Université Echahid Hamma Lakhdar d'El-oued
Département d'informatique
Niveau: 2 ème LMD Informatique
Module: Algorithmiques et structures de données
(Les pointeurs)

La syntaxe pour déclarer un pointeur est la suivante.

```
type *NomDuPpointeur;
```

Par exemple, si nous souhaitons créer un pointeur sur int (c'est-à-dire un pointeur pouvant stocker l'adresse d'un objet de type int) et que nous voulons le nommer ptr, nous devons écrire ceci.

```
int *ptr;
int * ptr;
int* ptr;
```

Initialisation
Un pointeur, comme une variable, ne possède pas de valeur par défaut, il est donc important de l'initialiser pour éviter d'éventuels problèmes. Pour ce faire, il est nécessaire de recourir à l'opérateur d'adressage (ou de référencement): qui permet d'obtenir l'adresse d'un objet. Ce dernier se place devant l'objet dont l'adresse souhaite être obtenue. Par exemple comme ceci.

```
int a = 10;
int *p;
p = a;
```

```
include < stdio.h >
int main(void)
{
int a = 10;
int *p;
p = a;
printf("a = % d", a);
printf("*p = %d", *p);
printf("&a = %d", &a);
printf("p = %d", p);
printf("&p = %d", &p);
return 0;
}
```

Résultats

```
**********************************************************
a = 10
*p = 10
&a = 6487580
p = 6487580
&p = 6487568
**********************************************************
```

NB:
Faites bien attention à ne pas mélanger différents types de pointeurs! Un pointeur sur int n'est pas le même qu'un pointeur sur long ou qu'un pointeur sur double. De même, n'affectez l'adresse d'un objet qu'à un pointeur du même type.

```
int a;
double b;
int *p = &b; /* Faux */
int *q = &a; /* Correct */
double *r = p; /* Faux */
```

La constante NUL
Afin de clarifier un peu les codes sources, il existe une constante définie dans l'en-tête ¡stddef.h¿: NULL. Celle-ci peut être utilisée partout où un pointeur nul est attendu.

```
int *p = NULL; /* Un pointeur nul */
```

Pointeur de Pointeur:

```
include < stdio.h >
int main(void)

int a = 10;
int *pa = &a;
int **pp = &pa;
printf("la valeur de a avant de modifier le poiteur a = %d", a);
*pa=20;
printf("a = %d", a);
printf("*pa = %d", *pa);
printf("pa = %d", pa);
printf("**pp = %d", **pp);
printf("*pp = %d", *pp);
printf("pp = %d", pp);
return 0;
```

Résultats

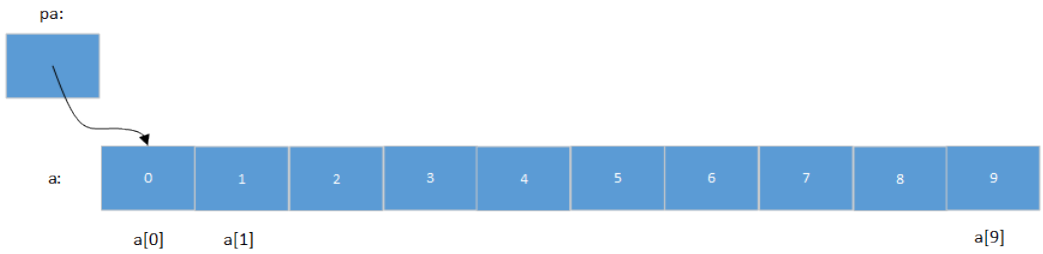
****
la valeur de "a" avant de modifier le poiteur "pa": a = 10
la valeur de "a" après de modifier le poiteur "pa": a = 20
*pa = 20
pa = 6487572
**pp = 20
*pp = 6487572
pp = 6487560
****

Pointeurs et tableaux

- Chaque opération avec les indices des tableuax peut être aussi exprimée par les pointeurs.
- si on affecte un nom du tableau à un poiteur, ce dernier point sur l'adresse du ptremier élement du tableau (&tableau et tableau signifient la même adresse, celle du premier élement du tableau) **??**.

  int a[10];
  int * Pa;
  Pa = a// est équivalent à Pa=& a[0].

- Si P pointe sur un élément quelconque du tableau, alors P+i pointe sur



l'l'élément suivant.

P + i pointe sur la i ième composante à droite de *P.
P - i pointe sur la i ième composante à gauche de *P.
soit P=Tab, alors:

> *(P+1) désigne le contenu Tab[1];
> *(P+2) désigne le contenu Tab[2];
> .....
> *(P+i) désigne le contenu Tab[i];

- Incrémentation et décremettation d'un pointeur.

> Si P pointe sur la composante Tab[i], alors après l'instruction:
> P++; P pointe sur Tab[i+1].
> P+=n; P pointe sur Tab[i+n].
> P−−; P pointe sur Tab[i-1].
> p− =n; p pointe sur Tab[i-n].

Note: Si P pointe sur la composante Tab[i], alors: P+i ne prend pas la valeur du ième octet après P mais prend l'adresse de la ième composante après P tout dépend du type du pointeur tel que chaque type s'addresse selon un nombre définit d'octets pour chaque composante.

Si T un tableau de type floats et P est un pointeur du même type:

float T[20], x;
float *p;
après les instructions:
P=T;
x=*(P+5);

x contient la valeur du sixième élément du T, T[5].
Une donnée de type float ayant besoin de 4 octets, le compilateur obtient l'adresse P + 5 en ajoutant 5 * 4 = 20 octets à l'adresse dans P.

Soustraction et comparaison de deux pointeurs:

- Soustraction de deux pointeurs:
  soit P1 et P2 deux pointeurs qui pointent sur le même tableau.
  - (P1-P2) donne le nombre d'éléments entre P1 et P2 dans le tableau mais pas le nombre d'octets entre les deux adresses.

le résultat de (P1-P2)est:
- Négatif, si P1 précède P2.
- Zéro, P1=P2.
- Positif si P2 précède P1.
- Indéfinit si P1 et P2 ne pointe pas dans le même tableau.

- Comparaison de deux pointeurs:
  On peut comparer deux pointeurs par les opérateurs: >, <, <=, >=, ==, ! =.
  si les deux pointeurs à comparer sont dans le même tablaux, alors la comparaison se fait entre les deux indices correspondants, sinon la comparaison se fait entre les deux adresses (positions) correspondantes en mémoires.

Résumé

Soit Tab est un tableau de type quelconque:
- Tab définit l'adresse de Tab[0];
- Tab+i définit l'adresse de Tab[i];
- *(Tab+i) définit le contenu de Tab[i];

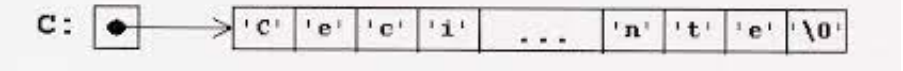
Si P=Tab alors:
- P pointe sur l'élément Tab[0];
- P+i pointe sur l'élément Tab[i];
- *(P+i) définit le contenu de Tab[i];

Pointeurs et chaines de caractères

- Tout ce qui a été mentionné concernant les pointeurs et les tableaux reste vrai pour les pointeurs et les chaînes de caractères.
- En plus, un pointeur vers une variable de type char peut aussi contenir l'adresse d'une chaîne de caractères constante et peut même être initialisé avec une telle adresse.

Exemple

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




```
char *ch="Bonsoir!";
```

```
char *ch1="bonjour";
char *ch2="bonjour tout le monde";
ch1=ch2;// la valeur de ch1 est perdue dans ce cas et pointe sur la deuxième chaine.
```

Note: Un pointeur sur char a l'avantage de pouvoir pointer sur des chaînes de n'importe quelle longueur.

Pointeurs et Tableaux à deux dimensions

Soit Mat un tableaux de entier tel que:

int Mat[4][6];

$$\text{Mat}= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Si un pointeur P=Mat, alors P pointe sur le tableau Mat[0]

Mat[0]= $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$
Si P=Mat+i , alors P pointe sur la i ème ligne de la matrice
la valeur (*(Mat+2))[3]=16.

Problème: comment faire pour accéder à une composante d'une matrice en utilisant seilement les pointeurs.
Solution: si Mat est une matrice et P est un pointeur sur cette matrice c-à-d P=Mat alors, pour accéder à un élémént de la matrice il faut convertir P en un type int de la manère suivante:

int Mat[4][6];

$$\text{Mat}= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

```
int *P=Mat;
int *P1;
//convertir le type de pointeur forcément.
P1=(int*)P;
ou bien
P1=(int*)Mat;
```

NB: P1 dans ce cas pointe sur l'élément Mat[0][0], et la matrice dans ce cas peut être traitée comme un tableau unidimensionnel de taille 24.
Exemple: calcul de la somme des élément d'une matrice à l'aide des pointeurs

int Mat[4][6];

$$\text{Mat}= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

```
int *P;
int i, somme;
P=(int*)Mat;
somme=0;
for(i=0;i<24;i++)
{
somme+=*(P+i);
}
```

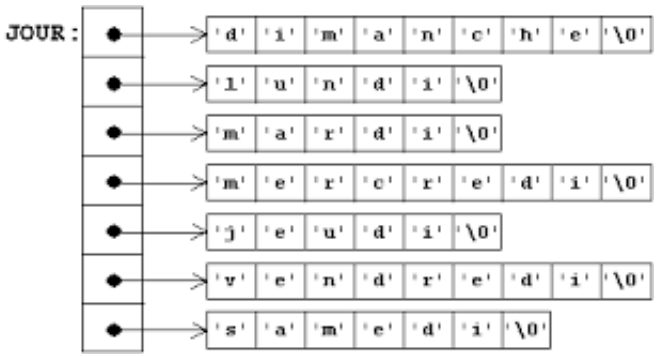
Tableau de pointeurs

Exemple: int *Tab[10]; // Tab est un tableau de 10 pointeurs sur des valeurs de type int.

char *Mois[ ]={"janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet", "aout", "septembre", "octobre", "novembre", "décembre"};

Mois est un tableau de 12 pointeurs de type chaine de caractères.
Pour afficher la première lettre de chaque mois :

```
int i;
for(i=0;i<12;i++)
Printf("%c", *Mois[i]);
```



Pour afficher le contenu de la cellule, il est nécessaire d'utiliser un pointeur sur le tableau de pointeurs

```
include < stdio.h >
int main(void)
char *ville[]="eloued","alger","oran","media";
char **pville;
pville =(char**)&ville;
while(*pville!=")
{
printf("%s", *pville);
pville++;
}
return 0;
```

Résultat:

****
eloued
alger
oran
media
****

Le pointeur générique Le type void * est le type pointeur générique, c'est à dire capable de pointer vers n'importe quel type d'objet.

```
void *pg;
int *p1, *p2;
int n=4;
p1 = &n;
pg=p1; /* pointeur générique = pointeur */
p2=pg; /* pointeur = pointeur générique */
```

Cependant cette affectation est interdite si les pointeurs son de types différents sans faire une conversion forcée de type, par exemple:

```
char *pc;
int *p;
p=pc; //interdite
p= (int *)pc; //correcte
pc=p; //interdite
pc = (char *)p; //correcte
```

Résumé

- Un pointeur est une variable dont le contenu est une adresse;
- L'opérateur d'adressage permet de récupérer l'adresse d'une variable;
- Un pointeur d'un type peut uniquement contenir l'adresse d'un objet du même type;
- Un pointeur nul contient une adresse invalide qui dépend de votre système d'exploitation;
- Un pointeur nul est obtenu en convertissant zéro vers un type pointeur ou en recourant à la macroconstante NULL.
- L'opérateur d'indirection (*) permet d'accéder à l'objet référencé par un pointeur;
- En cas de retour d'un pointeur par une fonction, il est impératif de s'assurer que l'objet référencé existe toujours;
- Le type void permet de construire des pointeurs génériques;
- L'indicateur de conversion %p peut être utilisé pour afficher une adresse. Une conversion vers le type void * est pour cela nécessaire.