INPT
LA GRANDE ÉCOLE DES TÉLÉCOMMUNICATIONS ET DES TECHNOLOGIES DE L'INFORMATION

Filière Smart-ICT

Algorithmique et Programmation C

Mr N.EL FADDOULI
elfaddouli@emi.ac.ma
nfaddouli@gmail.com

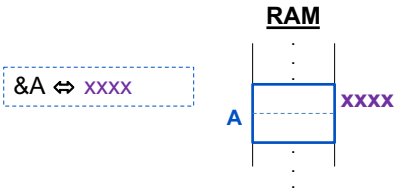
Année Universitaire:2024/2025

1

109

Le langage C: Les pointeurs - Définition

- Chaque variable possède un **nom**, un **type**, une **valeur** et une **zone mémoire** dont la taille dépend du type de la variable (≥ 1 Octet).
- Cette zone mémoire, réservée pour la variable, est identifiée par une adresse qui est en fait l'adresse de son premier octet. Cette adresse est celle de la variable.
- L'opérateur **&** permet d'avoir l'adresse d'une variable: **&***nom_variable*



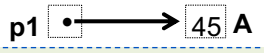
- Un **pointeur** est une **variable** qui stocke l'**adresse mémoire** d'une autre variable.
- Il Il permet de manipuler directement la mémoire. En utilisant des pointeurs, on peut accéder et modifier les données stockées à des adresses mémoire spécifiques.

Le langage C: Les pointeurs – Déclaration et utilisation (1/2)

- Pour déclarer un **pointeur** qui est une **variable**, on ajoute un astérisque (*) devant son nom, suivi du **type** de la **valeur de la zone mémoire** sur laquelle il va pointer.

Type ***nom_pointeur***;

Exemple:
```
int   *p1;   /* pointeur sur int: contiendra l'adresse d'une variable de type int */
char  *p2;   /* pointeur sur char: contiendra l'adresse d'une variable de type char */
float *p3;   /* pointeur sur float: contiendra l'adresse d'une variable de type float */
int A;
p1 = &A;
*p1 = 45;
```



- *p1 pointe sur sur A*
- *La zone mémoire de A est pointée par p1*

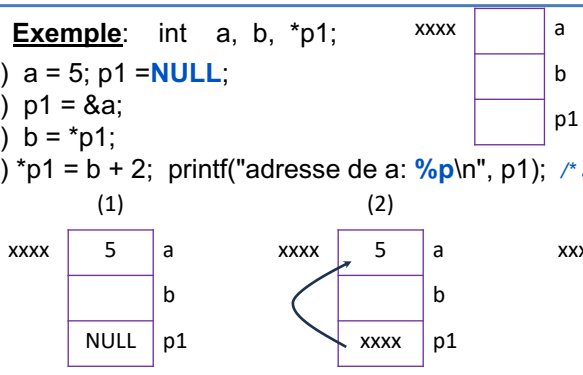
- ***&variable*** représente l'adresse
- ****pointeur*** représente le **contenu** de la zone mémoire pointée par ***pointeur***.
- Un pointeur ne reçoit que l'adresse d'une variable ou un autre pointeur de même type.

```
int a; double b;
int *p = &b;      /* Faux : type de p ≠ type de b*/
int *q = &a;      /* Correct: type de q = type de a */
double *r = p;    /* Faux: type de r ≠ type de p */
```

Le langage C: Les pointeurs – Déclaration et utilisation (2/2)

☞ **Exemple**: int a, b, *p1;

(1) a = 5; p1 =**NULL**;
(2) p1 = &a;
(3) b = *p1;
(4) *p1 = b + 2; printf("adresse de a: **%p**\n", p1); */* affichage de l'@, exemple: 0x16fa035c4*/*



☞ La valeur **NULL** indique que le pointeur qui ne pointe vers aucune adresse valide. Elle est utilisée pour indiquer que le pointeur ne pointe pas encore vers une variable ou une zone mémoire valide.

☞ Si on a un pointeur qui ne pointe pas vers une zone valide (*déjà réservée*) et on essaie d'accéder au contenu pointé, on aura un bug du programme à cause de l'erreur "**segmentation fault**".

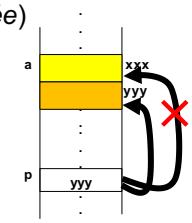
Par exemple: int *p1; printf("%d\n", *p1);

Le langage C: Les pointeurs – Incrémentation/Décrémentation

☞ Pour déplacer un pointeur d'une adresse à une autre, on utilise Les opérateurs ++ et -- qui permettent de modifier la valeur du pointeur.

☞ L'opérateur **++** modifie le pointeur pour qu'il pointe vers l'élément (*l'emplacement suivant*) selon le type du pointeur (*type de la valeur de la zone pointée*)

Exemple: int a, *p ;
p=&a ;
p++; */* ⇔ **p = p+1** ; */*



☞ **Remarque** : **(*p)++** incrément de **1** la valeur de la zone pointée.

☞ Si un entier est codé sur 4 octets *(sizeof(int)==4)*, on aura **yyy = xxx+4**. Ainsi p++ permet d'incrémenter p de 4 càd p++ ⇔ p = p+4

☞ De même, l'opérateur **--** modifier le pointeur pour qu'il pointe vers l'élément précédent.

☞ Conclusion: ***p++ ⇔ p= p+1 ⇔ p += 1 ⇨ p ← p + sizeof****(type_de_p)*

Le langage C: Allocation mémoire pour un pointeur (1/2)

- L'espace réservé pour une variable est conservé durant toute l'exécution du bloc dans lequel cette variable est déclarée. On a une **réservation statique**.
- Un pointeur peut contenir l'adresse mémoire d'une variable déjà déclarée (**réservation statique**)
- Un pointeur peut contenir l'adresse mémoire d'une zone **réservée dynamiquement** pendant 'exécution du programme.

Exemple : main() { .....

..... */* réserver un espace mémoire pour stocker N entier */*

..... */* utiliser cet espace réservé dynamiquement*/*

..... /** libérer cet espace mémoire */*

.....

}

Le langage C: Allocation mémoire pour un pointeur (2/2)

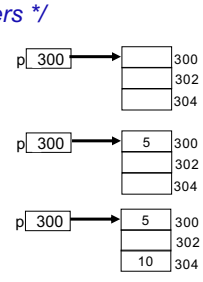
- On peut allouer (réserver) **dynamiquement** un espace mémoire et stocker son adresse dans un pointeur en utilisant la fonction **malloc** de la bibliothèque <**malloc.h**> (ou <**stdlib.h**>)
- La syntaxe d'appel de cette fonction est la suivante:

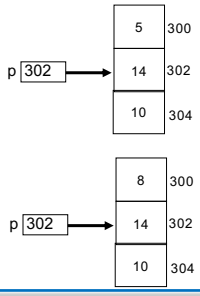
pointeur **= (*type_du_pointeur**) malloc (*taille_mémoire*);**

- Si l'allocation mémoire est impossible *(mémoire insuffisante)*, malloc retourne **NULL**.

Exemple:

```
int  *p; /* réserver un espace pour 3 entiers */
(0)  p =(int*) malloc (3*sizeof(int));
(1)  *p = 5;
(2)  *(p+2) = 10;
(3)  *(++p) = 14;
(4)  p--;   *(p++) = *p +3;
```





Le langage C: Libération mémoire

☞ Pour libérer un espace mémoire alloué dynamiquement, on fait appel à la fonction **free** comme suit: **free(*pointeur*);**

où ***pointeur*** contient l'adresse d'une zone mémoire préalablement obtenue par un appel à **malloc** ou **calloc.**

☞ ***Exemple:***

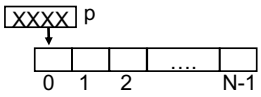
```
main()  {  int *p, N, i, T[100];
    printf(" Donner le nombre d'entiers:");
    scanf("%d",&N);    float  L[N];
    p = (int *) malloc(N * sizeof(int));
    for(i=0;  i<N;  i++)
        { printf("Donnez l'entier d'indice %d:", i);
          scanf("%d", p+i);
        }
    free(p);
    .... }
```

L'espace mémoire de T et L ne sera libéré qu'à la fin du programme

On obtient un tableau dynamique qu'on peut redimensionner avec realloc ou libérer avec free

p+i ⇔ adresse de la case d'indice i
L'adresse de la case obtenue après i déplacements à partir de p.

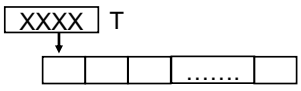
&p[i]

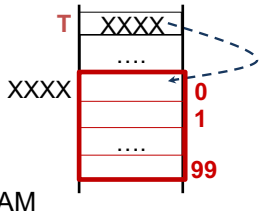


Le langage C: Tableaux & pointeurs (1/2)

☞ Un tableau **statique** est un **pointeur constant** sur le premier élément.

☞ Exemple: int T[100];





☞ Les éléments sont stockés à la suite dans la RAM

☞ Pour accéder à un élément du tableau: ***Tableau* [*indice*] ⇔ *(*Tableau* + *indice*)**

☞ Exemple: int T[5], i ;

```
T[0]=10;     /* ⇔  *T  = 10      */
T[1]=34;     /*     *(T+1)  = 34  */
*(T+i )=20;  /*     T[ i ] = 20   */
printf("%d\n",*(T+1));
```

☞ Remarques:

- ♦ scanf("%d", &T[i]); ⇔ scanf("%d",T+i);
- ♦ Instructions ***impossibles***: **T++**; **T=T+ i** ; ….

Le langage C: Tableaux & pointeurs (2/2)

☞ Exemple: int T[50], ***p**, i, N ;

```
printf("Nombre d'éléments:");   scanf("%d",&N);
/*  Les trois boucles suivantes sont équivalentes */
for (i=0; i<N; i++)
        { printf("T[%d]=",i);   scanf("%d",&T[i] ); }
for(p=T; p<T+N; p++)
        { printf("T[%d]=",p-T);   scanf("%d", p ); }
for( i= 0; i<N; i++)
        { printf("T[%d]=",i);   scanf("%d",T+i ); }
/*  Les trois boucles suivantes sont équivalentes */
for (i=0; i<N; i++)    { printf("%d   ",  T[ i ] ); }
for(p=T; p<T+N; p++)   { printf("%d   ", *p);  }
for( i= 0; i<N; i++)     { printf("%d   ", *(T+i)  ); }
```


Le langage C: Pointeurs et chaînes de caractères (1/2)

☞ Une chaîne peut être manipulée à travers un pointeur comme tout type de tableau puisqu'elle est un tableau de type char.

1) char *p ;

   p = (**char***) **malloc** (*taille_chaine*) ;

2) char *q = "***texte***" ; /* Initialisation ⇨ allocation automatique */

☞ **Exemple**:

```
char *p;
p = (char*) malloc (20);
p ="toto"; /* ⚠ incorrect */
strcpy(p, "toto") ;
*p = 'B' ;          /* ⇔ p[0] = 'B'  */
*(p+2)='L' ;        /* ⇔ p[2] = 'L'  */
scanf("%s", p) ;    /*  lecture d'une chaîne */
printf("%s\n", p) ;  /*  affichage d'une chaîne */
printf("%c\n",*p) ;  /*  affichage de p[0] */
```

Le langage C: Pointeurs génériques

- **void** est un type en langage C qui est dit "**incomplet**" par ce que sa **taille n'est pas calculable**.
  - ⇨ *on ne peut pas déclarer une variable de type **void***
- On peut déclarer un **pointeur générique** de type **void**
  - ⇨ *il peut référencer (pointer sur) n'importe quel **type** de variable.*
  - ⇨ *on peut affecter n'importe quel pointeur à un pointeur générique (et inversement)*
- Exemple:

```
int main(){  int a=4;  double b=5.6;
void *p;  double *r;
p = &a;   printf("Int : %d\n", *(int*)p);
p = &b;   printf("Double : %lf\n", *(double*)p);
r = p ;
printf("Double (direct) : %lf\n", *r);
p = r ;
return 1;}
```

```
Int : 4
Double : 5.600000
Double (direct) : 5.600000
```


Le langage C: Pointeurs et chaînes de caractères (2/2)

- **Exercices:**

1- Inverser une chaîne en utilisant un pointeur sur des caractères. *(utilisez une seule chaîne)*

2- Copier une chaîne, saisie au clavier, dans une autre *(utilisez 2 pointeurs)*.

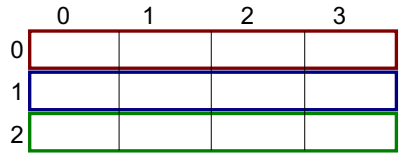
3. Utiliser un tableau statique de **pointeurs** de type **char** pour mémoriser N chaînes saisies.

Le langage C: Pointeurs et tableaux de deux dimensions (1/3)

- Un tableau de deux dimensions (*matrice*) est un **tableau de tableaux** càd chaque élément est à son tour un tableau dont chaque case contient une valeur (entier, ...)

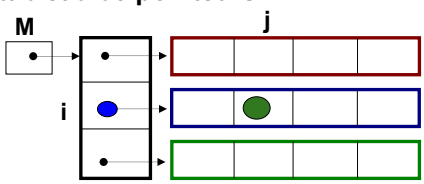
Exemple: matrice **M** de 3 lignes et 4 colonnes

int **M**[3][4] ;



- Sachant qu'un **tableau** est **un pointeur sur le premier élément**

⇨ On peut conclure qu'une matrice est un **tableau** de **pointeurs**.



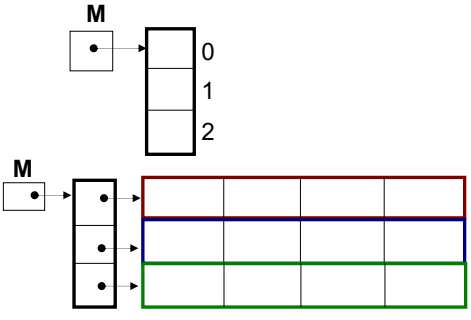
- Conclusion: Une matrice ≡ **pointeur sur pointeur**

M[i][j] ⇔ ***(*(M+i) + j)**

Le langage C: Pointeurs et tableaux de deux dimensions (2/3)

- Pour faire l'allocation dynamique de mémoire pour une matrice, on doit d'abord faire l'allocation mémoire pour les pointeurs sur les lignes.
- On fera ensuite l'allocation mémoire pour chaque ligne.
- **Exemple**: Pour une matrice **M** de 3 lignes et 4 colonnes

```
int  ** M , i,  j ;
M = (int **) malloc ( 3 * sizeof (int*)  );
for ( i = 0; i<3 ; i++)
        M[ i ] = (int *) malloc (4 * sizeof (int) );
        /* Rappel:  *(M+i)   ⇔   M[ i ]  */
for (i=0; i<3 ; i++)
    for (j=0; j<4 ; j++)
      { printf("Case(%d,%d):",i, j);  scanf("%d", &M[ i ][ j ] );  }
            /* Rappel:    *(M+i) + j   ⇔   &M[ i ][ j ]    */
```



Le langage C: Pointeurs et tableaux de deux dimensions (3/3)

☞ **Exemple**: Pour une matrice **M** de L lignes et C colonnes

```
int **M, L, C, i, j;
....
.....
M = (int **) malloc( L * sizeof(int *));
for(i=0; i< L ; i++)
        M[ i ] = (int *) malloc (C * sizeof (int));
.....
.....
.....
for(i=0; i<L; i++)
        free( M[ i ] ) ; /* ou bien:  free(*(M +i) )   */
free(M);
.....
```

Déterminer L et C par lecture ou calcul

Utilisation de la matrice M

Libération de l'espace mémoire réservé pour la matrice M