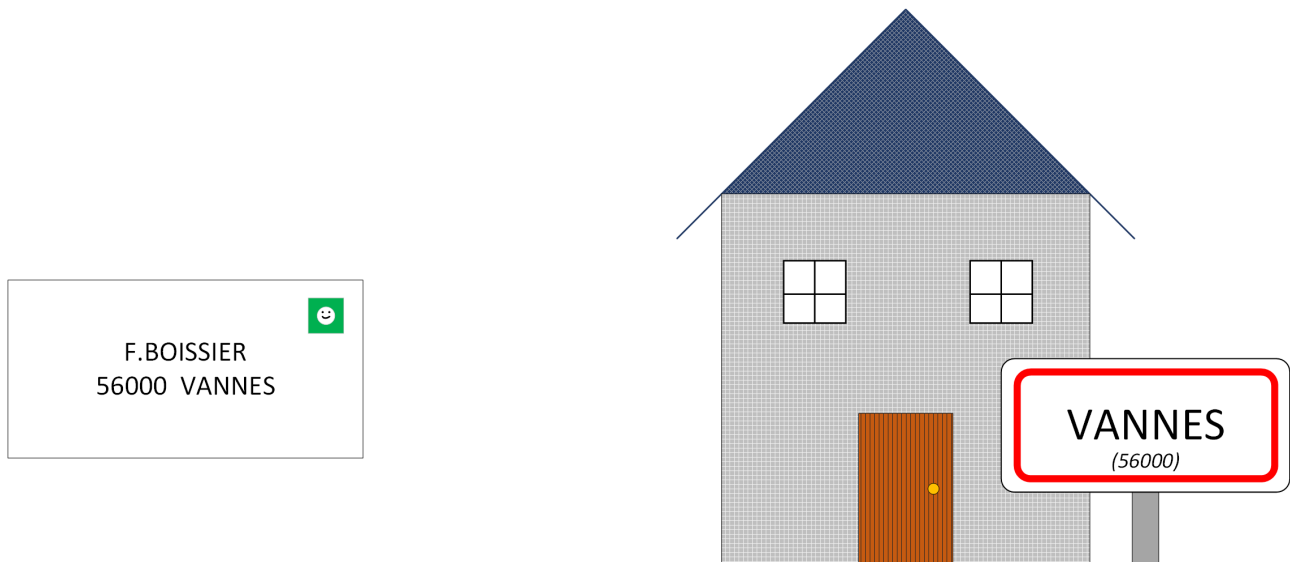


Ici, on indique que *ptr* va prendre l'adresse permettant de pointer vers *i*. *ptr* a donc pris une référence vers *i*. On remarque que *ptr* n'a pas été déréféré : c'est normal, *ptr* est une adresse, et on lui donne l'adresse vers *i*.

Ensuite, dans la deuxième ligne, on met la valeur 42 dans *i*. Si on déréfère *ptr*, on y trouvera donc 42.

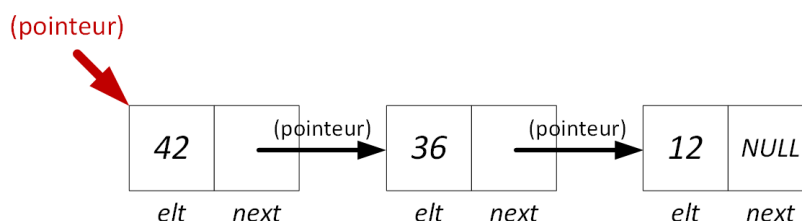
La dernière ligne indique que l'on va mettre 8 dans la case pointée par *ptr*. Comme *ptr* pointe vers la variable *i*, on est train de mettre 8 dans *i*. Finalement, *i* contient donc 8.

Pour résumer avec un schéma distrayant, on peut tout simplement repenser au concept d'adresse dans le monde réel.



- L'adresse sur l'enveloppe est un pointeur vers une maison très précise.
- Chaque maison (ou case mémoire) est physiquement à une seule adresse précise.
- Lorsque la Poste prend l'enveloppe et lit l'adresse pour pouvoir livrer le courrier au bon endroit, elle est en train de *déréférencer* l'adresse vers la maison qui doit recevoir l'enveloppe. La Poste est donc en train d'effectuer une opération similaire à ce que ferait cette expression : (*adresse) ou en mettant la vraie valeur : (*56000)
(Vous vous rendez également compte que le code postal 56000 correspond en réalité à Vannes)
- À l'inverse, lorsque quelqu'un vous demande où vous habitez, cette personne vous demande une *référence vers* votre maison, c'est-à-dire votre adresse. En donnant l'adresse de votre logement, vous êtes en train de donner une référence vers votre maison comme le ferait l'expression : &(Ma Maison) c'est-à-dire l'adresse et le code postal 56000
(Donner la référence vers ma maison, c'est donner l'adresse de mon logement, donc indiquer que j'habite à Vannes avec le code postal 56000)
- Si vous indiquez à la Poste de rediriger vers une autre adresse tous les courriers arrivant à votre adresse, alors vous vous rapprochez du concept où l'on définit des pointeurs de pointeurs : ** adresse
(Le premier déréférencement accède bien à une case, mais celle-ci contient une adresse indiquant en fait qu'il faut déréférencer de nouveau pour atteindre la maison/l'emplacement réel de la donnée)

Dans peu de temps, vous allez implémenter les listes chaînées à base de pointeurs. Cette structure de données permet de prendre des éléments et les stocker à des positions précises, puis de les sortir de la liste selon leur position ou d'autres critères. Fonctionnellement parlant, il s'agit donc de stocker des éléments dans un certain ordre, comme dans le schéma suivant :



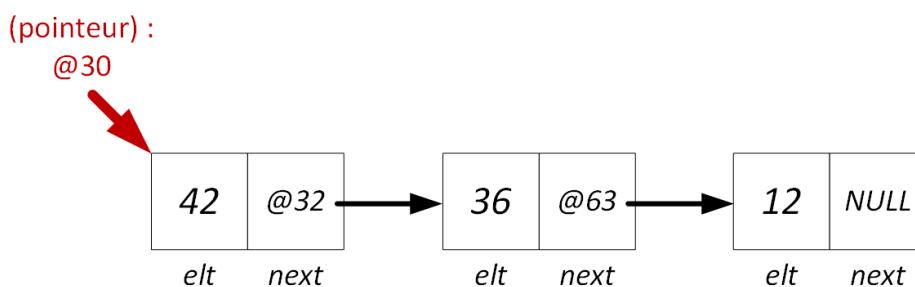
Vous pouvez observer que la structure de données contient à la fois un élément et un pointeur vers le maillon suivant. Le dernier maillon dispose d'un pointeur *NULL*.

Cette valeur *NULL* est spéciale, car lorsqu'on la déréférence, cela déclenche une erreur. Il ne faut donc JAMAIS la déréférencer. Par contre, c'est une valeur remarquable que l'on peut tester.

Ainsi, avant de déréférencer un pointeur, ou lorsque l'on veut constater que l'on est au dernier élément de la liste, on teste si le pointeur est à *NULL* ou pas.

Revenons sur la mémoire, les pointeurs, et la liste chaînée. Pour stocker en mémoire cette liste, il a fallu appeler la fonction *allouer* (ou *malloc* en C) pour chaque maillon ajouté. Cette fonction demande au système de trouver de l'espace et de le donner au programme. Vous ne pouvez quasiment pas deviner où se trouve cette case en mémoire. Si votre programme a fait énormément d'allocations en mémoire et de libérations de la mémoire, alors les éléments peuvent se trouver à des endroits très distincts/des espaces non contiguës.

L'image suivante illustre cela : le maillon contenant l'élément 42 est à l'adresse 30, le maillon suivant contenant l'élément 36 se trouve à l'adresse 32, et enfin, le dernier maillon contenant l'élément 12 se trouve à l'adresse 63 (on notera que c'est le dernier élément parce que le pointeur vers l'élément suivant a l'adresse *NULL*).



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|----|----|----|----|----|----|----|----|----|
| 30 | 42 | 32 | 36 | 63 | 90 | 60 | 75 | 42 | 38 | 14 |
| 40 | 12 | 34 | 56 | 78 | 91 | 01 | 11 | 21 | 31 | 41 |
| 50 | 51 | 20 | 95 | 43 | 22 | 79 | 00 | 14 | 24 | 37 |
| 60 | 84 | 62 | 35 | 12 | 00 | 44 | 52 | 64 | 32 | 16 |

Ainsi, les pointeurs sont simplement des *adresses* vers des cases mémoire ($int * ptr$), et lorsque l'on souhaite accéder à la donnée au bout de l'adresse/au bout du pointeur on effectue un *déréférencement* ($(*ptr)$).

Pour retrouver l'adresse d'un élément que l'on manipule, on demande une *référence vers* cet élément ($ptr = \&i$).

Ce document et ses illustrations ont été réalisés par Fabrice BOISSIER en octobre 2022