INF 2331- Structures de données

LGI – Semestre 3

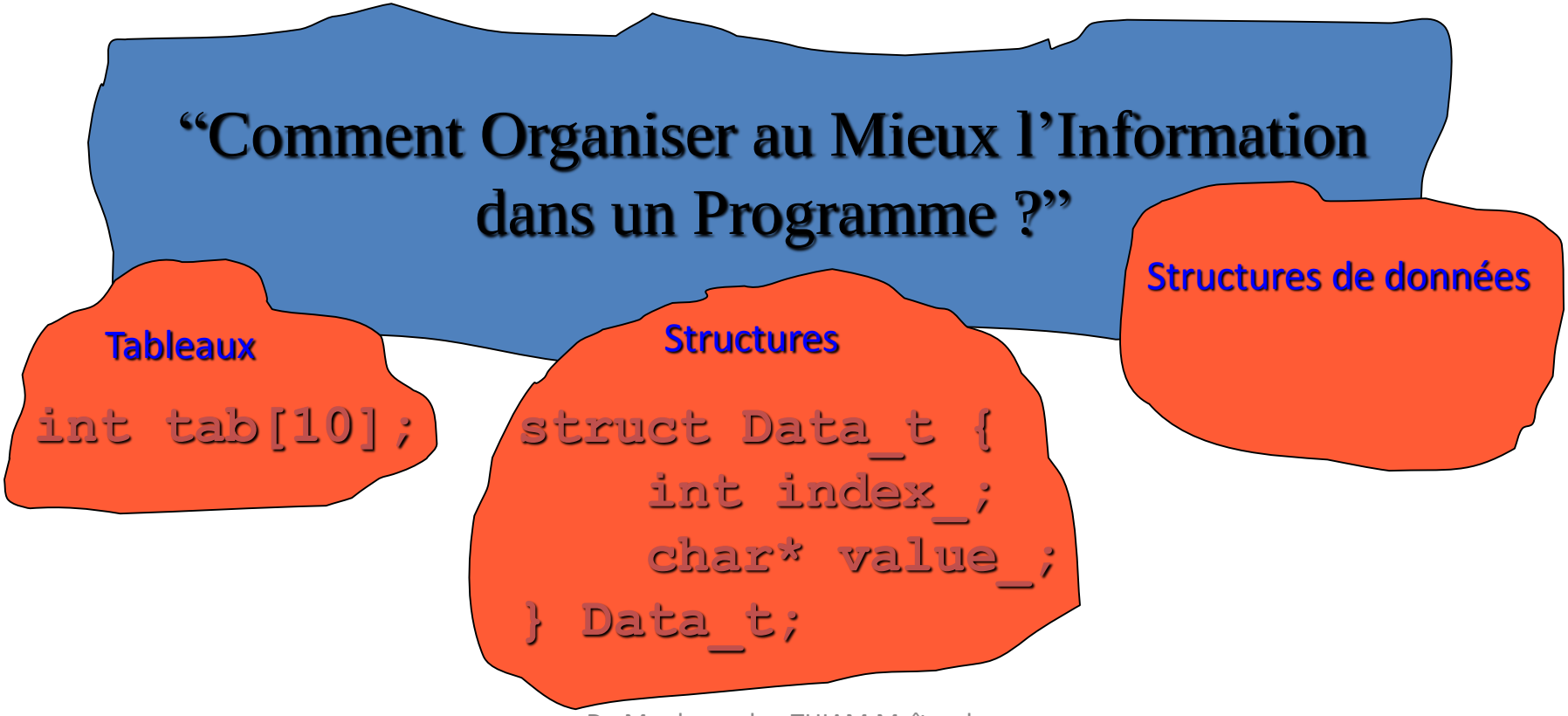
Département informatique

UFR des Sciences et technologies

Université de Thiès

Introduction

Problème métaphysique:



Structure de données

Définition Wikipédia (14/03/2015)

- une structure logique destinée à contenir des données afin de leur donner une organisation permettant de simplifier leur traitement.

- **Exemple :** On peut présenter des numéros de téléphone *

  - par département,
  - par nom
  - par profession (pages jaunes),
  - par numéro téléphonique (annuaires destinés au télémarketing),
  - par rue et/ou
  - une combinaison quelconque de ces classements.

  À chaque usage correspondra une structure d'annuaire appropriée.

Objectifs du cours

- Connaître les principales structures de données manipulées dans un programme
- Connaître les notions de structures linéaires
- Connaître les notions de structures arborescentes
- Connaître des arbres particuliers
- Connaître les principes de recherche rapides
- Savoir appliquer les concepts de base ci-dessus à la programmation (tp)
- Connaître les notions de graphes (facultatif)

Références

- Web
- Aho et al. *Structures de données et algorithmes*, Addisson-Wesley / InterEditions. 1989.
- Aho et Ullman. *Concepts fondamentaux de l'informatique*, Dunod. 1993.
- Sedgewick. *Algorithmes en C*. Addisson-Wesley / InterEditions. 1991.
- Alfred Aho John Hopcroft, Jeffrey Ullman : Structures de données et algorithmique

Overview

1. CHAPITRE 0 : PRÉSENTATION DU COURS
2. CHAPITRE 1 : LES STRUCTURES LINÉAIRES
3. CHAPITRE 2 : ARBRES ET ARBORESCENCES
4. CHAPITRE 3 : QUELQUES ARBRES PARTICULIERS
5. CHAPITRE 4 : RECHERCHES RAPIDES

Overview

1. CHAPITRE 0 : PRÉSENTATION DU COURS
2. CHAPITRE 1 : LES STRUCTURES LINÉAIRES
3. CHAPITRE 2 : ARBRES ET ARBORESCENCES
4. CHAPITRE 3 : QUELQUES ARBRES PARTICULIERS
5. CHAPITRE 4 : RECHERCHES RAPIDES

PRÉSENTATION DU COURS

Présentation générale

- **Unité d'Enseignement**
  - **Titre :** INFORMATIQUE
  - **Sigle** : **INF 233**
- **Élément constitutif**
  - **Titre :** Structures de données
  - **Sigle** : **INF 2331**
- **Autres éléments constitutifs de l'UE (1)**
  - Analyse et Conception des Systèmes d'Information (**INF 2332**)
  - Bases de données (**INF 2333)**

Volume horaire & Notation

- **CM :** 30H
- **TD/TP :** 20H
- **TPE :** 50H
- **Coefficient de l'UE :** 4
- **Crédits de l'UE :** 12
- **Evaluation**
  - **Contrôle des connaissances :** 40%
  - **Examen écrit :** 60%

Responsables

- **Magistral**

  Dr. Mouhamadou THIAM
  Maître de conférences en Informatique
  Intelligence Artificielle : Sémantique Web
  Email : mthiam@univ-thies.sn

- **Travaux dirigés et pratiques**

  M. Papa DIOP
  Ingénieur Systèmes Informatiques et Bases de Données
  Diplômé de l'Université Gaston Berger de Saint-Louis
  Email : papaddiop@gmail.com

Overview

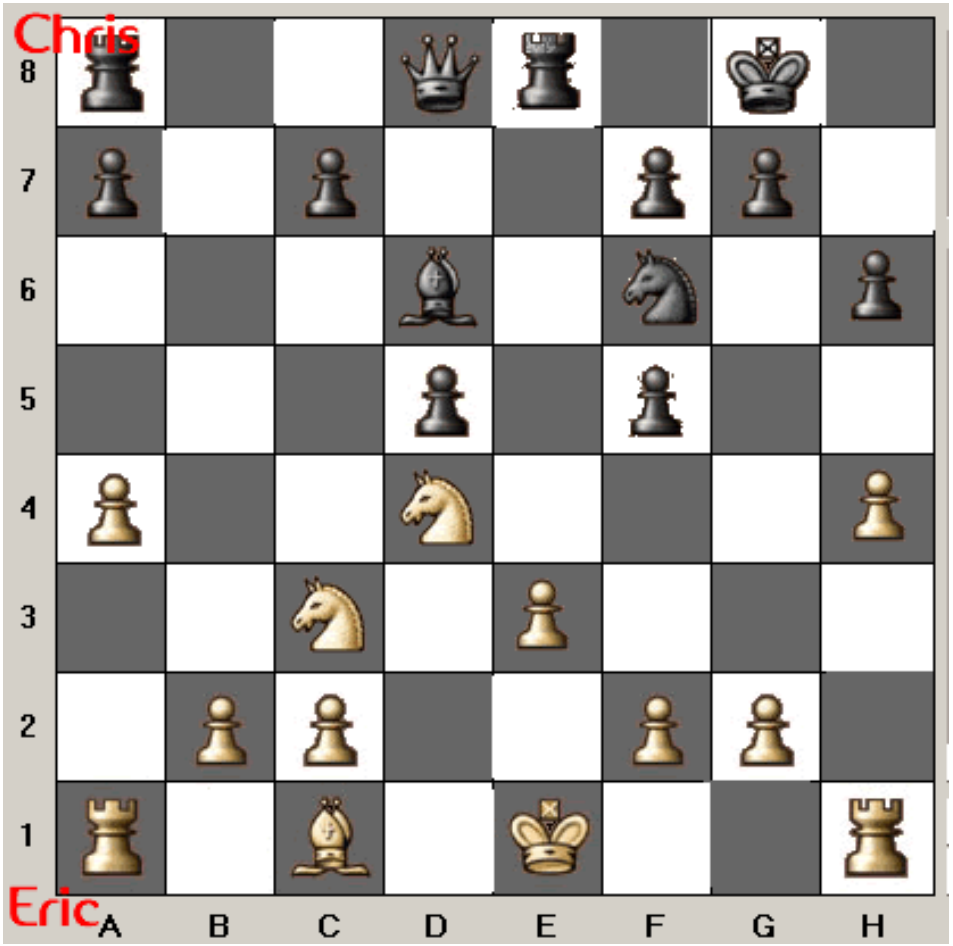
1. CHAPITRE 0 : PRÉSENTATION DU COURS
2. CHAPITRE 1 : LES STRUCTURES LINÉAIRES
3. CHAPITRE 2 : ARBRES ET ARBORESCENCES
4. CHAPITRE 3 : QUELQUES ARBRES PARTICULIERS
5. CHAPITRE 4 : RECHERCHES RAPIDES

CHAPITRE 1 : LES STRUCTURES LINÉAIRES

1) Les tableaux
2) Les pointeurs
3) Les listes chainées
4) Les piles
5) Les queues ou files
6) Applications aux matrices creuses (listes orthogonales)

TABLEAUX

Motivation



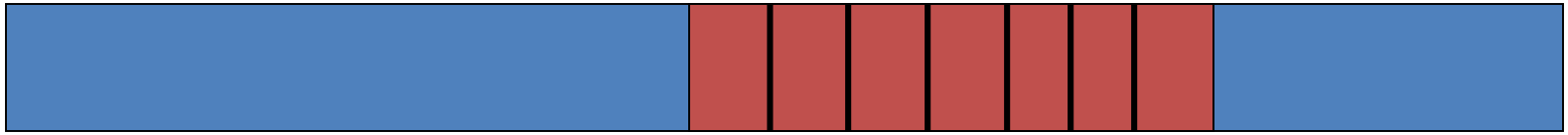
Structure de donnée:

- tableau a 2 dimension

Algorithmes:

- surtout I.A.

Les tableaux



Accès indexé (de 0 à n-1 pour un tableau de n éléments)

Stockage compact

Taille fixe, en général

Réajustement de taille coûteux en temps

Insertion d'élément onéreuse en temps.

QUESTION

- Qui n’est pas
  - Excellent
  - Très bien
  - Bien
  - Passable
  - **Tu peux reprendre tes siestes matinales du lundi**

POINTEURS - RAPPELS

Intuition

- ***Kernighan et Ritchie dans "programming in C"***

« ... Les pointeurs étaient mis dans le même sac que l'instruction ***goto*** *comme une excellente technique de formuler des programmes incompréhensibles. Ceci est certainement vrai si les pointeurs sont employés négligemment, et on peut facilement créer des pointeurs qui pointent 'n'importe où'. Avec une certaine discipline, les pointeurs peuvent aussi être utilisés pour programmer de façon claire et simple. C'est précisément cet aspect que nous voulons faire ressortir dans la suite. ... »*

Définition

- **Pointeur** : *une variable spéciale qui peut contenir l'**adresse** d'une autre variable.*

- La plupart des langages de programmation offrent la possibilité d'accéder aux données dans la mémoire de l'ordinateur à l'aide de ***pointeurs.***

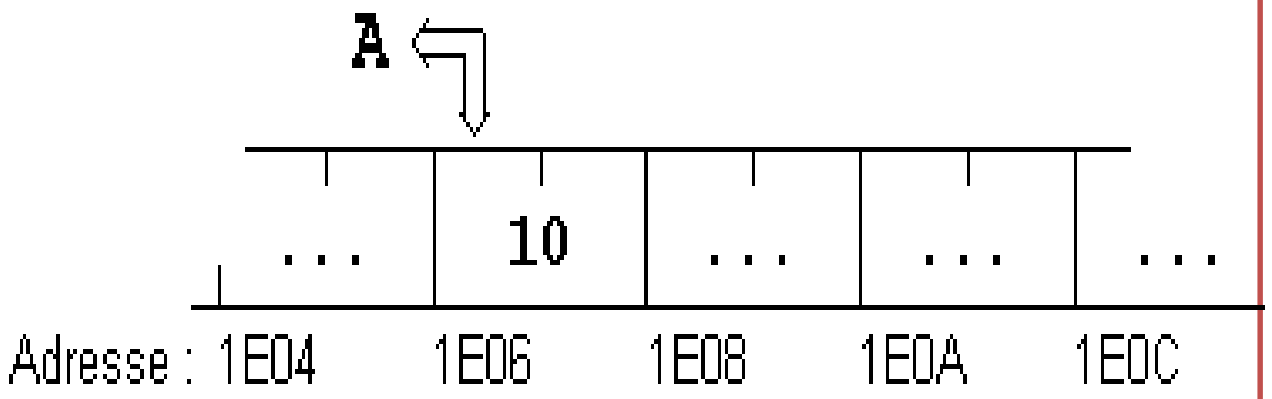
Cas du langage C

- Ils jouent un rôle primordial dans la définition de fonctions
  - passage des paramètres fait toujours par valeur
  - pointeurs sont le seul moyen de passe des variables par adresse.
  - traitement de tableaux et de chaînes de caractères dans des fonctions serait impossible sans l'utilisation de pointeurs

Adressage de variables (1)

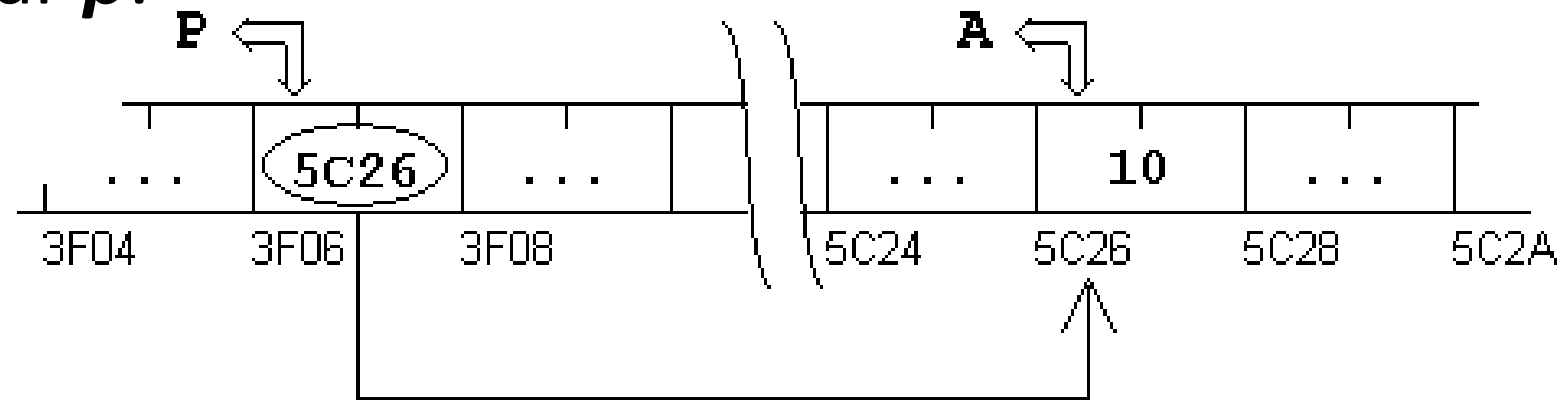
- **Adressage direct :** Accès au contenu d'une variable par le nom de la variable.
- ***Exemple :***

```
short A;
A = 10;
```



Adressage de variables (2)

- **Adressage indirect** : copier l'adresse d'une variable ***a*** (dont nous ne pouvons ou ne voulons pas utiliser) dans un pointeur ***p,*** on peut accéder à l'information de ***a*** en passant par ***p***.



Utilisation de Pointeurs

- Limité à un type de données
- Si ***p*** contient l'adresse de ***a*** alors ''***p*** pointe sur ***a***''
- Un pointeur peut pointer sur différentes adresses
- Le nom d'une variable reste lié à la même adresse

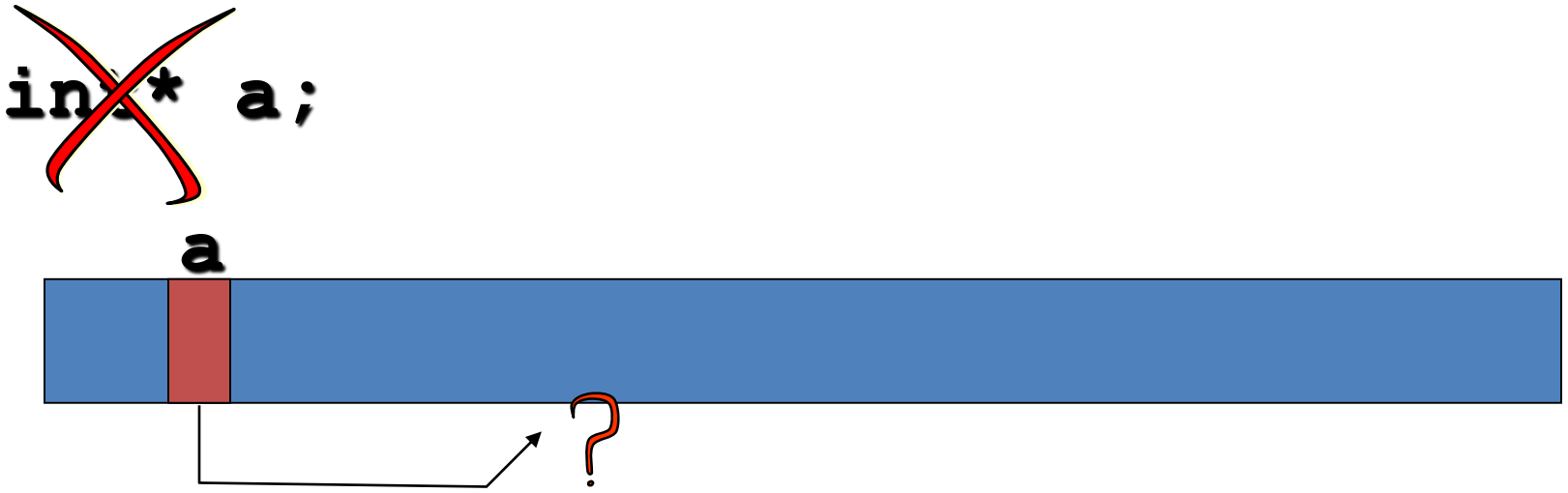
Opérateurs sur les Pointeurs

- Opérateurs de base
  - Adresse de : **&**
  - Contenu de : ***

Déclaration de pointeur (1)

- Déclaration:
  - L'instruction <***Type***> * <***nom_pointeur***>
    - Déclare un pointeur ***nom_pointeur***
    - Reçoit des adresses de variables de type <***Type***>
    - **Exemple** : *int * P*;

Déclaration de pointeur (2)

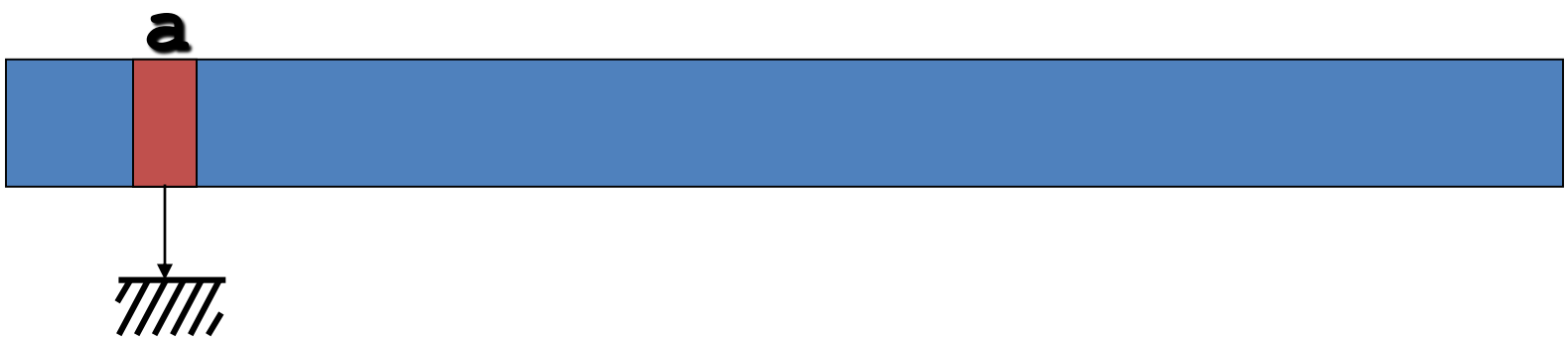


- Déclaration d'un pointeur vers un entier

Déclaration de pointeur (fin)

~~int* a;~~

```
int* a = NULL;
```



- ***Déclaration*** d'un pointeur vers un entier ***et initialisation*** à "NULL"

Utilisation de pointeurs (1)

- *<***nom_pointeur***> désigne le contenu de l'adresse référencée par le pointeur.
- Exemple
  - Int * P;
  - Int A=10, B;
  - P=&A;
  - **B=*P;** // B=10;
  - *P=90; // A = 90

Utilisation de pointeurs (2)

- &<***nom_var***> → l'adresse de la variable
  - S'applique à des variables et des tableaux.
  - Non à des constantes ou expressions

  - **Exemple :**
    - *int N;*
    - *printf("Entrez un nombre entier : ");*
    - *scanf("%d", &N);*

Opérations sur les pointeurs

- Opérations élémentaires sur pointeurs
  - Priorité de **&** et * : même priorité que les opérateurs unaires (!, ++, --).
  - Dans une expression ils sont tous évalués de droite à gauche.
  - Si ***P*** pointe sur ***X*** alors ***P** et **X** sont interchangeables

Incrémentation de pointeur

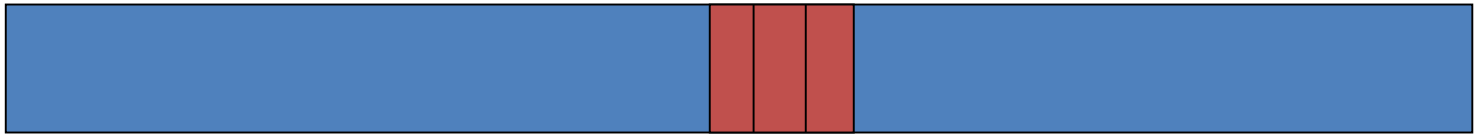
- Exemple : après l'instruction **P = &X**;
  - **Y = *P+1 Y=X+1;**
  - ***P = *P+10 X=X+10;**
  - **++*P ++X**
  - **(*P)++ X++ (parenthèses obligatoires)**

Pointeur *NULL*

- Zéro (0) est utilisé pour dire qu'un pointeur ne pointe ''nulle part''‘
- Exemple: *int * p; p = 0;*
- Les pointeurs sont des variables
  - *Int * p1, *p2; → p1=p2;*

Allocation d'espace (1)

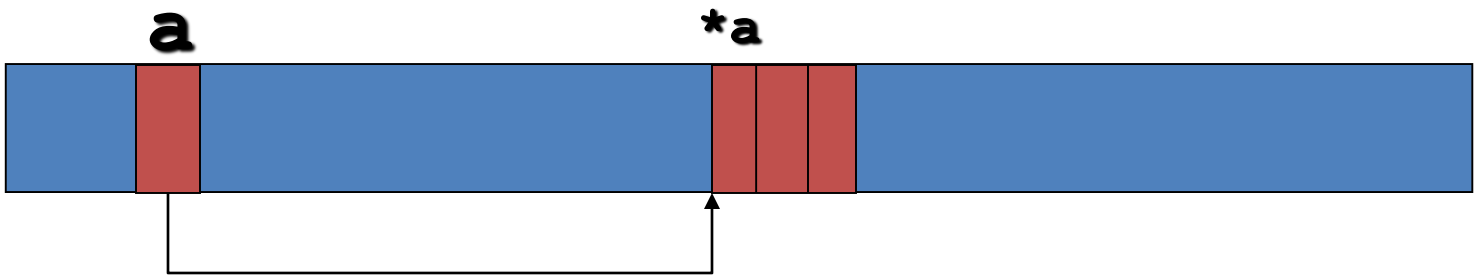
- `malloc(3*sizeof(int));`



- ***Allocation dynamique*** de place mémoire (pour 3 entiers)

Allocation d'espace (2)

- ~~int* a = malloc(3*sizeof(int));~~
- int* a = (int*)malloc(3*sizeof(int));

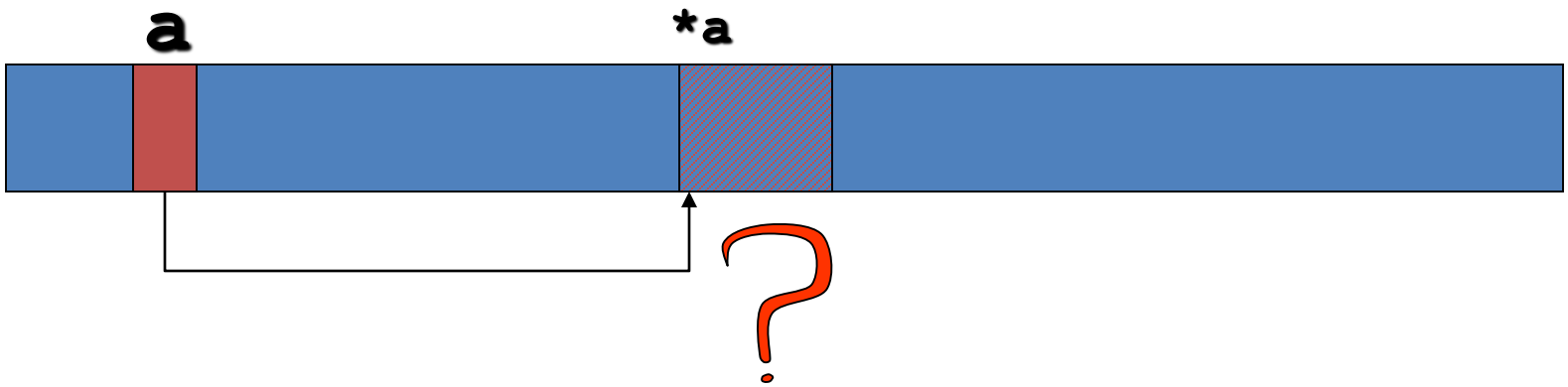


- ***Allocation dynamique*** de place mémoire (pour 3 entiers) et **assignation**

Libération de la mémoire

```
free(a);
```

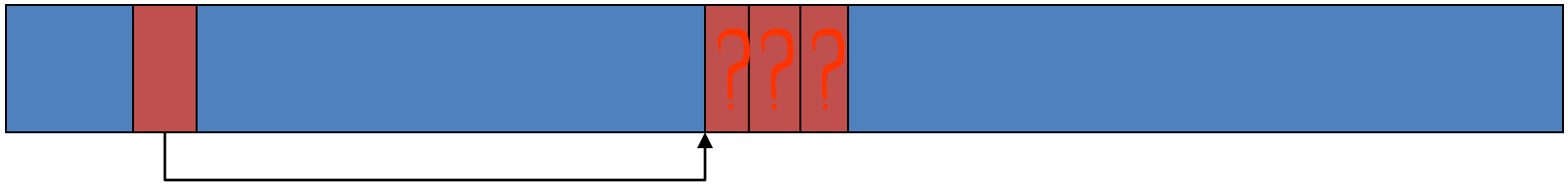
```
a = NULL;
```



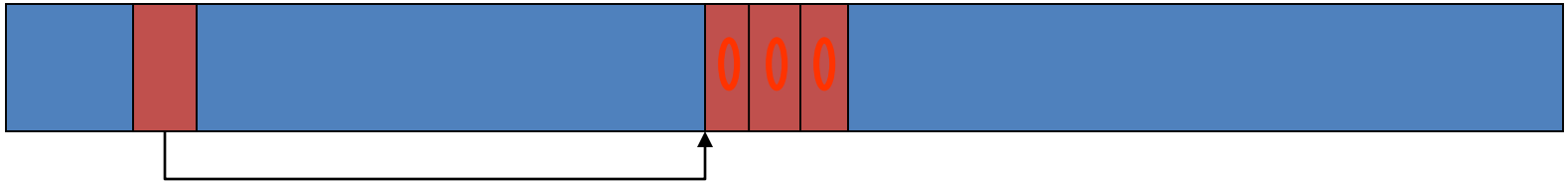
Désallocation dynamique

Réallocation de mémoire

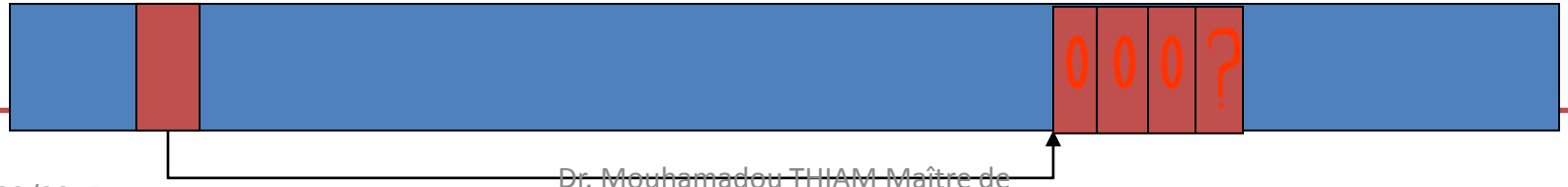
- `int* a = (int*)malloc(3*sizeof(int));`



- `int* a = (int*)calloc(3, sizeof(int));`



- `a = (int*)realloc(4*sizeof(int));`



Remarques

- Pb de malloc pour les tableaux: pas d'initialisation. → calloc

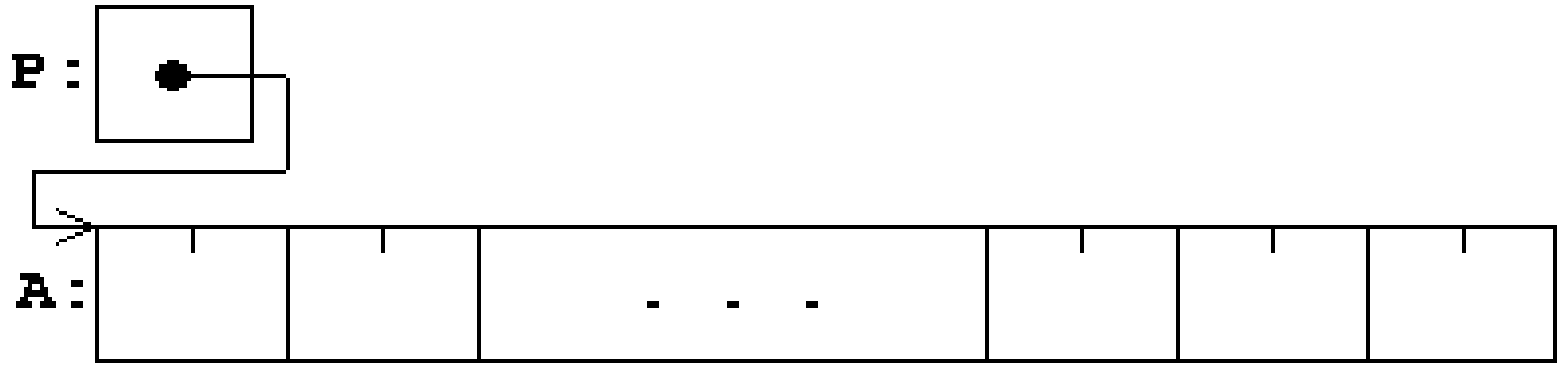
- Coût mémoire de realloc:
  - Alloue un nouveau bloc de mémoire
  - Copie les valeurs de l'ancien bloc dans le nouveau
  - Libère l'ancien bloc
  - Retourne l'adresse du nouveau bloc

Pointeurs et tableaux (1)

- Étroite relation entre pointeur et tableau
- Toute opération avec indices peut être faite à l'aide de pointeurs
- Adressage des composantes d'un tableau
  - nom Tableau = adresse de son premier élément
  - ***&tableau[0]*** et ***tableau*** sont une seule et même adresse
  - Nom tableau = pointeur constant sur son premier élément

Pointeurs et tableaux (2)

- ***Exemple*** :
  - **A** tableau de type **int** : **int A[10]**
  - **P** pointeur sur **int : int *P;**
  - **P = A;** équivalente à **P = &A[0];**



Pointeurs et tableaux (3)

- Si p pointe sur une composante du tableau
  - **P+1** pointe sur la composante suivante
  - **P+i** pointe sur la i-ième composante derrière **P**
  - **P- i** pointe sur la i-ième composante devant **P**.

- Ainsi, après l'instruction **P = A;** (A un tableau)
  - le pointeur P pointe sur A[0], et
  - ***(P+1)** désigne le contenu de A[1]
    ***(P+2)** désigne le contenu de A[2]
  - **…**
  - ***(P+i)** désigne le contenu de A[i]

Pointeurs et tableaux (fin)

- Différences entre un pointeur et un tableau
  - Un pointeur est une variable
    - P=A et P++ sont des opérations permises
  - Un nom de tableau est une constante
    - A=P ou A++ ne sont pas permises

Intérêts des pointeurs

- Gestion de l'espace mémoire en cours d'exécution
- Modifications de variables passées en paramètres de fonction
- Représentation de tableaux: accès direct et indexé
- Références croisées
- Fonctions virtuelles en programmation objet