Module: Algorithmique Et Programmation C II

AÏCHA EL GOLLI
aicha.elgolli@essai.ucar.tn
Janvier-Mai 2023

ECOLE SUPÉRIEURE DE LA STATISTIQUE
ET DE L'ANALYSE DE L'INFORMATION

ORGANISATION DU COURS

✓Module de 42h : **<u>14 semaines</u>** x3h :

- 1h30 cours (Mme El Golli)+ 1h30 TP (Mme Mansour) P3 et P4
- Travail et cours sur classroom

✓ Contenu des chapitres (P3):

1. *Chapitre I :* Introduction aux pointeurs
2. Chapitre II: Les structures
3. Chapitre III: la récursivité

✓Contenu des chapitres (P4):

1. *Chapitre IV :* Les listes chaînées
2. Chapitre V: Les files et piles
3. Chapitre VI: les fichiers

✓ Modalités d'évaluation :

Note de CC: note de TP (évaluations au niveau des TP) - Bonus/malus sur l'implication (présence / pertinence des interactions) en cours et DS en fin de P3 (35%)

✓ Examen final sur les concepts vus en cours/TPs (65%) ==>fin P4

Chapitre 1: Introduction aux pointeurs

AÏCHA EL GOLLI
aicha.elgolli@essai.ucar.tn

L'importance des pointeurs en C

- La plupart des langages de programmation offrent la possibilité d'accéder aux données dans la mémoire de l'ordinateur à l'aide de ***pointeurs***, c.-à-d. à l'aide de variables auxquelles on peut attribuer les ***adresses*** *d'autres variables*.
- En C, les pointeurs jouent un rôle primordial dans la définition de fonctions:
  - Comme le passage des paramètres en C se fait toujours par la valeur, les pointeurs sont le seul moyen de changer le contenu de variables déclarées dans d'autres fonctions. Ainsi le traitement de tableaux et de chaînes de caractères dans des fonctions serait impossible sans l'utilisation de pointeurs

Définition: Pointeur

- 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 simple de ce type ou l'adresse d'une composante d'un tableau de ce type.
- Si un pointeur P contient l'adresse d'une variable A, on dit que **'P pointe sur A'**.
- Syntaxe de déclaration d'un pointeur:

Algorithmique	Langage C
<NomPointeur>: Pointeur sur <Type>;	<Type> *<NomPointeur>;
déclare un pointeur <NomPointeur> qui peut recevoir des adresses de variables du type <Type>	
p: pointeur sur entier;	int *p;

peut être interprétée comme suit:

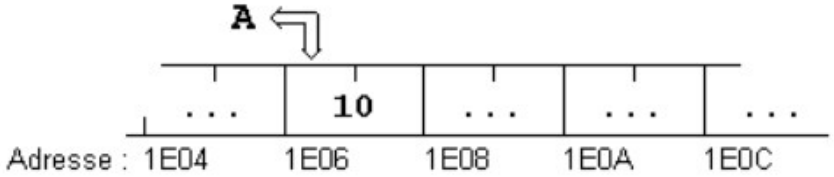
*P est de type int ou P est un pointeur sur int ou P peut contenir l'adresse d'une variable du type int

Adressages des variables

Il existe deux modes d'adressage principaux:

- **Adressage direct:** La valeur d'une variable se trouve à un endroit spécifique dans la RAM de l'ordinateur. Le nom de la variable nous permet alors d'accéder *directement* à cette valeur.

Exemple: int A=10;



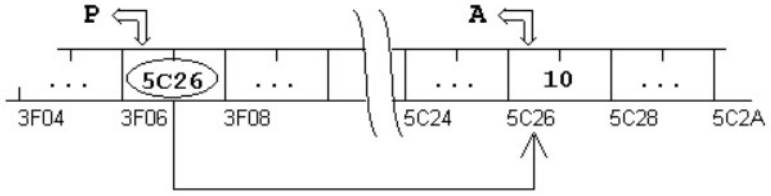
- **Adressage indirect:** Si nous ne voulons ou ne pouvons pas utiliser le nom d'une variable A, *nous pouvons copier l'adresse de cette variable dans une variable spéciale P, appelée* ***pointeur*****.** Ensuite, nous pouvons retrouver l'information de la variable A en passant par le pointeur P.

Exemple:

Soit A une variable contenant la valeur 10 et P un pointeur qui contient l'adresse de A. En mémoire, A et P peuvent se présenter comme suit:

int A=10;

int *P=&A;



Adressages des variables

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

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

Opérateurs de base

- Lors du travail avec des pointeurs, nous avons besoin:
  - d'un opérateur « adresse de »: & pour obtenir l'adresse d'une variable.
  - d'un opérateur « contenu de »: * pour accéder au contenu d'une adresse.

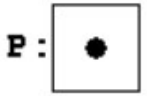
- &<NomVariable> fournit l'adresse de la variable <NomVariable>
- L'opérateur & nous est déjà familier par la fonction scanf, qui a besoin de l'adresse de ses arguments pour pouvoir leur attribuer de nouvelles valeurs.
- Exemple:

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

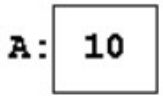
- Attention !: l'opérateur & peut seulement être appliqué à des objets qui se trouvent dans la mémoire interne, c.-à-d. à des variables et des tableaux. Il ne peut pas être appliqué à des constantes ou des expressions.

Représentation schématique

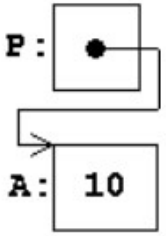
- Soit P un pointeur non initialisé: int *P;



- et A une variable (du même type) contenant la valeur 10 : int A=10;



- Alors l'instruction: P = &A;
- affecte l'adresse de la variable A à la variable P. Nous pouvons illustrer le fait que 'P pointe sur A' par une flèche:



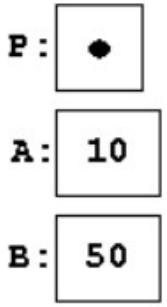
- Remarques

** Les constantes pointeurs NULL ou 0 (en C) indiquent l'absence d'adresse. Donc, par exemple en C, l'affectation p = NULL, veut dire que p ne pointe aucune variable.*

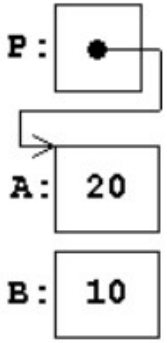
*** Il ne faut jamais utiliser l'indirection (* en C) avec un pointeur ne contenant pas l'adresse d'une variable, il y aura alors une erreur de segmentation**

Opérateurs de base

- L'opérateur 'contenu de' : *
- *<NomPointeur> désigne le contenu de l'adresse référencée par le pointeur <NomPointeur>
- Exemple: int A=10, B=50, *P;



- Après les instructions:
  P = &A;
  B = *P;
  *P = 20;



- Lors de la déclaration d'un pointeur en C, ce pointeur est lié explicitement à un type de données. Ainsi, la variable P déclarée comme pointeur sur int ne peut pas recevoir l'adresse d'une variable d'un autre type que int.

Opérations élémentaires sur pointeurs

- En travaillant avec des pointeurs, nous devons observer les règles suivantes:
  - 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.
  - 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
- (*P)++ ↔ X++ Attention, dans le dernier cas, les parenthèses sont nécessaires: Comme les opérateurs unaires * et ++ sont évalués de droite à gauche, sans les parenthèses le pointeur P serait incrémenté, non pas l'objet sur lequel P pointe.

Le pointeur NULL

- On peut uniquement affecter des adresses à un pointeur.
- Seule exception: La valeur numérique 0 (zéro) est utilisée pour indiquer qu'un pointeur ne pointe 'nulle part'.

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

Finalement, les pointeurs sont aussi des variables et peuvent être utilisés comme telles. Soit P1 et P2 deux pointeurs sur int, alors l'affectation `P1 = P2;` copie le contenu de P2 vers P1. P1 pointe alors sur le même objet que P2.

```
int A;
int *P;
P = &A;
```

A désigne le contenu de A **&A désigne l'adresse de A** **P désigne l'adresse de A** ***P désigne le contenu de A**	**En outre:** **&P désigne l'adresse du pointeur P** ***A est illégal (puisque A n'est pas un pointeur)**

Exercice

```
main()
   {
      int A = 1;
      int B = 2;
      int C = 3;
      int *P1, *P2;
      P1=&A;
      P2=&C;
      *P1=(*P2)++;
      P1=P2;
      P2=&B;
      *P1-=*P2;
      ++*P2;
      *P1*=*P2;
      A=++*P2**P1;
      P1=&A;
      *P2=*P1/=*P2;
      return 0;
   }
```

Corrigé

```
main()
   {
      int A = 1;
      int B = 2;
      int C = 3;
      int *P1, *P2;
      P1=&A;
     P2=&C;
      *P1=(*P2)++;
      P1=P2;
      P2=&B;
      *P1-=*P2;
      ++*P2;
      *P1*=*P2;
      A=++*P2**P1;
      P1=&A;
      *P2=*P1/=*P2;
      return 0;
   }
```

	A	B	C	P1	P2
Init.	1	2	3	indéfinie	indéfinie
P1=&A	1	2	3	&A	indéfinie
P2=&C	1	2	3	&A	&C
*P1=(*P2)++	3	2	4	&A	&C
P1=P2	3	2	4	&C	&C
P2=&B	3	2	4	&C	&B
*P1-=*P2	3	2	2	&C	&B
++*P2	3	3	2	&C	&B
P1=*P2	3	3	6	&C	&B
A=++*P2**P1	24	4	6	&C	&B
P1=&A	24	4	6	&A	&B
*P2=*P1/=*P2	6	6	6	&A	&B

Pointeurs comme argument d'une fonction

Nous avons vu que les paramètres formels d'une fonction en C peuvent être modifiés sans que le paramètre effectif ne soit modifié. Il est pourtant souvent utile de pouvoir modifier un ou plusieurs paramètres effectifs depuis une fonction.

```
Procédure increm( d/r val: entier)
Début
Val++;
Fin
Algorithme changement
var: entier;
Début
var←0;
increm(var);
Afficher(" var = ", var);
Fin
```

Ce programme affiche:
var = 1

```
#include <stdio.h>
void increm (int *val)
{
  (*val)++ ;   /* les opérateurs unaires * et
++ sont évalués de droite à gauche, sans les
parenthèses le pointeur val serait incrémenté,
non pas l'objet sur lequel val pointe.
                */
}
void main ()
{
  int var = 0 ;
  increm (&var) ;
  printf ("var = %d\n", var) ;
}
```

Pointeurs et tableaux

- L'identificateur d'un tableau employé tout seul sans l'indice entre [ ] signifie l'adresse mémoire du premier élément du tableau. De plus cet identificateur est considéré comme étant de type pointeur sur le type correspondant aux éléments du tableau.
- Voici quelques notations équivalentes:

t + 1 ↔ &t[1]

t[i] ↔ *(t + i)

t[0] ↔ *t

Pointeurs et tableaux

Voici trois manières d'initialiser un tableau avec la valeur 123:

```
#include <stdio.h>

#define NB_ELEM 4

void main ()
{
  int tabl[NB_ELEM] ;
  int i ;
  int *p ;

  /* méthode 1 */
  for (i =  0; i < NB_ELEM; i++)
    tabl[i] = 123 ;

  /* méthode 2 */
  for (i =  0; i < NB_ELEM; i++)
    *(tabl + i) = 123 ;

  /* méthode 3 */
  p = tabl ;
  for (i =  0; i < NB_ELEM; i++)
    *p++ = 123 ;
}
```

L'instruction *p++ dans la troisième variante mérite une explication: Après l'instruction p = tabl ; p pointe au début du tableau, c-à-d. il pointe le premier élément du tableau. L'instruction *p++ est équivalente aux deux instructions suivantes:

*p = 123 ; p++ ;

Pointeurs et tableaux

En déclarant un tableau A de type **int** et un pointeur P sur **int**,

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

l'instruction:

```
P = A;
```
est équivalente à
```
P = &A[0];
```

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

Exemple: Soit A un tableau contenant des éléments du type float et P un pointeur sur float:

```
float A[20], X;
float *P;
```

Après les instructions,

```
P = A;
X = *(P+9);
```

X contient la valeur du 10-ième élément de A, (c.-à-d. celle de A[9]).

Une donnée du type float ayant besoin de 4 octets, le compilateur obtient l'adresse P+9 en ajoutant 9 * 4 = 36 octets à l'adresse dans P.

Comme A, représente l'adresse de A[0],

*(A+1) désigne le contenu de A[1]

*(A+2) désigne le contenu de A[2]

...

*(A+i) désigne le contenu de A[i]

Pointeurs et tableaux

Attention

Il existe toujours une différence essentielle entre un pointeur et le nom d'un tableau:

- 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.

Pointeurs et tableaux

Exemple

Les deux programmes suivants copient les éléments positifs d'un tableau T dans un deuxième tableau POS.

Formalisme tableau

```
main()
{
 int T[10] = {-3, 4, 0, -7, 3, 8, 0, -1, 4, -9};
 int POS[10];
 int I,J;  /* indices courants dans T et POS */
 for (J=0,I=0 ; I<10 ; I++)
     if (T[I]>0)
        {
         POS[J] = T[I];
         J++;
        }
 return 0;}
```

Pointeurs et tableaux

Nous pouvons remplacer systématiquement la notation **`tableau[I]`** par **`*(tableau + I)`**, ce qui conduit à ce programme:

Formalisme pointeur

```
main()
{
 int T[10] = {-3, 4, 0, -7, 3, 8, 0, -1, 4, -9};
 int POS[10];
 int I,J;  /* indices courants dans T et POS */
 for (J=0,I=0 ; I<10 ; I++)
     if (*(T+I)>0)
        {
         *(POS+J) = *(T+I);
         J++;
        }
 return 0;}
```

Les variables et leur utilisation

`int A;` déclare une ***variable simple*** du type **int**

A désigne le contenu de A
&A désigne l'adresse de A

`int B[];` déclare un ***tableau*** d'éléments du type **int**

B désigne l'adresse de la première composante de B. (Cette adresse est toujours constante)
B[i] désigne le contenu de la composante i du tableau
&B[i] désigne l'adresse de la composante i du tableau

en utilisant le formalisme pointeur:

B+i désigne l'adresse de la composante i du tableau
*(B+i) désigne le contenu de la composante i du tableau

Les variables et leur utilisation

`int *P;` déclare un pointeur sur des éléments du type int.

P peut pointer sur des variables simples du type int ou sur les composantes d'un tableau du type int.
P désigne l'adresse contenue dans P (Cette adresse est variable)
*P désigne le contenu de l'adresse dans P

Si P pointe dans un tableau, alors

P désigne l'adresse de la première composante
P+i désigne l'adresse de la i-ième composante derrière P
*(P+i) désigne le contenu de la i-ième composante derrière P

Soit P un pointeur qui 'pointe' sur un tableau A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

a) *P+2 => la valeur 14
b) *(P+2) => la valeur 34
c) A+3 => l'adresse de la composante A[3]
d) P+(*P-10) => l'adresse de la composante A[2]
e) *(P+*(P+8)-A[7]) => la valeur 23

Pointeurs et chaînes de caractères

De la même façon qu'un pointeur sur **int** peut contenir l'adresse d'un nombre isolé ou d'une composante d'un tableau, un pointeur sur **char** peut pointer sur un caractère isolé ou sur les éléments d'un tableau de caractères. Un pointeur sur **char** peut en plus contenir *l'adresse d'une chaîne de caractères constante* et il peut même être *initialisé* avec une telle adresse.

Pointeurs et chaînes de caractères

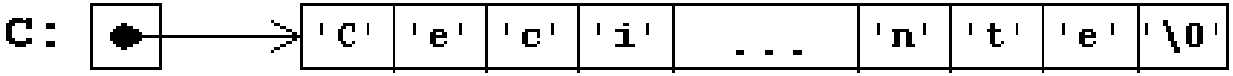
- Pointeurs sur char et chaînes de caractères constantes

Affectation

a) On peut attribuer l'adresse d'une chaîne de caractères constante à un pointeur sur char:

Exemple

```
char *C;
C = "Ceci est une chaîne de caractères constante";
```



Nous pouvons lire cette chaîne constante (p.ex: pour l'afficher), mais il n'est pas recommandé de la modifier.

Pointeurs et chaînes de caractères

- Initialisation

b) Un pointeur sur char peut être initialisé lors de la déclaration si on lui affecte l'adresse d'une chaîne de caractères constante:

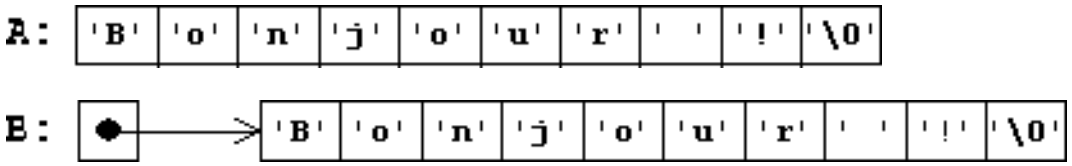
```
char *B = "Bonjour !";
```

Attention : Il existe une différence importante entre les deux déclarations:

```
char A[] = "Bonjour !";   /* un tableau  */
char *B  = "Bonjour !";   /* un pointeur */
```

A est un tableau qui a exactement la grandeur pour contenir la chaîne de caractères et la terminaison '\0'. Les caractères de la chaîne peuvent être changés, mais le nom A va toujours pointer sur la même adresse en mémoire.

B est un pointeur qui est initialisé de façon à ce qu'il pointe sur une chaîne de caractères constante stockée quelque part en mémoire. Le pointeur peut être modifié et pointer sur autre chose. La chaîne constante peut être lue, copiée ou affichée, mais pas modifiée



Pointeurs et chaînes de caractères

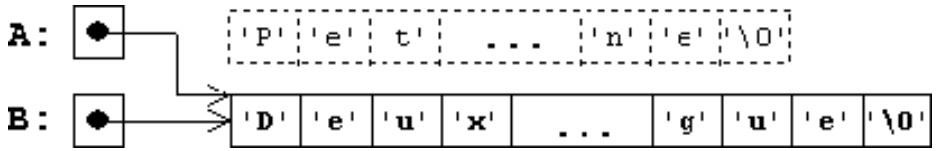
- Modification

c) Si nous affectons une nouvelle valeur à un pointeur sur une chaîne de caractères constante, nous risquons de perdre la chaîne constante. D'autre part, un pointeur sur char a l'avantage de pouvoir pointer sur des chaînes de n'importe quelle longueur:

Exemple

```
char *A = "Petite chaîne";
char *B = "Deuxième chaîne un peu plus longue";
A = B;
```

Maintenant A et B pointent sur la même chaîne; la "Petite chaîne" est perdue:

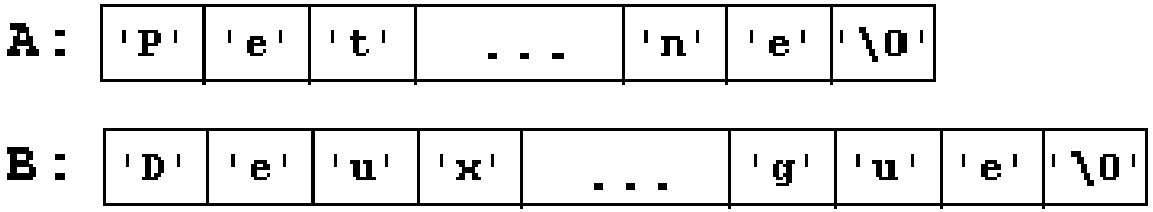


Attention : Les affectations discutées ci-dessus ne peuvent pas être effectuées avec des tableaux de caractères.

Pointeurs et chaînes de caractères

- ***Exemple***

```
char A[45] = "Petite chaîne";
char B[45] = "Deuxième chaîne un peu plus longue";
char C[30];
A = B;                /* IMPOSSIBLE -> ERREUR !!! */
C = "Bonjour !";      /* IMPOSSIBLE -> ERREUR !!! */
```



Dans cet exemple, nous essayons de copier l'adresse de B dans A, respectivement l'adresse de la chaîne constante dans C. Ces opérations sont impossibles et illégales parce que ***l'adresse représentée par le nom d'un tableau reste toujours constante***.

Pour changer le contenu d'un tableau, nous devons changer les composantes du tableau l'une après l'autre (p.ex. dans une boucle) ou déléguer cette charge à une fonction de <*stdio*> ou <*string*>.

Pointeurs et chaînes de caractères

```
void main()
{char  *p;
char C[]="hello tout le monde";
p=C;
printf("%s\n", C);
  for(;*p!=0;p++)
  {
      if(*p!=' ') *p=*p+1;
  }
  printf("%s\n", C);
}
```