Programmation Impérative II

Allocation dynamique de mémoire
Matrice

minh-anh.tran@u-pec.fr

Rappel. Après sa déclaration, un pointeur est nul systématiquement. Il ne pointe vers aucune adresse !!!

Utilisation de pointeurs

1ère option: Pointer vers l'adresse d'une variable déjà définie

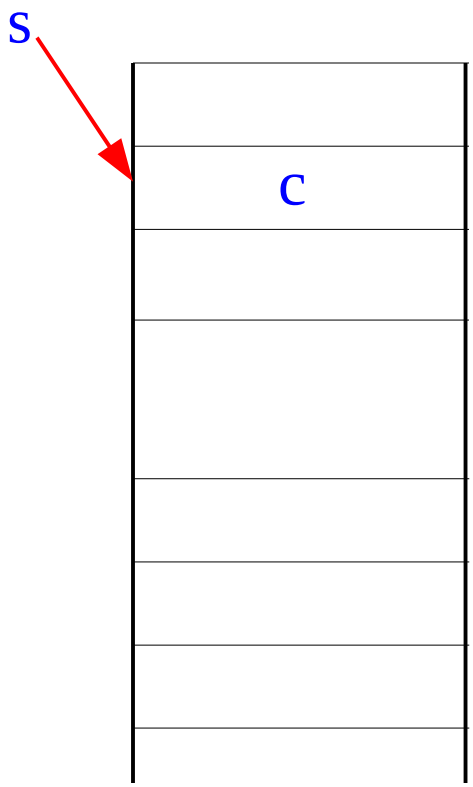
```
char * s, c;

s = &c;
s[0] = getchar ();

printf ("%c\n", s[0]);
printf ("%c\n", *s);
```

Déclaration de s, c

Pointer s vers c

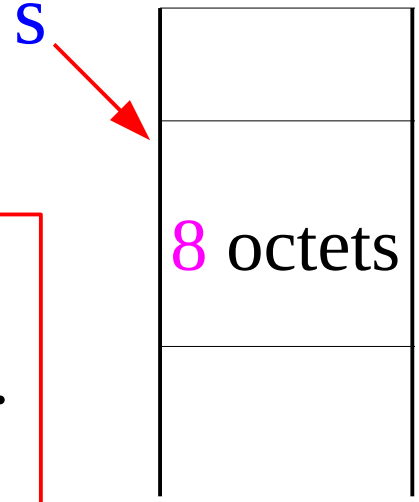


2ème option: allocation dynamique de mémoire

Allocation dynamique - *malloc*

Par la fonction *malloc* de la bibliothèque ***stdlib***

```
s = malloc (8);
```



Attention.

- La valeur de chaque bit est indéterminée.
- Il faut utiliser la bibliothèque ***stdlib***

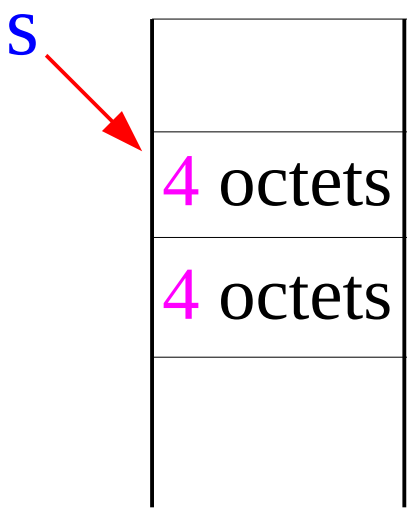
  ```
  #include <stdlib.h>
  ```

- S'il n'y a plus de mémoire disponible,

  s = NULL

Allocation dynamique - *calloc*

```
s = calloc(2, 4);
```

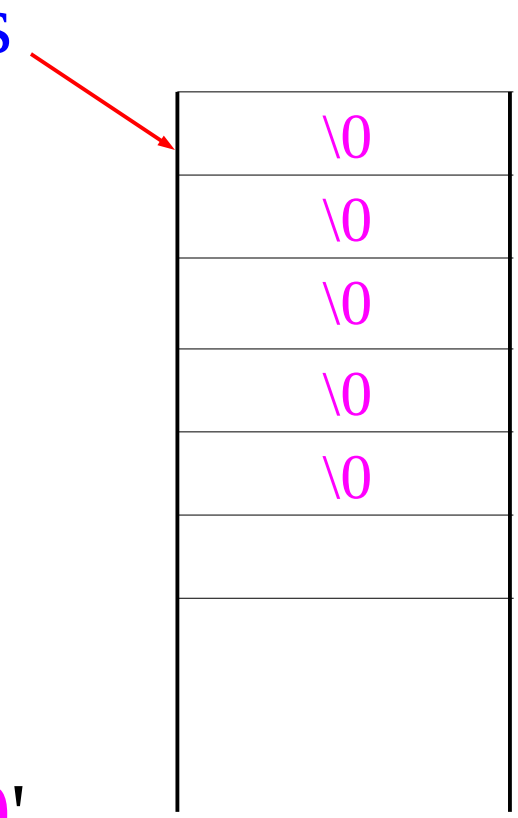


presque équivalente à

```
s = malloc(8);
```

La différence. Tous les bits sont mis à 0

Allocation dynamique - *calloc*

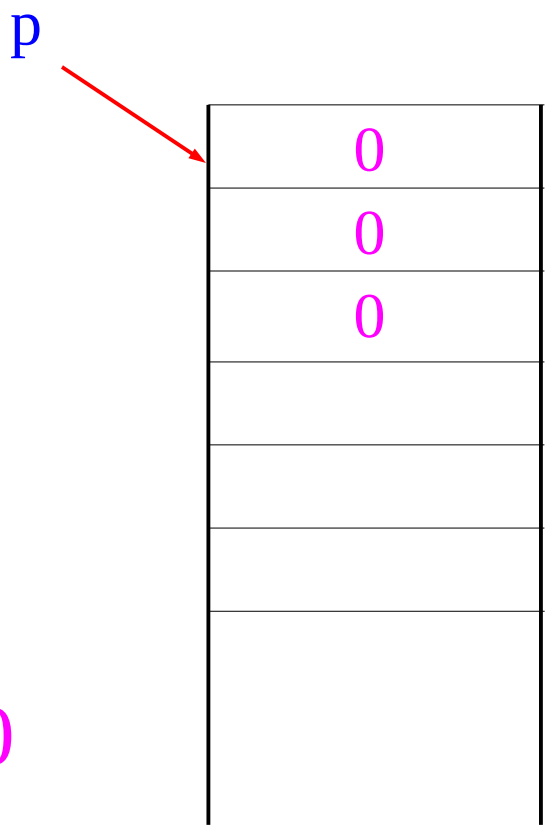


Exemples.

```
char * s;
s = calloc(5, sizeof(char));
```

réserve (5*sizeof(char)) octets sur la mémoire, et les mettre à valeur '\0'

Allocation dynamique - *calloc*



Exemples.

```
int * p;
p = calloc(3, sizeof(int));
```

réserve (3*sizeof(int)) octets sur la mémoire, et les mettre à valeur 0

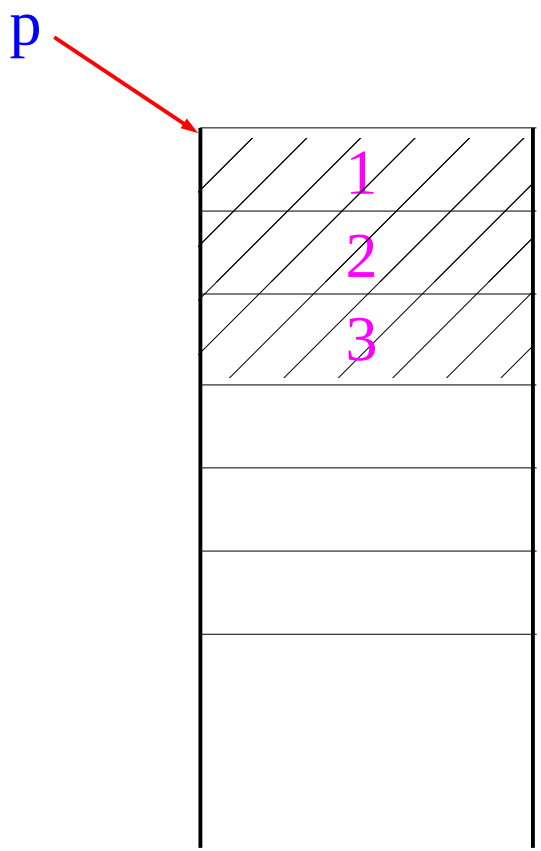
De même pour les autres types: float, double...

Allocation dynamique

La fonction ***sizeof***

Type	Taille
char	1 octet
short int	2 octets
int	4 octets
float	4 octets
double	8 octets

Allocation dynamique - *realloc*



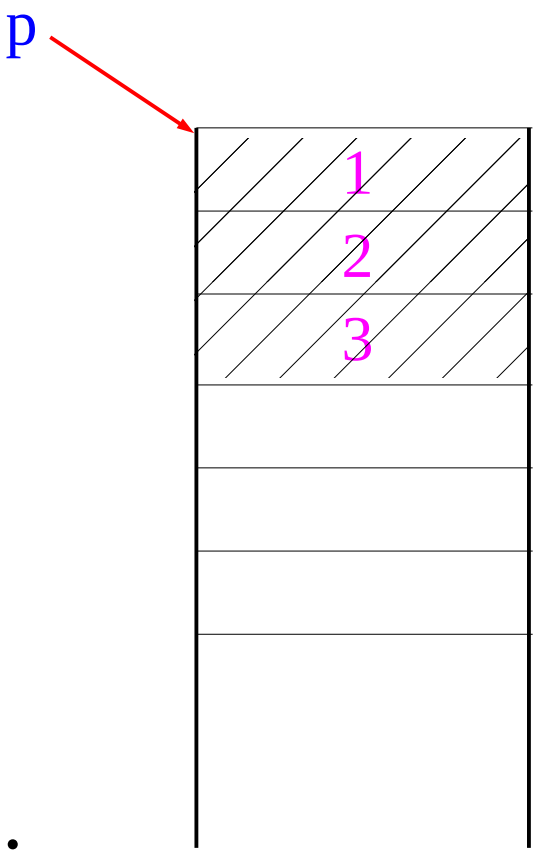

```
int * p;
p = malloc(2 * sizeof(int));
p[0] = 1; p[1] = 2;

p = realloc(p, 3 * sizeof(int));
p[2] = 3;
```

redimensionne le bloc alloué avec la nouvelle taille 3 * sizeof(int)

p = NULL s'il n'y a plus de mémoire disponible

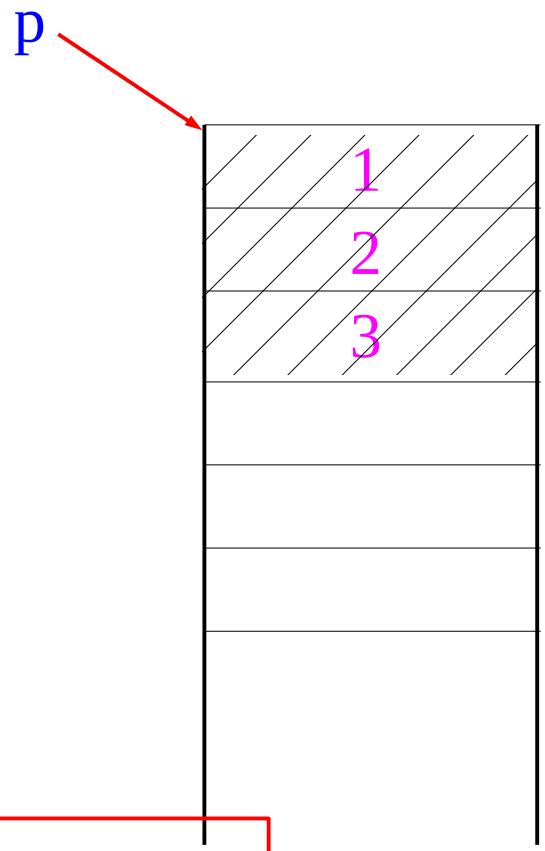
Allocation dynamique - *realloc*




```
int * p;
p = malloc(2 * sizeof(int));
p[0] = 1; p[1] = 2;

int * temp;
temp = realloc(p, 3 * sizeof(int));
if (temp != NULL)
   { p = temp; }
```

Allocation dynamique - *free*



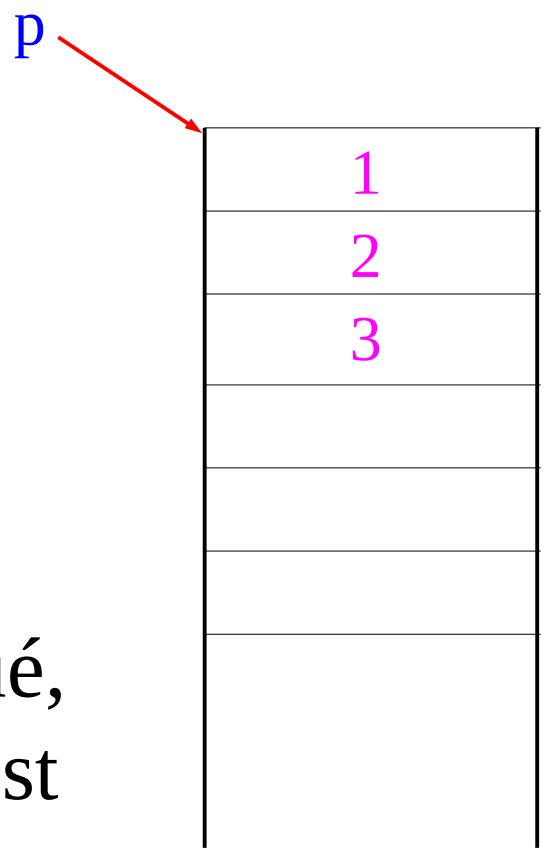
Libération de mémoire:

free (p);

Libérer la mémoire reservée
- rendre cette mémoire disponible

Attetion. La fonction free ne change pas la valeur de **p**, c.a.d. **p** pointe vers la même location

Allocation dynamique - *free*



Si p=NULL, cette fonction ne fait strictement rien.

En revanche, si l'on appelle **free(p)** dont **p** a précédemment été désalloué, le comportement de cette fonction est indéterminée. Il vaut mieux mettre:

```
free (p);
p = NULL;
```

Matrices - Tableaux à 2 indices

```
int T[3][4];
```

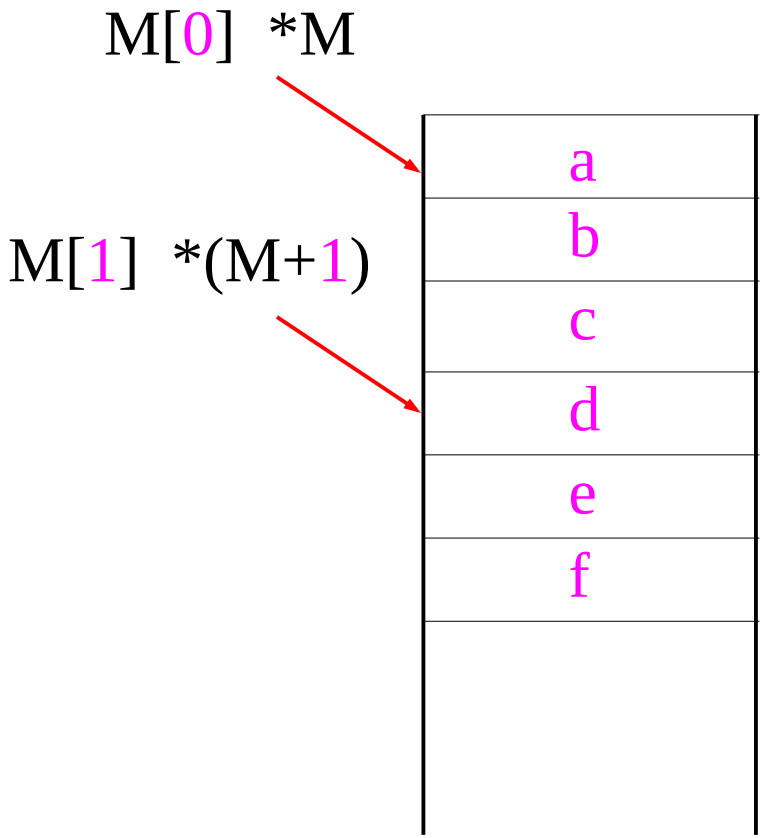
T désigne un tableau de 3 éléments, chacun de ces éléments étant un tableau de 4 éléments

T	&T[0][0]
T[0]	&T[0][0]
T[1]	&T[1][0]
T+1	&T[1][0]
*(T+1)+2	&T[1][2]

Matrice

```
char M[2][3] = {{'a', 'b', 'c'},
                {'d', 'e', 'f'}};
```

M[0] est l'adresse du M[0][0]
M[1] est l'adresse du M[1][0]

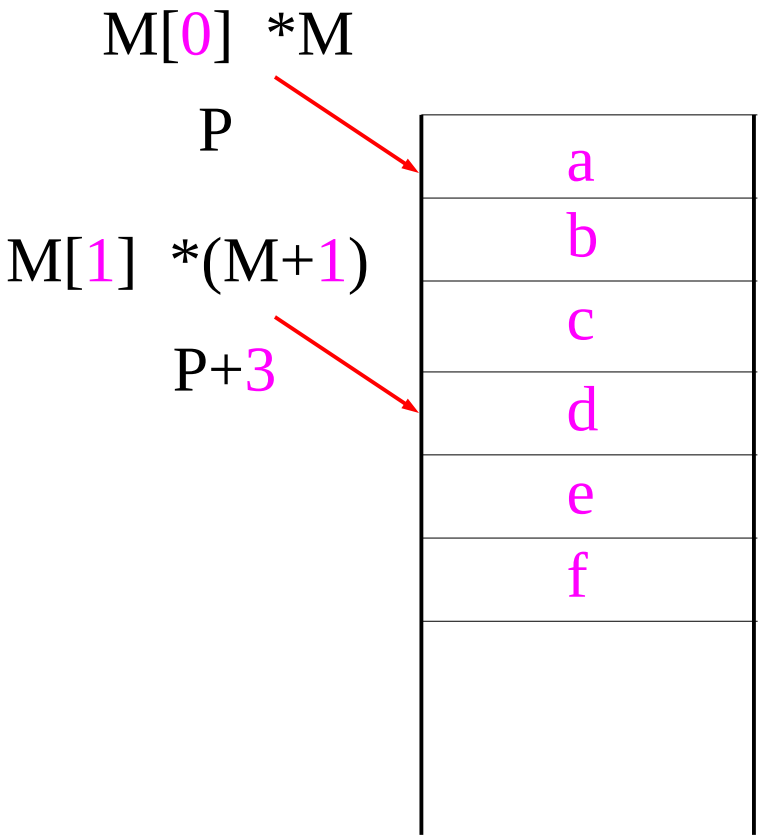


Matrice

```
char M[2][3] = {{'a', 'b', 'c'}
                {'d', 'e', 'f'}};

char * P;
P = (char *) M;
```

changement de type



Matrice – allocation dynamique

```
char ** M;

M = malloc (2 * sizeof(char *));
M[0] = calloc (2 * sizeof(char));
M[1]= malloc (3 * sizeof(char));

M[0][0] = 'a'; M[0][1] = 'b';
M[1][0] = 'c'; M[1][1] = 'd'; M[1][2] = 'e';
```

Que se passe-t-il dans la mémoire?

Matrice – allocation dynamique

Libération de mémoire:

```
free(M[0]); M[0] = NULL;
free(M[1]); M[1] = NULL;
free(M); M = NULL;
```