



## *Exos Langage C*

# Les pointeurs

Exercices sur les pointeurs		
Auteur	Version - Date	Nom du fichier
G.VALET	Version 1.3 - Nov 2010	exo-langageC-pointeurs.docx
Quelques exercices sur les pointeurs. Pour effectuer correctement ces exercices, se référer au "cours de programmation en C de M. LEBRET" et notamment à la partie sur les "pointeurs", les "structures" et les "fonctions"		

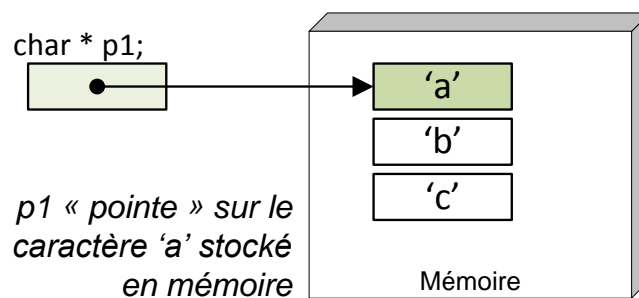
## A. Sommaire


A. SOMMAIRE.....	2
B. PRESENTATION DES POINTEURS .....	3
<i>B.1. Définition .....</i>	<i>3</i>
<i>B.2. Utilisation.....</i>	<i>3</i>
<i>B.3. Arithmétique des pointeurs .....</i>	<i>4</i>
C. QUELQUES QUESTIONS .....	5
D. ENONCE DES EXERCICES .....	6
<i>D.1. Déclaration et utilisation .....</i>	<i>6</i>
<i>D.2. Afficher une chaîne de caractères .....</i>	<i>6</i>
<i>D.3. Passage par adresse .....</i>	<i>6</i>
<i>D.4. Traitement de chaîne.....</i>	<i>7</i>
<i>D.5. Calculer la taille d'une chaîne.....</i>	<i>7</i>
<i>D.6. Conversion binaire-décimal .....</i>	<i>8</i>
a. Objectif.....	8
b. Algorithme .....	8
c. Poids fort.....	8
d. Vérification d'usage .....	9
e. Mise en œuvre .....	9

## B. Présentation des pointeurs

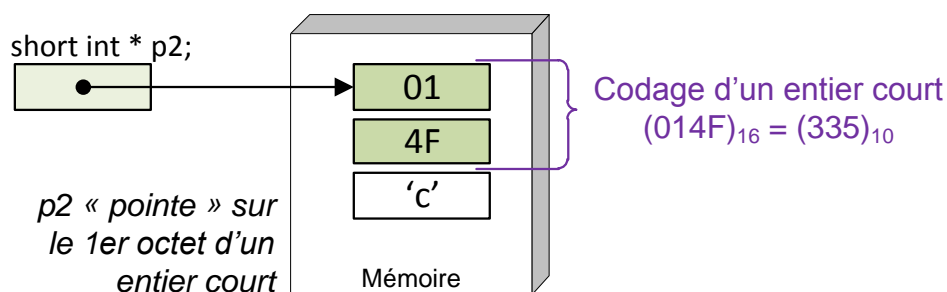
### B.1. Définition

Un pointeur est une variable qui désigne un emplacement mémoire (une adresse) occupée par une donnée dont le type est défini lors de la déclaration. Lorsqu'on définit un pointeur, on précise également le type de donnée que contient l'emplacement pointé :



 Une variable pointeur est déclarée en utilisant le caractère « \* » devant le nom de la variable. Cette notation permet de distinguer les variables réelles des pointeurs.

Précisons également qu'un pointeur « pointe » sur une donnée en mémoire et que cette donnée peut être codée sur plusieurs emplacements mémoire. Dans ce cas, le pointeur « pointe » sur le premier emplacement de la donnée :



### B.2. Utilisation

Pour déclarer un pointeur, il suffit d'utiliser la syntaxe suivante :

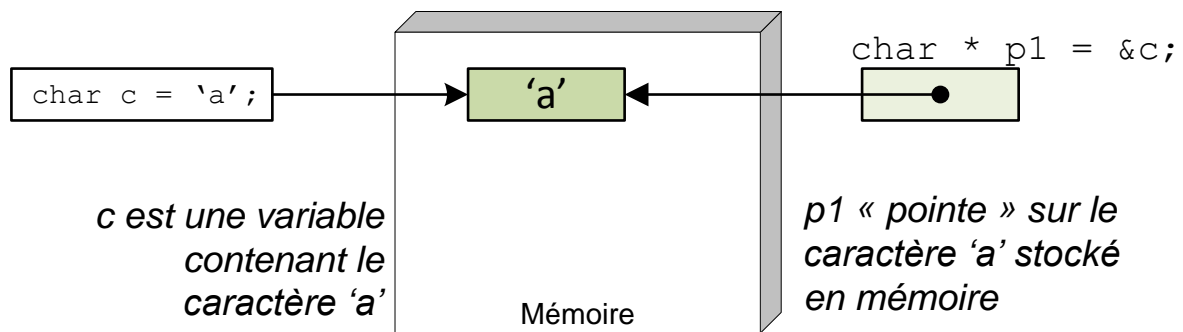
```
int *i ;           // « * » dans la déclaration indique qu'il s'agit d'un pointeur
```

Le pointeur est alors utilisable pour affecter le contenu de l'adresse correspondante :

```
*i = 134;          // « * » dans l'affectation définit le contenu de l'adresse pointée
```

Il est possible de manipuler les pointeurs de telle manière qu'ils « pointent » sur un emplacement déjà occupé par une variable :

```
char c = 'a' ;
char *p1 = &c ;    // « & » signifie adresse de la variable c. p1 « pointe » sur c
```



### B.3. Arithmétique des pointeurs

Il s'agit sans doute d'une notion apportant le plus de puissance aux pointeurs. L'arithmétique des pointeurs permet de manipuler la mémoire avec une grande flexibilité avec d'excellentes performances. Voici quelques explications :

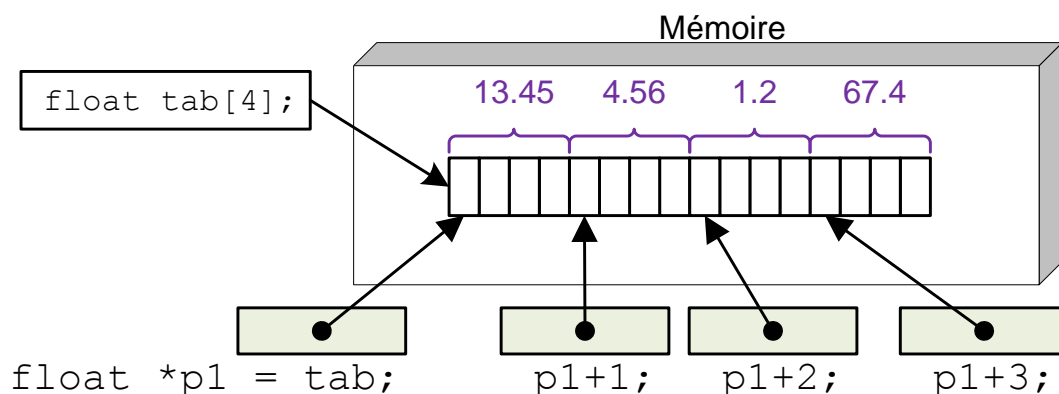
- Etant donné qu'un pointeur est « typé », nous savons donc quel type de donnée est stocké à l'emplacement mémoire associé.
- Chaque type de donnée a une taille définie à l'avance ( char = 1 octet, float = 4 octets, double = 8 octets, ...)
- Les opérateurs arithmétiques tiennent compte de cette taille ( +, -, ...) et permettent de déplacer un pointeur vers un autre emplacement mémoire

Voici un exemple illustrant l'arithmétique des pointeurs :

```
float tab[4] = { 13.45 , 4.56, 1.2, 67.4 }; // tab est l'adresse du tableau
float *p1 = tab ; // p1 est un pointeur qui pointe sur le 1er élément du tableau

for (int i=0 ; i<4 ; i++) { // Boucle d'affichage des flottants
    printf("%f , ", p1); // Affichage du flottant pointé par p1
    p1 = p1 + 1; // p1 pointe sur le flottant suivant
}
```

Dans cet exemple, « p1 » pointe sur le 1<sup>er</sup> élément du tableau :



## C. Quelques questions

Voici quelques questions en relation avec les pointeurs et les structures.

**Question 1.** Décrivez dans un tableau, les différents types de données numériques et alphanumériques en précisant l'intervalle de nombres et l'occupation mémoire.

**Question 2.** Le programme suivant manipule 2 pointeurs de types différents. L'un sur le type « unsigned int » et l'autre sur un « char ». Les 2 pointeurs vont pointer à la même adresse en mémoire. Quel sera le contenu de p2 affiché ? Expliquez ce résultat

```
#include <stdio.h>

int main() {
    unsigned int entier = 134486;
    char *p;
    p=&entier;    // p pointe sur entier
    printf("Contenu de p =%c\n", *p);

    return 0;
}
```

**Question 3.** Dessinez un schéma de la mémoire pour la structure suivante en prenant en compte la taille des différents éléments de la structure.

```
struct ethernet {
    char macdest[6];
    char macsource[6];
    short int type;
    int checksum;
};
```

## D. Enoncé des exercices

### D.1. Déclaration et utilisation



Déclarez un pointeur sur un « char » et affectez le contenu avec le caractère 'r'.

### D.2. Afficher une chaîne de caractères

Soit le programme suivant :

```
#include <stdio.h>

void afficher( char *p) {

    while ( *p != '\0') {
        // A COMPLETER
    }
}

int main() {
    char *chaine = "bonjour    les    amis";
    afficher(chaine);
    return 0;
}
```



Complétez la boucle « while » de la fonction « afficher » pour qu'elle affiche tous les caractères de la chaîne SANS les espaces (Rappel : le caractère '\0' est le caractère de fin de chaîne en C).

### D.3. Passage par adresse

Soit la fonction « main » suivante appelant une fonction « calcul » qui réalise deux opérations : la somme et la différence de 2 entiers.

```
int main() {

    int a=12, b=45;
    int somme, difference;

    calcul(a,b, &somme, &difference);

    printf("a+b=%d , a-b=%d\n", somme, difference);

    return 0;
}
```



Ecrire la fonction « calcul » pour qu'elle puisse réaliser le travail demandé

## D.4. Traitement de chaîne

Le but est de copier une chaîne de caractère dans une autre en inversant la casse (Passer de minuscule à majuscule et vice-versa). La fonction « scase », donc voici la déclaration, assure cette tâche :

```
void scase( char *p1, char *p2);
```

Voici le programme principal chargé d'appeler "scase" :

```
int main() {  
  
    char ch1[] = "abcdefghi jKLMnopqrstuvwxyz";  
    char ch2[26];  
  
    scase(ch1, ch2);  
  
    printf("ch1 = %s , ch2= %s \n", ch1, ch2);  
  
    return 0;  
}
```



Complétez la fonction « scase » en utilisant les 2 pointeurs passés en paramètres de la fonction. N'oubliez pas que, d'après la table ASCII, les majuscules et les minuscules sont séparés de 32 caractères :

```
char c1 = 'A';  
char c1bis = 'a';  
char c2 = c1 + 32; // c2 contient 'a'  
char c3 = c1bis - 32; // c3 contient 'A'
```

## D.5. Calculer la taille d'une chaîne

La fonction strlen renvoie la longueur d'une chaîne de caractères. Un débutant en langage C écrit cette fonction de la façon suivante:

```
int strlen(char *s)  
{  
    int i;  
    int len = 0;  
    for (i=0; s[i]!=0; i++) {  
        len++;  
    }  
    return len;  
}
```

En fait comme s est un pointeur, on peut s'en servir pour parcourir la chaîne plutôt que d'utiliser une variable supplémentaire i. Il suffit pour cela de l'incrémenter pour le faire pointer successivement vers les différents caractères de la chaîne.

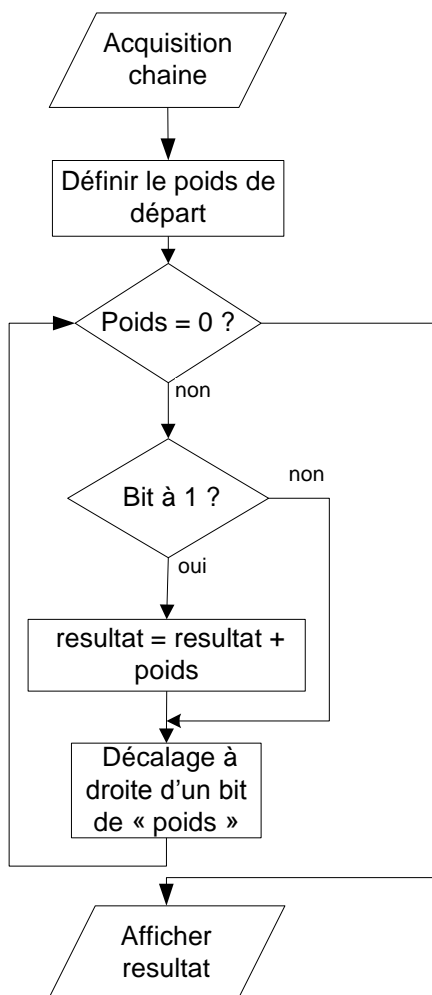
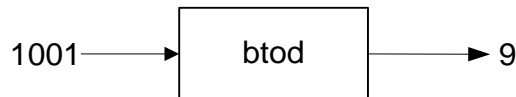


Réécrire ainsi la fonction strlen en utilisant une seule variable locale.

## D.6. Conversion binaire-décimal

### a. Objectif

L'objectif est de créer un programme ( btod.c ) capable de convertir un nombre binaire en nombre décimal. Une chaîne de caractère sera passée en argument du programme. Cette chaîne – une suite de 0 et de 1 – sera convertie en décimal.



### b. Algorithme

L'algorithme sera basé sur le parcours de la chaîne passée en paramètre. Si le caractère est un « 1 », on ajoutera le poids correspondant à une variable « resultat ». A la fin du parcours de la chaîne, la variable « resultat » sera le nombre converti en décimal.

### c. Poids fort

A chaque caractère de la chaîne d'entrée, correspond un poids. Il faut donc connaître la taille de la chaîne pour connaître le poids du 1<sup>er</sup> caractère.





Trouvez une relation entre la taille de la chaîne et le poids fort du nombre à traduire.

#### d. Vérification d'usage

Nous devons vérifier qu'il y a bien 1 paramètre passé en argument. On utilise la variable « argc » et « argv » qui contiennent respectivement le nombre de paramètres et les valeurs de ces paramètres sous forme d'un tableau de pointeur sur des chaînes :

```
int main(int argc, char *argv[]) {  
  
    if (argc<2) {  
        printf("Usage : %s chiffre_binaire\n", argv[0]);  
        exit(1);  
    }  
    ...  
}
```



*Le 1<sup>er</sup> argument du tableau « argv » contient le nom du programme et non le 1<sup>er</sup> paramètre passé au programme.*

#### e. Mise en œuvre



Ecrivez le programme et testez son bon fonctionnement.