Filière : BTS-DSI
2020/2021

Les pointeurs en langage C

Par : Youssef Lachhab
youssef.lachhab.2020@gmail.com

PLAN

- Notion du pointeur
- Pointeurs et tableaux
- Tableaux dynamiques

Notion de pointeur

Introduction

En algorithmique les pointeurs jouent un rôle primordial :

- C'est le seul moyen de changer le contenu de variables déclarées dans d'autres fonctions,
- Très utilisés dans le traitement des structures de données suivantes :
  - Les tableaux,
  - les chaînes de caractères,
  - les listes chainées,
  - les arbres, les graphes,
  - …

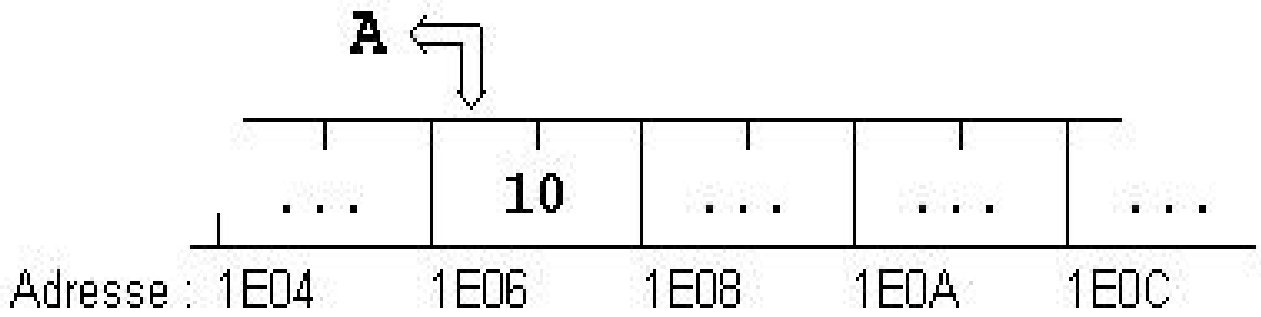
1) Adressage des variables

a) Adressage direct

Accès au contenu d'une variable par le nom de la variable.

Exemple :

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

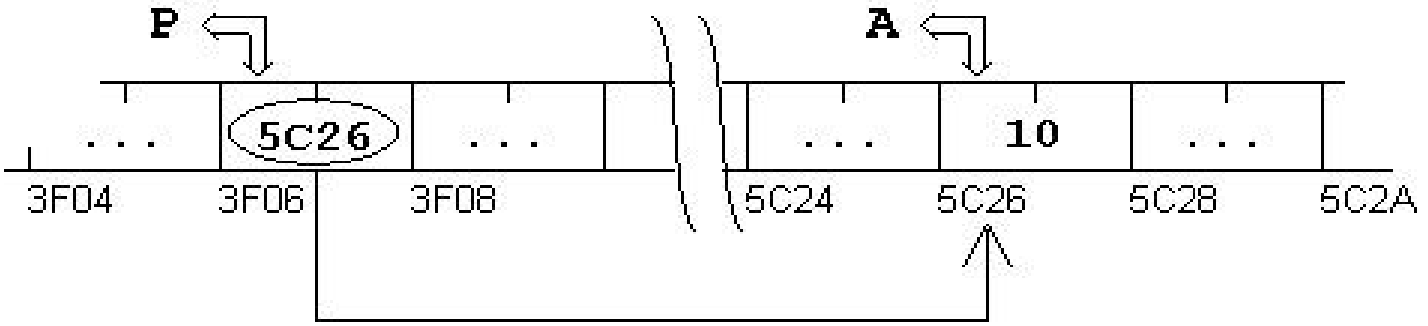


1) Adressage des variables

b) Adressage indirect

Accès au contenu d'une variable, en passant par un pointeur qui contient l'adresse de la variable.

Exemple :



P est une variable qui contient l'adresse de la variable **A.**

On dit que **A est un pointeur sur** la variable **A.**

2) Les pointeurs

Définition:

*Un **pointeur** est une variable spéciale qui peut contenir l'**adresse** d'une autre variable.*

Remarque:

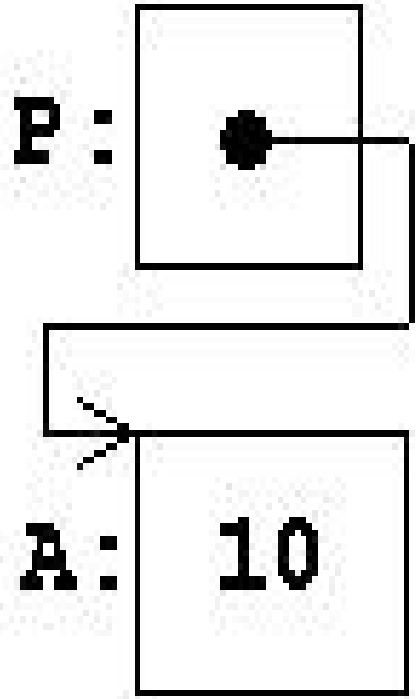
*Un **pointeur** est une variable qui peut 'pointer' sur différentes adresses. Alors que le **nom d'une variable** reste toujours lié à la même adresse.*

2) Les pointeurs

Représentation schématique :

Soit **P** un pointeur non initialisé et **A** une variable (du même type) contenant la valeur **10**,

nous pouvons illustrer le fait que ***'P pointe sur A'*** (**P = &A;**) par une flèche:



2) Les pointeurs

a) Les opérateurs de base

☙ *L'opérateur 'adresse de' : &*

&NomVariable : fournit l'adresse de la variable.

Il est déjà utilisé avec la fonction **scanf,** qui a besoin de l'adresse de ses arguments pour pouvoir leur attribuer de nouvelles valeurs.

***Exemple* :**

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

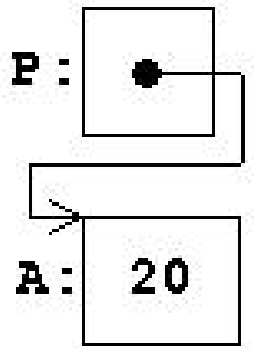
2) Les pointeurs

L'opérateur 'contenu de' : *

***NomPointeur** : désigne le contenu de l'adresse référencée par le pointeur.

Exemple :

```
A = 20;
P = &A;
```



Le contenu de la variable A peut être affiché de deux façons :

```
printf(" A = ", A);
```

Ou bien:

```
printf(" A = ", *P);
```

2) Les pointeurs

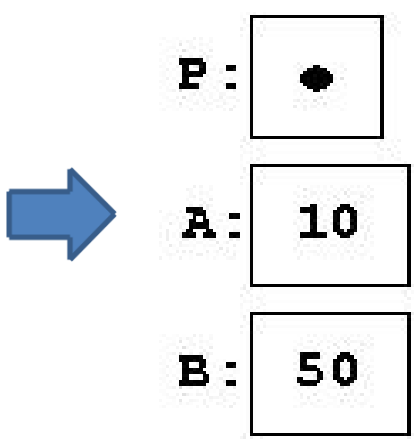
Déclaration d'un pointeur

Syntaxe :

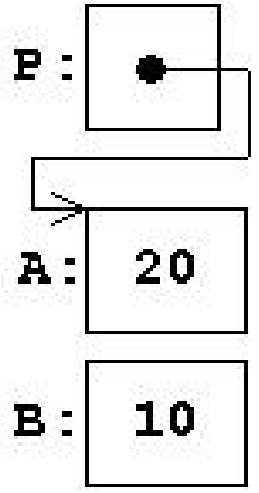
Type *NomPointeur

Exemple :

```
int *P;
int A=10;
int B=50;
```




```
P  = &A;
B  = *P ;
*P = 20;
```



Remarque :

Lors de la déclaration d'un pointeur en C, ce pointeur est lié explicitement à un seul type de données.

2) Les pointeurs

Exercice 1 :

1) Ecrire une fonction qui permet d'échanger le contenu de deux variables reçus en paramètres.
2) Tester la fonction en affichant le contenu des deux variables avant et après l'échange.

2) Les pointeurs

b) Les opérations élémentaires sur pointeurs

- ***Priorité de * et &*** :

- Les opérateurs * et & ont la même priorité que les autres opérateurs unaires (la négation **!**, l'incrémentation ++, la décrémentation --).

- Dans une même expression, les opérateurs unaires *, &, !, ++, -- sont évalués de droite à gauche.

2) Les pointeurs

- ***Priorité de * et &*** :

– Si un pointeur P pointe sur une variable X, alors *P peut être utilisé partout où on peut écrire X.

Exemple : Après l'instruction **P = &X;** les expressions suivantes, sont équivalentes:

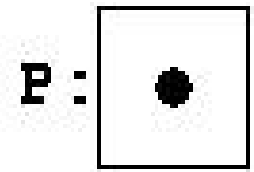
Y = *P+1	⇔	Y = X+1
*P = *P+10	⇔	X = X+10
*P += 2	⇔	X += 2
++*P	⇔	++X

2) Les pointeurs

- ***Le pointeur NULL*** :

La valeur numérique 0 (zéro) est utilisée pour indiquer qu'un pointeur ne pointe 'nulle part'.

```
int *P;
P = 0;
```



- ***Copie d'un pointeur :***

Soit P1 et P2 deux pointeurs sur int, alors l'affectation

P1 = P2; copie le contenu de P2 vers P1.

Alors, **P1 pointe alors sur le même objet que P2.**

3) Pointeurs et tableaux

En C, chaque opération avec des indices de tableaux peut aussi être exprimée à l'aide de pointeurs.

3) Pointeurs et tableaux

a) Adressage des composantes d'un tableau

Le nom d'un tableau représente l'adresse de son premier élément.

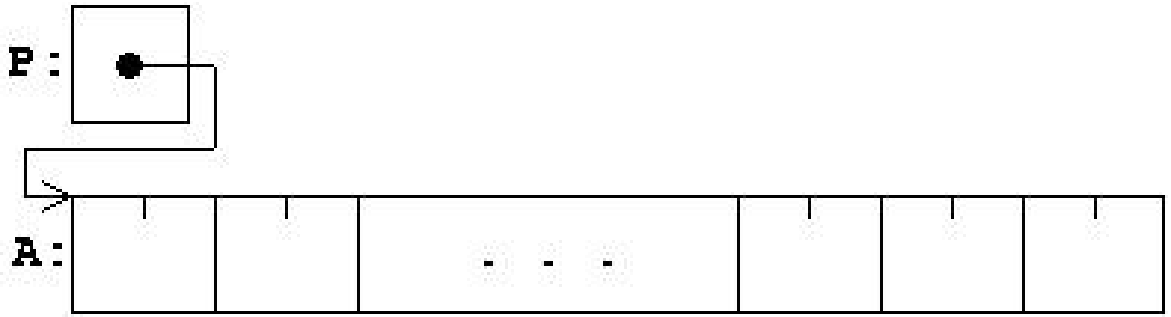
En d'autre termes : &tableau[0] et **tableau** sont une seule et même adresse.

- *Le nom d'un tableau est un **pointeur constant** sur le premier élément du tableau.*

3) Pointeurs et tableaux

Exemple :

```
int A[10];
int *P; P =
A;
```



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]

3) Pointeurs et tableaux

Si **P** pointe sur une composante quelconque d'un tableau, alors **P+1** pointe sur la composante suivante.

Plus généralement,

P - i pointe sur la i-ième composante derrière P. P

+ i pointe sur la i-ième composante devant P.

3) Pointeurs et tableaux

Remarque :

Un *pointeur* est une variable, donc des opérations comme **P = A** ou **P++** sont permises.

Le *nom d'un tableau* est une constante, donc des opérations comme **A = P** ou **A++** sont impossibles.

3) Pointeurs et tableaux

b) Arithmétique des pointeurs

Toutes les opérations avec les pointeurs tiennent compte automatiquement du type et de la grandeur des objets pointés.

3) Pointeurs et tableaux

Soient P1 et P2 deux pointeurs sur le même type de données.

Opération	Description
P1 = P2;	P1 sur le même objet que P2.

Si P pointe sur l'élément A[i] d'un tableau, alors

P+n	pointe sur A[i+n]
P-n	pointe sur A[i-n]
P++	P pointe sur A[i+1]
P- -	P pointe sur A[i-1]

3) Pointeurs et tableaux

Si P1 et P2 pointent *dans le même tableau* :

Opération	Description
P1-P2	fournit le nombre de composantes comprises entre P1 et P2.

Le résultat est :

négatif,	si P1 précède P2
zéro,	si P1 = P2
positif,	si P2 précède P1
indéfini,	si P1 et P2 ne pointent pas dans le même tableau

3) Pointeurs et tableaux

Comparaison de deux pointeurs

On peut comparer deux pointeurs par

<, >, <=, >=, ==, !=.

La comparaison de deux pointeurs qui pointent *dans le même tableau* est équivalente à la comparaison des indices correspondants.

3) Pointeurs et tableaux

Exercice 2 :

1) Ecrire la fonction ***adresseMin(float *T , int taille)*** qui prend en paramètre un tableau des réels, puis il retourne l'adresse de la valeur minimale du tableau.

2) Ecrire la fonction ***Tri(float *T, int taille)*** qui réalise le tri du tableau des réels reçu en paramètre, en utilisant le principe du tri par sélection (cette fonction doit utiliser la fonction précédente).

3) Ecrire un programme principal qui teste ces fonctions.

4) Allocation dynamique de mémoire

4) Allocation dynamique de mémoire

a) Déclaration statique de données

La déclaration des données permet de réserver la mémoire automatiquement. Nous parlons alors de la ***déclaration statique*** des variables.

Exemple :

```
float A, B, C;          /* réservation de 12 octets */
short D[10][20];        /* réservation de 400 octets */
```

Cette technique oblige la connaissance de la taille des données à réserver avant l'exécution.

4) Allocation dynamique de mémoire

b) Allocation dynamique

L'***allocation dynamique*** permet de faire la réservation de la mémoire *pendant l'exécution du programme*.

- ***La fonction malloc***

La fonction **malloc** de la bibliothèque <***stdlib***> nous aide à localiser et à réserver de la mémoire au cours de l'exécution d'un programme.

4) Allocation dynamique de mémoire

Exemple :

```
char *T;
T=malloc(4000);
```

Fournit l'adresse d'un bloc de **4000 octets libres** et l'affecte à **T**.

S'il n'y a plus assez de mémoire, **T** obtient la valeur zéro.

4) Allocation dynamique de mémoire

b) La fonction sizeof

Fournit la taille d'une type de données , d'une variable ou d'une constante:

sizeof (var) : fournit la grandeur de la variable **var**
sizeof (const) : fournit la grandeur de la constante **const**
sizeof (type) : fournit la grandeur pour un objet du type **type**

4) Allocation dynamique de mémoire

Exemple :

Nous voulons réserver de la mémoire pour N valeurs du type **int** ; la valeur de N est lue au clavier:

```
int N;
int *PNum;
printf("Introduire le nombre de valeurs :");
scanf("%d", &N);
PNum = malloc(N*sizeof(int));
```

4) Allocation dynamique de mémoire

b) La fonction free()

Si nous n'avons plus besoin d'un bloc de mémoire que nous avons réservé à l'aide de ***malloc***, alors nous pouvons le libérer à l'aide de la fonction ***free*** de la bibliothèque <***stdlib***>.

Syntaxe :

free (Pointeur);

Elle libère le bloc de mémoire désigné par le <Pointeur>;

Elle n'a pas d'effet si le pointeur a la valeur zéro.

4) Allocation dynamique de mémoire

Remarques :

- Il est conseillé d'affecter la valeur zéro au pointeur immédiatement après avoir libérer l'espace mémoire.

-Si nous ne libérons pas explicitement la mémoire à l'aide de **free**, alors elle est libérée automatiquement à la fin du programme.

4) Allocation dynamique de mémoire

Exercice 3 :

Ecrire un programme principal qui permet de tester les fonctions de l'exercice 2 (AdresseMin et Tri) sur un tableau alloué et initialisé d'une manière dynamique.