

# Langage C : pointeurs et allocation dynamique

<b>POINTEURS.....</b>	<b>1</b>
<b>SYNTAXE GENERALE .....</b>	<b>1</b>
<b>DEFINITION DE POINTEURS : .....</b>	<b>1</b>
<b>ALLOCATION DYNAMIQUE.....</b>	<b>2</b>
<b>RESERVATION.....</b>	<b>2</b>
LA COMMANDE MALLOC() : .....	2
L'OPERATEUR sizeof ET LES TAILLES DES TYPES. ....	2
TEST DE LA VALIDITE DE LA ZONE OBTENUE : .....	3
TRANSTYPAGE DE MALLOC() : .....	4
LA COMMANDE CALLOC() : .....	5
<b>LIBERATION.....</b>	<b>7</b>
<b>REALLOCATION, COPIE DE ZONE .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>

## Pointeurs

### Syntaxe générale

La syntaxe utilisée par le langage C est exactement la même que celle utilisée par le langage algorithmique pour les deux opérateurs fondamentaux que nous avons traités concernant les pointeurs :

& signifie : "adresse de"  
\* signifie : "contenu de"

### définition de pointeurs :

Un pointeur, en langage C, est également défini par le type de la valeur que l'on manipule lorsque l'on accède à son contenu. Ainsi, un pointeur sur un `reel`, défini de la manière suivante en algorithmique :

```
reel *pointeur;
```

Sera défini de la manière suivante en C, si l'on utilise `double` comme nom de type :

```
double *ptr;
```

En langage C, la valeur NULL vue en cours d'algorithmique est également utilisable, il suffit pour cela d'inclure le fichier d'entête **stdlib.h**.

Rappel sur les noms de type du langage C :

Langage algorithmique	langage C	taille en octets
caractere	char	1
entier	long	4
reel	double	8

## Allocation dynamique

### Réservation

Il existe en Langage C deux commandes correspondant à la commande reservation du langage algorithmique, ces deux commandes sont accessibles à condition d'inclure le fichier **stdlib.h** au début du programme.

*La commande malloc() :*

Allocation simple : commande `malloc()` , pour Memory ALLOCation. C'est la traduction la plus simple de la demande de réservation de mémoire.

Syntaxe : **`malloc(nombre_d_octets_demande)`** ; donne l'adresse d'une zone de mémoire comportant le nombre d'octets demandé.

*L'opérateur sizeof et les tailles des types.*

Le calcul du nombre d'octets à demander peut être fait directement par l'ordinateur. Ce nombre total est égal au : nombre de variables demandées × la taille en octet d'une variable.

En langage algorithmique, la demande de réservation se fait en précisant ce nombre de valeurs et le type. En langage C, nous utiliserons l'opérateur `sizeof()` auquel on fournit entre parenthèses le nom du type concerné, et qui indique le nombre d'octets occupé en mémoire par une variable de ce type.

Exemple : le programme en langage C suivant :

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    long a;
    char let;
    double x;

    printf("la variable a   occupe %ld octets\n", sizeof(long));
    printf("la variable let occupe %ld octets\n", sizeof(char));
    printf("la variable x   occupe %ld octets\n", sizeof(double));

    system("PAUSE");
    return(0);
}
```

affiche :

```
la variable a    occupe 4 octets
la variable let occupe 1 octets
la variable x    occupe 8 octets
```

pour traduire la commande reservation, il suffit de multiplier le nombre de variables demandées par la taille du type précisé :

exemples : on suppose que les variable `nb` et `m` sont définies et initialisées avec une certaine valeur.

Langage algorithmique	Langage C
<code>reservation(15 entiers);</code>	<code>malloc(15*sizeof(long));</code>
<code>reservation(nb caracteres);</code>	<code>malloc(nb*sizeof(char));</code>
<code>reservation((3*m+1) reel);</code>	<code>malloc((3*m+1)*sizeof(double));</code>

Toute utilisation de la commande `malloc()` du langage C doit utiliser l'opérateur `sizeof()`, pour des questions de portabilité et de compatibilité.

La commande `malloc()`, comme la commande `reservation` à laquelle elle est associée, indique l'adresse de la zone de la mémoire disponible, il faut donc immédiatement stocker cette valeur dans un pointeur. La commande `malloc()` doit donc systématiquement se trouver dans une ligne comportant une affectation de son résultat dans un pointeur.

```
#include <stdlib.h>
```

```
void main()
{
    long *zone;
    long nb;

    printf("nombre de valeurs :");
    scanf("%ld",&nb);

    zone = NULL;    // initialisation de précaution

    if (nb > 0)
    {
        zone = malloc(nb*sizeof(long));
    }
}
```

*test de la validité de la zone obtenue :*

`malloc()`, dans la plupart des cas, fonctionne correctement et renvoie bien l'adresse de la zone de mémoire demandée. Cependant, il se peut que cette commande échoue. Dans ce

cas, toute tentative d'accès à une zone de mémoire non allouée conduit à un plantage en bonne et due forme du programme, ce qu'il faut impérativement éviter. Pour cela, il est possible de tester si le pointeur a été correctement initialisé. En cas d'échec, en effet, `malloc()` renvoie la valeur `NULL`. On peut donc tester l'égalité du pointeur à cette valeur pour savoir s'il est possible de manipuler la zone de mémoire demandée (que l'on n'a donc pas forcément obtenue).

Cela s'écrit de la manière suivante :

```
#include <stdlib.h>

void main()
{
    double *zone;
    long nb;

    nb = 25;

    zone = malloc(nb * sizeof(double));

    // test de la validité du pointeur

    if (zone == NULL) // malloc n'a pas fonctionné
    {
        printf("erreur d'allocation par malloc()\n");
    }
    else
    {
        // accès à la zone de mémoire et suite du programme
    }
}
```

*transtypage de malloc() :*

la commande `malloc()` donne l'adresse d'une zone de mémoire, mais cette adresse retournée ne permet pas de savoir quel est le type des valeurs qui seront stockées dans cette zone. En effet, ce que l'on indique à `malloc()`, c'est simplement le nombre d'octets demandés. Dans ce sens, on dit que l'adresse renvoyée ou donnée par `malloc()` est **générique** : cela se traduit en langage C en disant que l'adresse de la zone donnée par `malloc()` est de type `void*` : pointeur générique. Or cette adresse sera affectée à un pointeur du programme, dont le type sera `long*` ou `char*` ou encore `double*` : au sens strict, ces types sont différents. Pour éviter les messages d'avertissement et garder une bonne compatibilité, on prend systématiquement la précaution de faire précéder l'utilisation de

`malloc()` d'un transtypage, en indiquant entre parenthèses le type du pointeur dans lequel sera affecté l'adresse que la commande fournira.

### Exemple :

Au lieu d'écrire :

```
long *ptr;  
long n;  
  
printf("nombre de valeurs :");  
scanf("%ld", &n);  
  
ptr = malloc(nb*sizeof(long));
```

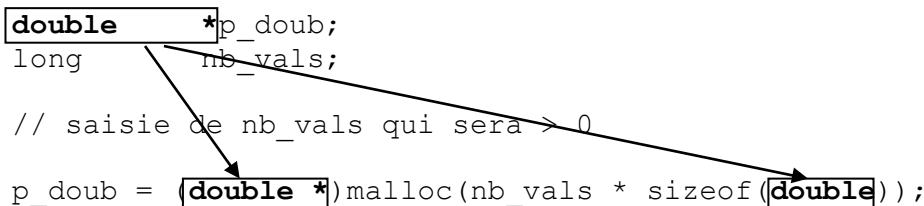
on écrira :

```
ptr = (long *)malloc(nb*sizeof(long));
```

De même, selon le type du pointeur dans lequel on affectera la valeur donnée par la commande.

Voici un moyen simple de bien vérifier qu'une allocation est bien écrite : dans l'exemple suivant, on utilise le type `double`, mais on peut le remplacer par `char` ou `long` sans aucun problème :

```
double *p_doub;  
long nb_vals;  
  
// saisie de nb_vals qui sera > 0  
  
p_doub = (double *)malloc(nb_vals * sizeof(double));
```



le type du pointeur affecté doit être le même que le type de transtypage; et le type des valeurs stockées doit être le même que celui précisé à l'opérateur `sizeof`.

### *La commande `calloc()` :*

`Calloc` signifie Clear memory ALLOCation. A la différence de `malloc()`, `calloc()` permet de réserver une zone de mémoire et de l'initialiser en y reportant la valeur 0. Cela est intéressant dans la mesure où la zone possède une valeur déterminée à l'avance. Sa syntaxe est également légèrement différente :

```
calloc(nombre_variables, taille_d_une_variable);
```

`calloc()` donne l'adresse générique (donc de type `void*`) d'une zone permettant de stocker le nombre de variables demandé, chacune de ces variables occupant la taille, en octets, précisée. En clair, cela revient à remplacer la multiplication (\*) de `malloc()` par une virgule.

Lorsque `calloc()` échoue à trouver une zone de mémoire adéquate, l'adresse donnée est `NULL`, comme avec `malloc()`. Vous pouvez utiliser, indifféremment, `malloc()` ou `calloc()`. Le transtypage doit également être fait.

Exemples (repris de la section concernant `malloc()`) :

```
#include <stdlib.h>

void main()
{
    long *zone;
    long nb;

    printf("nombre de valeurs :");
    scanf("%ld",&nb);

    zone = NULL;    // initialisation de précaution

    if (nb > 0)
    {
        zone = (long *)calloc(nb, sizeof(long));
    }
}
```

deuxième exemple :

```
#include <stdlib.h>

void main()
{
    double *zone;
    long nb;

    nb = 25;

    zone = (double *)calloc(nb, sizeof(double));

    // test de la validité du pointeur

    if (zone == NULL) // calloc n'a pas fonctionné
    {
        printf("erreur d'allocation par calloc()\n");
    }
    else
```



```
{  
    // accès à la zone de mémoire et suite du programme  
}
```

## Libération

La commande `liberer(pointeur)` se traduit simplement par la commande du langage C : **`free()`**, à laquelle on doit également fournir, entre parenthèses, le nom du pointeur désignant la zone de mémoire à libérer. Exemple :

```
#include <stdlib.h>
```

```
void main()
```

```
{  
    char *zone;  
    long nb;  
  
    nb = 12;  
  
    zone = (char *)malloc(nb * sizeof(char));  
  
    // traitements  
  
    free(zone) ;  
}
```

`free()` fonctionne aussi bien avec les zones allouées par `malloc()` que pour les zones allouées avec `calloc()`.