Les Pointeurs

- Toute variable manipulée dans un programme est stockée quelque part en **mémoire centrale**.
- Cette mémoire est constituée d'octets qui sont identifiés de manière univoque par un numéro qu'on appelle **adresse**.
- Pour retrouver **une variable**, il suffit donc de connaitre **l'adresse de l'octet ou elle est stockée.**
- Pour des raisons évidentes de lisibilité, on désigne souvent les variables par des **identificateurs**, et non par leur adresse.
- C'est **le compilateur** qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire.
- Toutefois, il est parfois très pratique de manipuler directement une variable par son adresse.

- On appelle **Lvalue** (left value) tout objet pouvant être placé à gauche d'un opérateur d'affectation.

- Une **Lvalue** est caractérisée par:
  -**son adresse**, c'est-à-dire l'adresse-mémoire à partir de laquelle l'objet est stocké;
  -**sa valeur**, c'est-`a-dire ce qui est stocké à cette adresse.

Dans l'exemple,

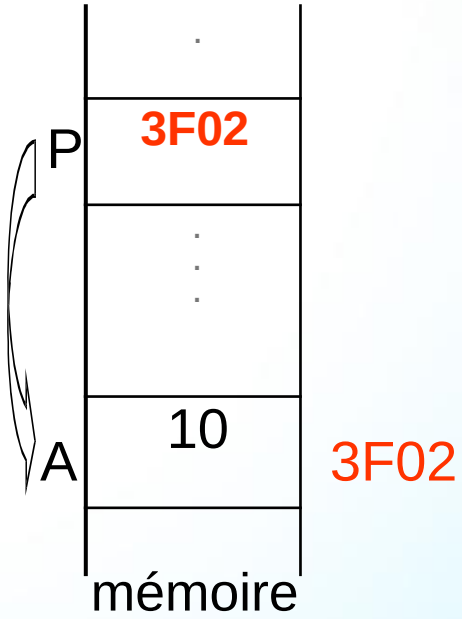
- int i, j;
  i = 3;
  j = i;

objet	adresse	valeur
i	4831836000	3
j	4831836004	3

- Deux variables différentes ont des adresses différentes. L'affectation j = i; n'opère que sur les valeurs des variables.

Pointeurs : définition

- Un pointeur est une variable spéciale qui peut contenir l'adresse d'une autre variable.
- Exemple : Soit A une variable contenant la valeur 10 et P un pointeur qui contient l'adresse de A *(on dit que P pointe sur A)* .
  - Remarques :
    - Le nom d'une variable permet d'accéder *directement* à sa valeur (**adressage direct**).
    - Un pointeur qui contient l'adresse de la variable, permet d'accéder *indirectement* à sa valeur (**adressage indirect**).
    - Le nom d'une variable est lié à la même adresse, alors qu'un pointeur peut pointer sur différentes adresses



Intérêts des pointeurs

- Les pointeurs présentent de nombreux avantages :
  - Ils sont indispensables pour permettre le passage par référence pour les paramètres des fonctions
  - Ils permettent de créer des structures de données (listes et arbres) dont le nombre d'éléments peut évoluer dynamiquement. Ces structures sont très utilisées en programmation.
  - Ils permettent d'écrire des programmes plus compacts et efficaces

Déclaration d'un pointeur

- En C, chaque pointeur est limité à un type de donnée (même si la valeur d'un pointeur, qui est une adresse, est toujours un entier).
- Le type d'un pointeur dépend du type de la variable pointée. Ceci est important pour connaître la taille de la valeur pointée.
- On déclare un pointeur par l'instruction : **type *nom-du-pointeur ;**
  - type est le type de la variable pointée
  - ***** est l'opérateur qui indiquera au compilateur que c'est un pointeur (désigne en fait le contenu de l'adresse)

Déclaration d'un pointeur

- Exemple :

```
int *pi; //pi est un pointeur vers une variable de type int
float *pf; //pf est un pointeur vers une variable de type float
char *b; /* un pointeur b vers un caractère*/
double *y; /* un pointeur a vers un réel double*/
```

- Rq: la valeur d'un pointeur donne l'adresse du premier octet parmi les n octets où la variable est stockée

Déclaration d'un pointeur

- Dans la déclaration d'un pointeur, le symbole * est associé au nom de la variable pointeur et non pas au type

- Ainsi la déclaration:

int *a,b;

Ne déclare pas deux pointeurs a et b, mais un pointeur a vers un int et une variable simple b de type int.

- Pour déclarer deux pointeurs a et b dans la même ligne, il faut écrire:

int*a,*b;

Opérateurs de manipulation des pointeurs

- Lors du travail avec des pointeurs, nous utilisons :
  - un **opérateur 'adresse de': &** pour obtenir l'adresse d'une variable
  - un **opérateur 'contenu de': *** pour accéder au contenu d'une adresse

- Exemple :

```
int * p; //on déclare un pointeur vers une variable de type int
int i=10, j=30; // deux variables de type int
p=&i; // on met dans p, l'adresse de i (p pointe sur i)
printf("*p = %d \n",*p); //affiche : *p = 10
*p=20; // met la valeur 20 dans la case mémoire pointée par p (i vaut 20 après cette instruction)
p=&j; // p pointe sur j
i=*p; // on affecte le contenu de p à i (i vaut 30 après cette instruction)
```

Opérateurs de manipulation des pointeurs

- Considérons un pointeur p qui pointe vers une variable a de type T.

- L'expression ***p** est équivalente à **a**. Donc, on peut manipuler la variable a :
  - soit directement par son nom
  - soit indirectement par *p.

- En fait ***p** est considérer comme une variable de type T et tout ce qui applicable sur a, on peut l'appliquer sur ***p**

Opérateurs de manipulation des pointeurs

- Exemple :

Déclarer une variable de type int et l'initialiser par la valeur 5, puis déclarer un pointeur de type int et l'initialiser par l'adresse de la variable déclarée.

```
int a=5;
int*p=&a;
```

Modifier la valeur de a à 9 en utilisant les deux méthodes citées dans la définition.

```
a=9;/*modification directe de la valeur de a*/
*p=9;/* modification indirecte à l'aide d'un pointeur*/
```

Opérateurs de manipulation des pointeurs

- Exemple2 :
```
float a, *p;
p=&a;
printf("Entrez une valeur : \n");
scanf("%f ",p); //supposons qu'on saisit la valeur 1.5
printf("Adresse de a= %x, contenu de a= %f\n" , p,*p);
*p+=0.5;
printf ("a= %f\n" , a); //affiche a=2.0
```
- **Remarque :** si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X
  - X+=2 équivaut à *P+=2
  - ++X équivaut à ++ *P
  - X++ équivaut à (*P)++ // les parenthèses ici sont obligatoires car l'associativité des opérateurs unaires * et ++ est de droite à gauche

Exercice

Déterminer la valeur de la variable a après l'exécution de chacune des instructions suivantes:

```
int a=4;
int *p=&a;
(*p)++;
*p=(*p)*(*p);
```

Solution:

- ✓ a est une variable de type int initialisée à 4.
- ✓ p est un pointeur vers a. Donc *p vaut 4.
- ✓ (*p)++ permet incrémenter l'objet pointé par p, à savoir a. La valeur de a vaut maintenant 5.
- ✓ *p=(*p)*(*p), permet de calculer le carré de a et de l'affecter à a. Donc la variable a vaut 25.

Initialisation d'un pointeur

- A la déclaration d'un pointeur p, on ne sait pas sur quel zone mémoire il pointe. Ceci peut générer des problèmes :
  - **int *p;**

    ***p = 10;** //provoque un problème mémoire car le pointeur p n'a pas été initialisé

- **Conseil :** Toute utilisation d'un pointeur doit être précédée par une initialisation.
- On peut initialiser un pointeur en lui affectant :
  - l'adresse d'une variable (Ex: **int a, *p1; p1=&a;**)
  - un autre pointeur déjà initialisé (Ex: **int *p2; p2=p1;**)
  - la valeur 0 désignée par le symbole NULL, défini dans **<stddef.h>.**

    Ex: **int *p; p=0;ou p=NULL;** (on dit que p pointe 'nulle part': aucune adresse mémoire ne lui est associé)
- Rq: un pointeur peut aussi être initialisé par une allocation dynamique (voir fin du chapitre)

Affectation de pointeurs

- L'affectation d'un pointeur à un autre est possible si les deux pointeurs sont de même type T, alors **p=q**; permet d'affecter le pointeur q au pointeur p.

- Cela a pour effet de faire pointer p et q vers la même variable.

- Exemple:

  Déclarer une variable de type int et l'initialisée par la valeur 7, puis déclarer deux pointeurs de type int qui vont pointer vers la variable déclarée.

```
int a=7;
int*p=&a;
 int*q=p;
```

Opérations arithmétiques avec les pointeurs

- La valeur d'un pointeur étant un entier, certaines opérations arithmétiques sont possibles : ajouter ou soustraire un entier à un pointeur ou faire la différence de deux pointeurs

- Pour un entier i et des pointeurs p, p1 et p2 sur une variable de type T
  - **p+i (resp p-i)** : désigne un pointeur sur une variable de type T. Sa valeur est égale à celle de p incrémentée (resp décrémentée) de i*sizeof(T).
  - **p1-p2 :** Le résultat est un entier dont la valeur est égale à (différence des adresses)/sizeof(T).
- Remarque:
  - on peut également utiliser les opérateurs ++ et -- avec les pointeurs
  - la somme de deux pointeurs n'est pas autorisée

Opérations arithmétiques avec les pointeurs

- Exemples:

1. Soit a une variable de type int et p est un pointeur vers a. Sachant que le type est codé sur 4 octets et que l'adresse de la variable a est 1030:
   - quel est le contenu du pointeur p+3.
   - quel est le contenu du pointeur p-3.

2. La variable a de type int est rangée en mémoire à l'adresse 1000.
   Quel est le contenu de chacun des pointeurs suivants:

1.
```
int *p=&a;
int*q=p++;
int*r=++q;
```

2.
```
int *p=&a;
int*q=p--;
int*r=--q;
```

Opérations arithmétiques avec les pointeurs

1.

Le pointeur p pointe vers a, donc le contenu de p est l'adresse de p, c'est-à-dire 1030.

- Le pointeur p+3 pointe vers l'entier qui se trouve 3 fois après celui que contient la variable a. Donc le contenu du pointeur p+3 est 1030+3*4=1042.

- Le pointeur p-3 pointe vers l'entier qui se trouve 3 fois avant celui que contient la variable a. Donc le contenu du pointeur p-3 est 1030-3*4=1018

Opérations arithmétiques avec les pointeurs

2.

Le pointeur p pointe vers a, donc le contenu de p est l'adresse de p, c'est-à-dire 1000.

- q=p++; est équivalente à q=p; p=p+1; donc le contenu du pointeur q est 1000, mais le pointeur p est incrémenté et par suite, le contenu de est 1004.

- r=++q; est équivalente à q=q+1;r=q; Donc le contenu de q est 1004, et celui de r est 1004

- q=p--; est équivalente à q=p; p=p-1; Donc le contenu du pointeur q est 1000, mais le pointeur p est décrémenté et par la suite, le contenu de p est 996.

- r=--q; est équivalente à q=q-1;r=q; Donc le contenu de q est 996, et celui de r est 996.