Programmation

Pointeurs

Florent Devin

Ecole Internationale des Sciences du Traitement de l'Information

Introduction

Prédédemment

- Accès à une variable via son nom
  - `int_a = int_b;`
  - Recherche de la valeur de `int_b`
    - Connaître où est stocker `int_b` en mémoire
    - Aller chercher la valeur stockée dans cette case mémoire
    - Savoir où on doit stocker cette valeur (ici dans `int_a`)
    - Copier la valeur obtenue au bon endroit
  - Possibilité d'accéder différemment utilisation d'un type particulier pour mémoriser l'adresse d'une variable : pointeurs.

Définition

- Pointeur : variable qui mémorise une adresse mémoire
- Notion de référence
- Pointeur pointe une case mémoire
- Parfois appelé indirection
- Existence de pointeur constant : début d'un vecteur[1]

[1] cf diapositive 23

Variables et adresses

- Opérateur & : donne l'adresse d'une variable
- Utilisé par la fonction `scanf`
  - permet de spécifier où ranger la valeur lue
  - choix de l'adresse laissé au libre choix du compilateur
- Possibilité d'afficher une adresse en utilisant : %x, %X ou %p $\Leftrightarrow$ valeur hexadécimale

```
int_b = 5;
printf ("Valeur de b : %d\n",
        int_b);
printf ("Adresse de b : %x\n",
        &int_b);
```

Déclaration

Déclaration de pointeurs

- Pointeurs : référence tous types de données[2]
- Nécessité d'indiquer le type de donnée pointée
- Adresse : identique quelque soit le type
- Utile pour l'arithmétique
- Déclaration

```
type* nom;
type * nom;
type *nom;
```

- Définit un pointeur de type `type` ayant pour nom `nom`

[2] Exception faite des variables register, et champs de bits

Opérateur *

- Définition d'un nouvel opérateur : `*`
- Selon le contexte (lors de la déclaration)
- Peut être traduit par : `est un pointeur vers`

```
int* pint_a;
```

- `pint_a` est un pointeur vers un `int`
- Valeur de `pint_a` indéfinie lors de la définition
- **Besoin d'initialiser le pointeur avant utilisation**

Opérateur *

- Deuxième contexte (en utilisation)
- Peut être traduit par : `ce qui est pointé par`

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un
   entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

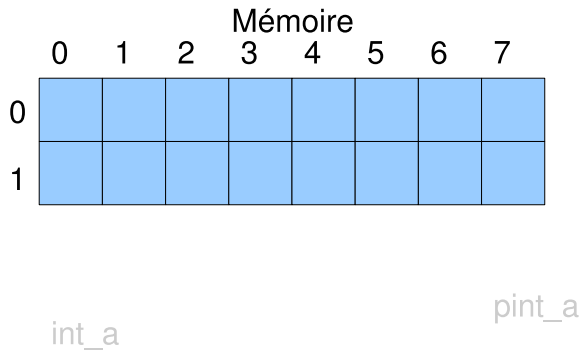


Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

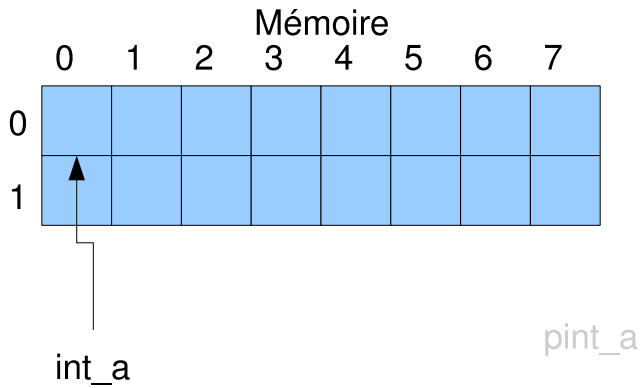


Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

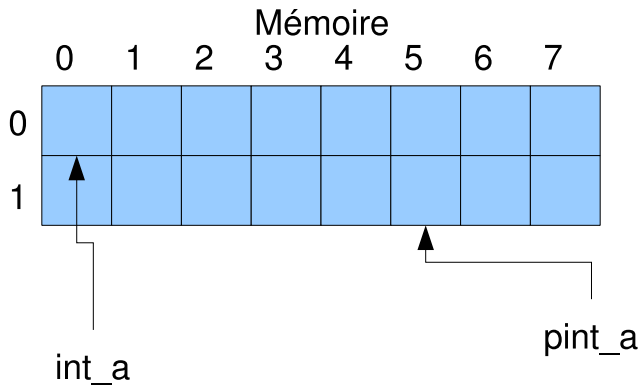


Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

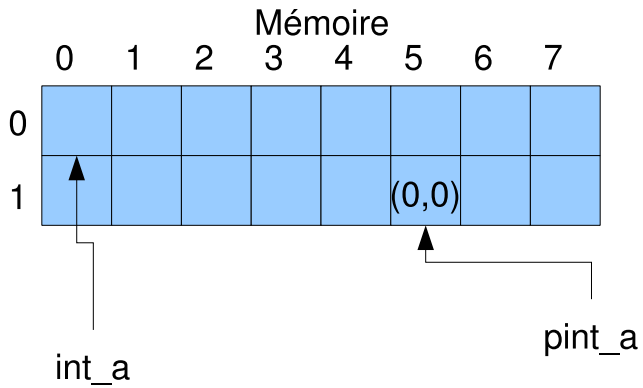


Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```

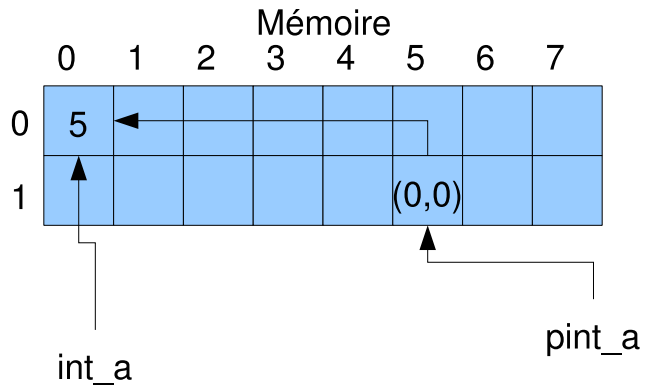
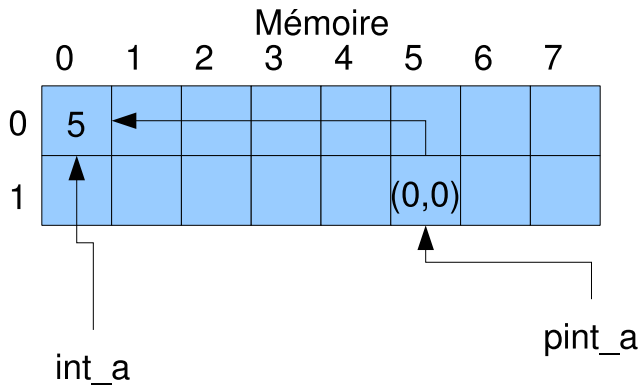


Schéma explicatif

```
int int_a;  // Variable
int* pint_a;// Pointeur sur un entier

// Affectation du pointeur
pint_a = &int_a;
// Initialisation de a
*pint_a = 5;
// Maintenant a vaut 5
```



Première utilisation

Échange de deux valeurs

- Sans pointeur : impossible à faire dans une fonction
- Avec pointeur
  - `void swap (int* a, int* b)`
  - Appel avec deux pointeurs qui contiennent les adresses dont le contenu est à échanger
  - Lors de l'appel de fonction : appel par valeur
  - Copie la valeur de l'adresse
    - Référence de la même case mémoire
  - Aussi appelé abusivement passage par référence

Arithmétique

Addition

```
int tint_tab[6]; //Tableau d'entier
int* pint_case;  //Pointeur sur un
   entier

//Affectation du pointeur à la
   première case du tableau
pint_case = &tint_tab[0];
// Affectation de tint_tab[1] à 3
pint_case+1 = 3;
```

- Ici l'adresse est augmenté de 1 * taille de la donnée pointée
- Attention au risque de "débordement"
- Possibilité d'incrémenter `pint_case++`

Parcours de tableau

Exemple

```
int tint_tab[6]; //Tableau d'entier
int* i; // Indice de boucle

for (i=&tint_tab[0]; i<&tint_tab[6];
     i++) {
        *i = 0;
}
```

Soustraction de pointeurs

```
int tint_tab[6]; //Tableau d'entier
int* pint_debut;  //Pointeur sur le
    début du tab
int* pint_fin;//Pointeur sur la fin
int int_taille; // Taille du tableau


//Affectation du pointeur à la
    première case du tableau
pint_debut = &tint_tab[0];
// Idem pour la fin
pint_fin = &tint_tab[5];

int_taille = pint_fin - pint_debut +
    1;
```

Autorisé ou non ?

- Opérations autorisées

  Addition **Incrémentation**
  Soustraction **Décrémentation**
  Comparaison **Affectation**

- Opérations interdites

  Multiplication **Division**
  Décalage **Opérations logiques**

- Opérations possibles

  Conversion de type (cast)

Tableaux statiques

- Parcours possible avec pointeur
- Besoin du début du tableau : `&tab[0]`
- Besoin de la taille : connue à l'avance (tableau statique)

Listing 1 – "Parcours d'un tableau avec un pointeur"

```
pint_p = &tint_tab[0];
for (i=0; i<N; i++) {
  (pint_p+i) = 0;
}
```

- Raccourcis syntaxique : `&tab[0]` $\Leftrightarrow$ `tab`

Tableaux dynamiques

- `int*` : permet de parcourir un tableau
- Peut être “vu” comme un tableau, à condition
  - d’avoir une adresse de départ
  - de connaître la taille du tableau
  - d’avoir “la jouissance” de la zone mémoire concernée
- Problématique :
  - Comment réaliser une allocation dynamique ?
  - Comment allouer un tableau sans connaître la taille à l’avance.

Allocation dynamique

- Principe
  - réservation d'une partie de la mémoire
  - utilisation de cette partie
  - libération de la zone
- Fonction : `void *malloc(size_t size)`
  - `size` : la taille souhaitée
  - Retourne l'adresse de début de zone allouée
  - `NULL` si allocation impossible

Exemple

Listing 2 – "Exemple d'allocation"

```
int* pint_tab; // Tableau dynamique
// Réservation de l'espace mémoire
pint_tab = malloc (15 * sizeof(int));
// SI l'allocation a échouée
if (pint_tab == NULL) {
        // On indique et on quitte
        fprintf (....);
        exit (-1);
}
.../...
//Libération de l'espace mémoire
free (tint_tab);
```