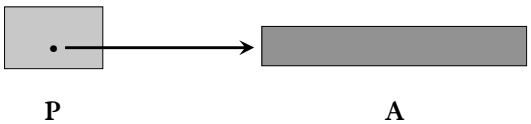
- ***CHAPITRE 7* : Pointeurs en langage C**
  - Introduction : Définition et Intérêts
  - Déclaration et initialisation d'un pointeur
  - Opérations élémentaires sur les pointeurs
  - Pointeurs et tableaux
  - Tableaux de pointeurs
  - Allocation dynamique de la mémoire

CHAPITRE 7 : Pointeurs en langage C

1. Introduction

1.1 Définition

- Un **pointeur** est une variable spéciale qui peut contenir l'**adresse** d'une autre variable.
- En C, chaque pointeur est limité à un type de données. Il peut contenir l'adresse d'une variable de ce type.
- Si un pointeur P contient l'adresse d'une variable A, on dit que **'P pointe sur A'**.



CHAPITRE 7 : Pointeurs en langage C

1. Introduction

1.2 Intérêts

- En C, l'utilisation de pointeurs est incontournable car ils sont étroitement liés à la représentation des tableaux
- Les **principales intérêts** des pointeurs résident dans la possibilité de :
  - Allouer de la mémoire dynamique sur le TAS, ce qui permet la gestion de structures de taille variable. Par exemple, tableau de taille variable.
  - Permettre le passage par référence pour des paramètres des fonctions (clef chapitre 8)
  - Réaliser des structures de données récursives (listes et arbres) (clef cours I4 structures de données)
  - …

CHAPITRE 7 : Pointeurs en langage C

2. Déclaration et initialisation d'un pointeur

2.1 Déclaration

- Un pointeur est une variable dont la valeur est égale a l'adresse d'une autre variable. En C, on **déclare un pointeur** par l'instruction :

type *nom_du_pointeur ;

où

- type est le type de la variable pointée,
- l'identificateur nom_du_pointeur est le nom de la variable pointeur et
- * est l'opérateur qui indiquera au compilateur que c'est un pointeur.

- Exemple : **int *p;**

  On dira que :

  p est un pointeur sur une variable du type **int** , ou bien

  p peut contenir l'adresse d'une variable du type **int**

  *p est de type int, c'est l'emplacement mémoire pointé par p.

CHAPITRE 7 : Pointeurs en langage C

2. Déclaration et initialisation d'un pointeur 2.1 Déclaration

Remarques :

- A la déclaration d'un pointeur p, il ne pointe a priori sur aucune variable précise : p est un pointeur non initialisé.

  Toute utilisation de p devrait être précédée par une initialisation.

- la valeur d'un pointeur est toujours un entier (codé sur **16 bits**, **32 bits** ou **64 bits**). le type d'un pointeur dépend du type de la variable vers laquelle il pointe. Cette distinction est indispensable à l'interprétation de la valeur d'un pointeur. En effet

  Pour un pointeur sur une variable de type char, la valeur donne **l'adresse de l'octet** ou cet variable est stockée.

  Pour un pointeur sur une variable de type short, la valeur donne **l'adresse du premier des 2 octets** où la variable est stockée

  Pour un pointeur sur une variable de type float, la valeur donne **l'adresse du premier des 4 octets** où la variable est stockée

CHAPITRE 7 : Pointeurs en langage C

2. Déclaration et initialisation d'un pointeur

2.2 Initialisation

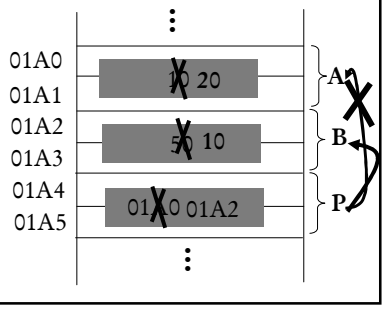
Pour initialiser un pointeur, le langage C fournit l'opérateur unaire **&**.
Ainsi pour récupérer l'adresse d'une variable A et la mettre dans le pointeur P (P pointe vers A) :

P = & A

Exemple 1 :

int A, B, *P; /*supposons que ces variables occupent la mémoire à partir de l'adresse 01A0 */

A = 10;
B = 50;
P = &A ; // se lit mettre dans **P** l'adresse de **A**
B = *P ; /* mettre dans **B** le contenu de de la variable pointe par **P***/
P = 20;** /*mettre la valeur **20** dans la variable pointé par **P/
P = &B; // **P** pointe sur **B**



CHAPITRE 7 : Pointeurs en langage C

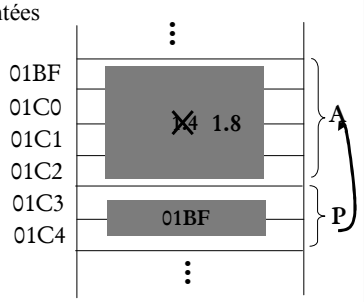
2. Déclaration et initialisation d'un pointeur

Exemple 2 :

```
#include <stdio.h>
#include <conio.h>

Void main( )
{ float  a , *p;  /*supposons que ces variables sont représentées
                    en mémoire à partir de l'adresse 01BF*/
   clrscr( ); // pour effacer l'écran → <conio.h>
   p = &a;
   printf("Entrer une valeur :");
   scanf("%f",p);  // on saisie la valeur 1.4

   printf("\nAdresse de a = %x  Contenu de a = %f",p,*p);
   *p += 0.4;
   printf("a = %f       *p = %f " , a,*p);
   getch( ); // pour lire un caractère → <conio.h>
}
```



CHAPITRE 7 : Pointeurs en langage C

3. Opérations élémentaires sur les pointeurs

- L'opérateur & : 'adresse de' : permet d'obtenir l'adresse d'une variable.
- L'opérateur * : 'contenu de' : permet d'accéder au contenu d'une adresse.
- Si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X.
- **Exemple :** int X=1, Y, *P Après l'instruction, **P = &X** ;

  On a :

Y = X + 1	équivalente à	Y = *P + 1
X += 2	équivalente à	*P += 2
++X	équivalente à	++ *P
X++	équivalente à	(*P)++

CHAPITRE 7 : Pointeurs en langage C

3. Opérations élémentaires sur les pointeurs (suite)

- Le seul entier qui puisse être affecté à un pointeur d'un type quelconque P est la constante entière 0 désignée par le symbole NULL défini dans <stddef.h>.

  On dit alors que le **pointeur P ne pointe 'nulle part'.**

- Exemple :

```
#include <stddef.h>
...
int  *p,  x , *q;
short  y = 10 , *pt = &y;
p = NULL ;  /* Correct */
p = 0 ;         /* Correct */
x = 0 ;
p = x ;        /* Incorrect ! bien que x vaille 0 */
q = &x ;
p = q ;        /* Correct : p et q pointe sur des variables de même type*/
p = pt ;       /* Incorrect : p et pt pointe sur des variable de type différent */
```

CHAPITRE 7 : Pointeurs en langage C

Exercices

Trouvez les erreurs dans les suites d'instruction suivantes :

a) **int *p , x = 34; *p = x;**
   ***p = x est incorrect parce que le pointeur p n'est pas initialisé**

b) **int x = 17 , *p = x; *p = 17;**
   ***p = x est incorrect. Pour que p pointe sur x → *p = &x**

c) **double *q; int x = 17 , *p = &x; q = p;**
   q = p incorrect. q et p deux pointeurs sur des types différent

d) **int x, *p; &x = p;**
   &x = p incorrect. &x n'est pas une variable (lvalue) et par conséquent ne peut pas figurer à gauche d'une affectation.

e) **char mot[10], car = 'A', *pc = &car ; mot = pc;**
   mot = pc incorrect. mot est un pointeur constant et on ne peut pas changer sa valeur. Ce n'est pas une variable (lvalue).