6- LANGAGE C

LES FONCTIONS II

Loïc Cuvillon
l.cuvillon@unistra.fr

Sommaire chapitre 6

Les fonctions II

Rappel : Passage de paramètres par valeur

- Que faire si une fonction doit modifier la valeur d'une variable de la fonction appelante (un argument effectif) ?

Exemple: Echanger le contenu de 2 variables du `main()`

```
void echange(int i, int j)
{   int tmp;
    tmp = i ;
    i = j   ;
    j = tmp ;
}
```

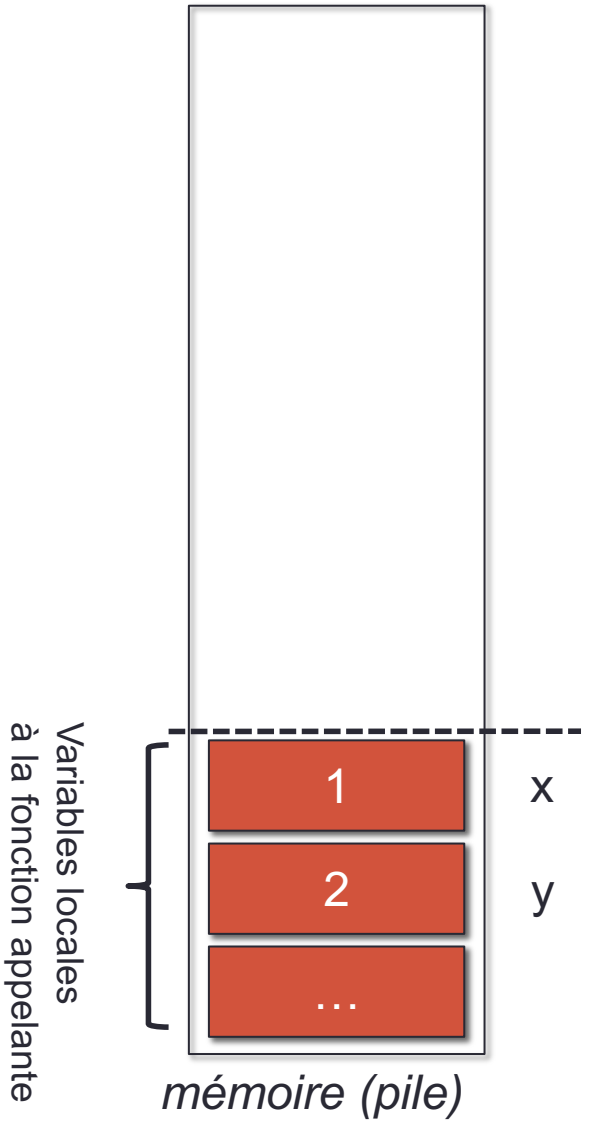
Que se passe-t-il pour `x` et `y` dans un programme qui appelle `echange (x, y)` ?

Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
        echange(x,y);
        ...
}
```

0

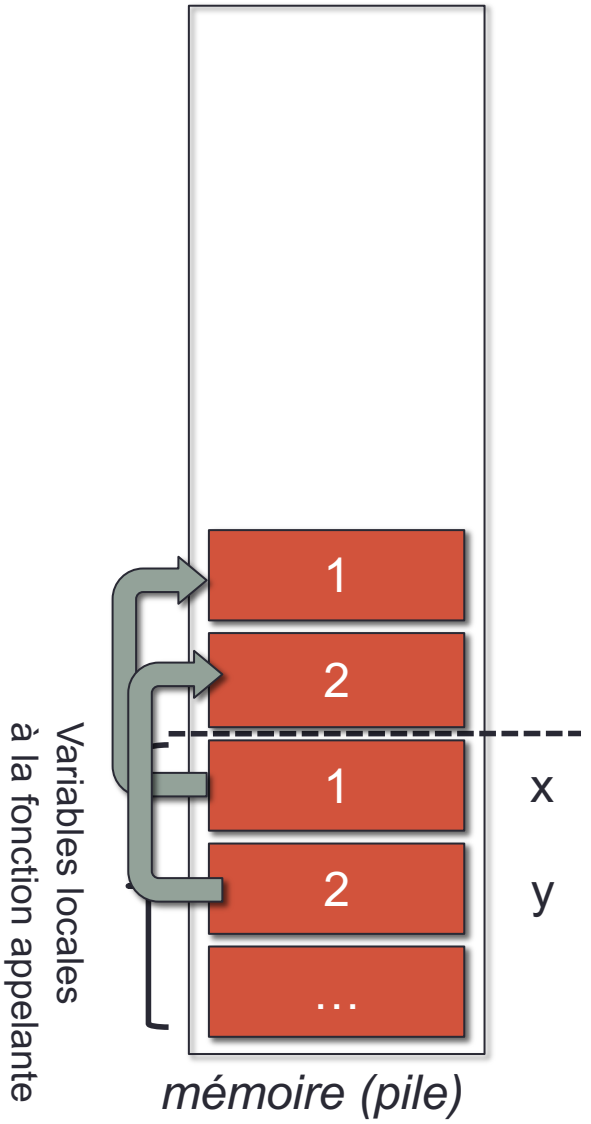
```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```



Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
1       echange(x,y);
        ...
}
```

```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```



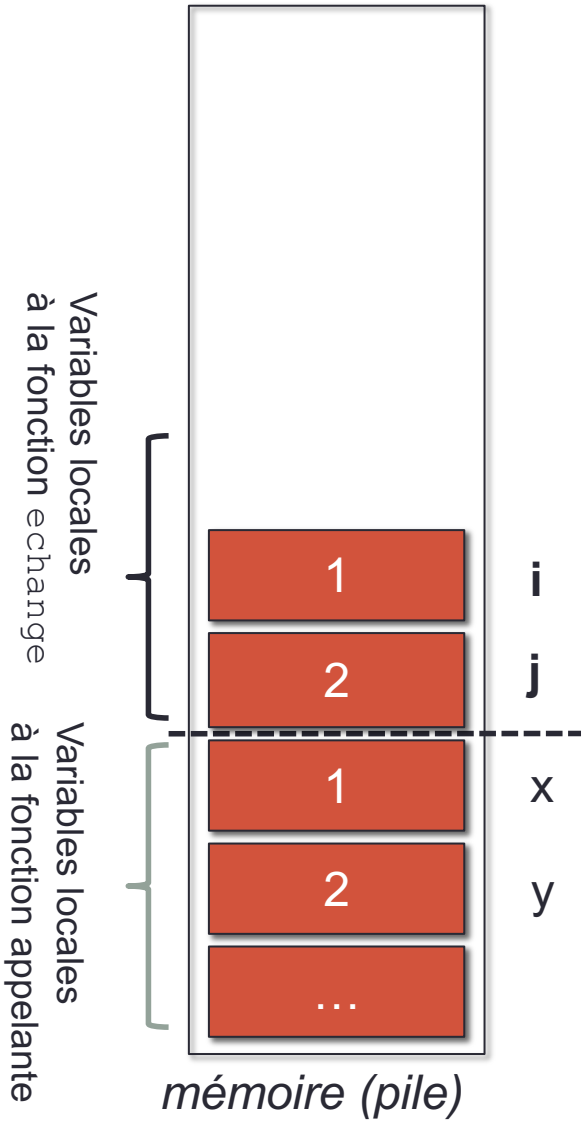
1. Appel fonction:
   - **copie de la valeur** des variables `x,y` sur la pile

Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
        echange(x,y);
        ...
}
```

2

```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```



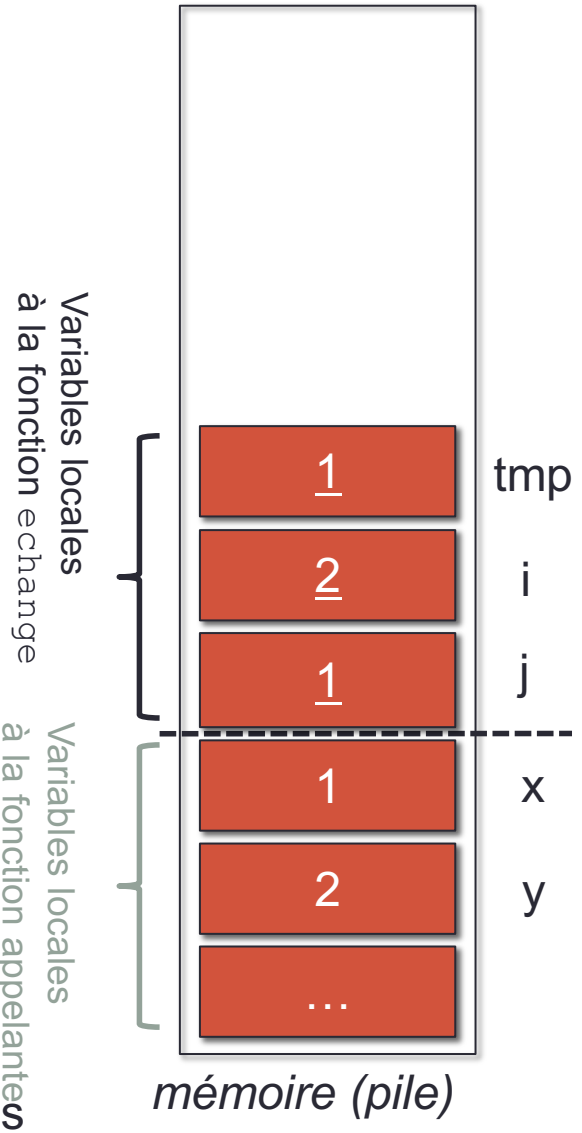
2. Entrée dans la fonction:
 - valeur des variables `i`,`j`, arguments formels, trouvée sur le sommet de la pile

Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
        echange(x,y);
        ...
}
```

```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```

3



3. Exécution de la fonction:
   - création variables locales
   - exécution instructions → `i` et `j` variables locales échangées

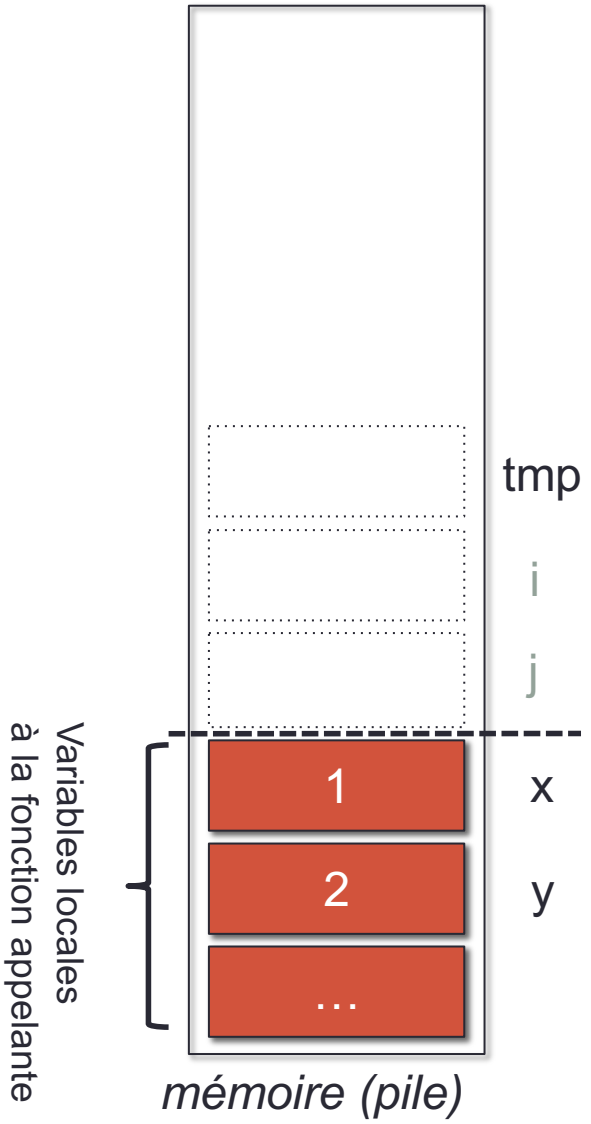
Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
        echange(x,y);
        ...
}
```

```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```

4

4. Sortie de la fonction :
   - suppression des variables et arguments locaux !

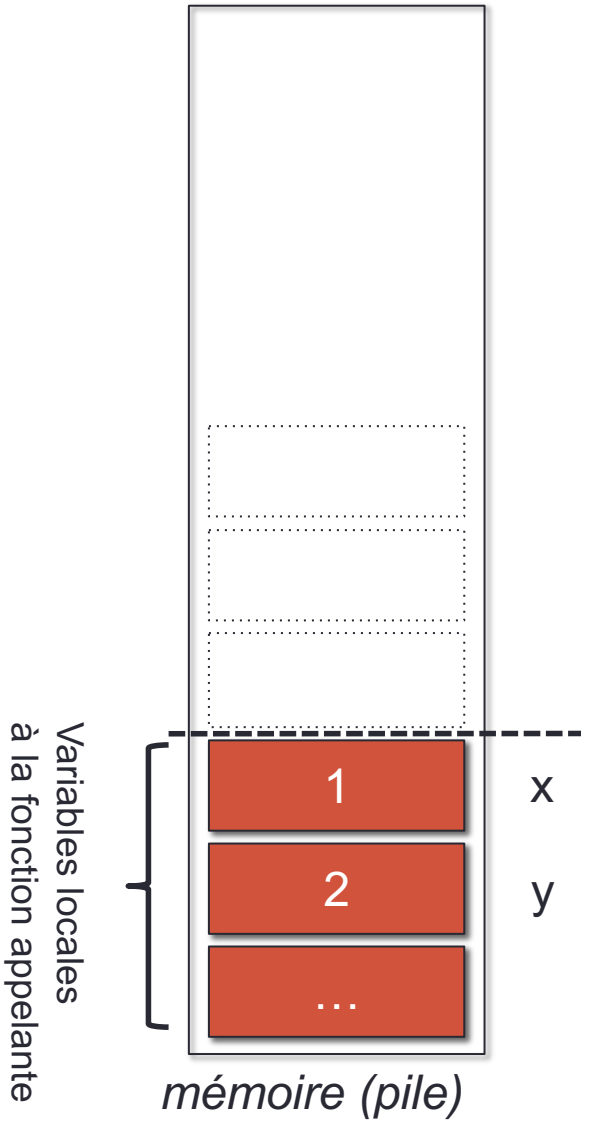


Rappel : Passage de paramètres par valeur

```
{
        int x = 1, y = 2;
        ...
5       echange(x,y);
        ...
}
```

```
void echange(int i, int j)
{
        int tmp;
        tmp = i ;
        i = j ;
        j = tmp ;
}
```

5. Retour à la fonction appelante:
   → **x et y n'ont pas été modifiés ni échangés**



Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                          x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                          x, y);
     return 0;
 }
```

1. affiche: `avant echange 1 2`

Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

2. copie valeurs 1 et 2 dans variables locales de la fonction `i` et `j`

1. affiche: `avant echange 1 2`

Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

2. copie valeurs 1 et 2 dans variables locales de la fonction `i` et `j`

3. échange des valeurs locales: i=2 et j=1

1. affiche: `avant echange 1 2`

Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

2. copie valeurs 1 et 2 dans variables locales de la fonction `i` et `j`

3. échange des valeurs locales: i=2 et j=1

1. affiche: `avant echange 1 2`

4. retour de la fonction
affiche: `apres echange 1 2`

Passage par valeur

```
void  echange ( int i, int j )
{    int tmp;
     tmp = i;
     i=j;
     j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(x, y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

2. copie valeurs 1 et 2 dans variables locales de la fonction `i` et `j`

3. échange des valeurs locales: i=2 et j=1

1. affiche: `avant echange 1 2`

4. retour de la fonction
affiche: `apres echange 1 2`

- `x` et `y` non échangés, non modifiables
  car la fonction travaille sur des copies de leurs valeurs

Sommaire chapitre 6

Les fonctions II

Passage de paramètres par adresse

- Comment permettre à une fonction de modifier la valeur d'une variable de la fonction appelante ?

→ en lui passant l'adresse mémoire de cette variable

- La fonction récupère l'adresse dans une variable pointeur.
- L'indirection * sur le pointeur permet indirectement à la fonction de modifier en mémoire la valeur de la variable de la fonction appelante !

- On parle de **passage de paramètres par adresse**[1]

```
void echange(int *i, int *j)
{   int tmp;
    tmp = *i ;
    *i = *j  ;
    *j = tmp ;    }
```

Appel avec `echange (&x, &y)`

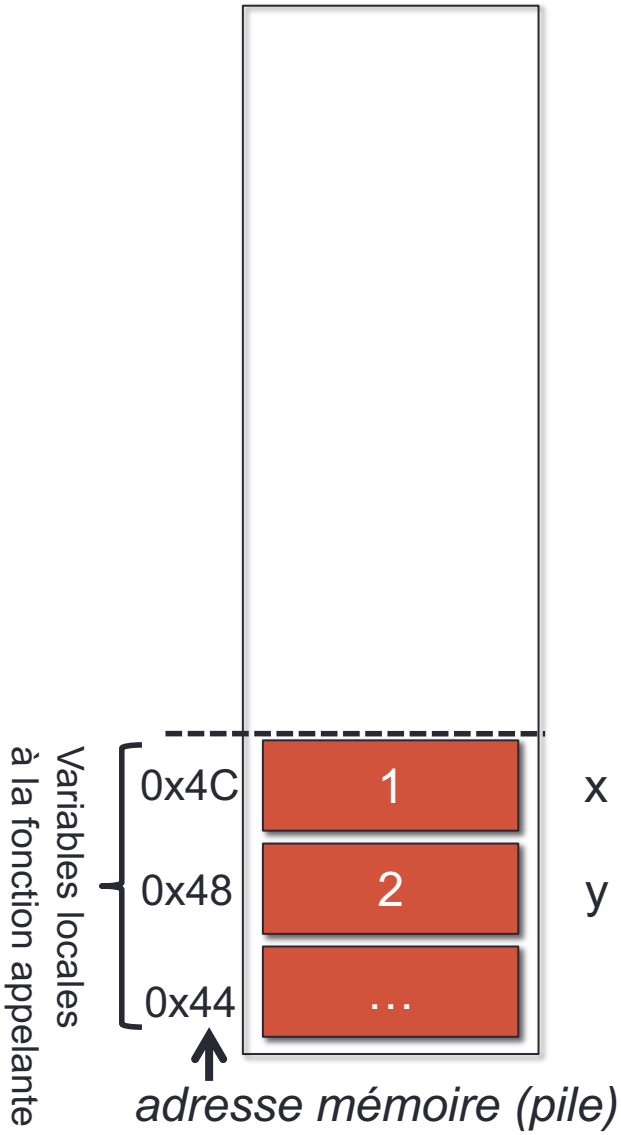
[1] Rigoureusement, c'est toujours un passage par valeur mais la valeur passée est ici une adresse mémoire.

Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

0

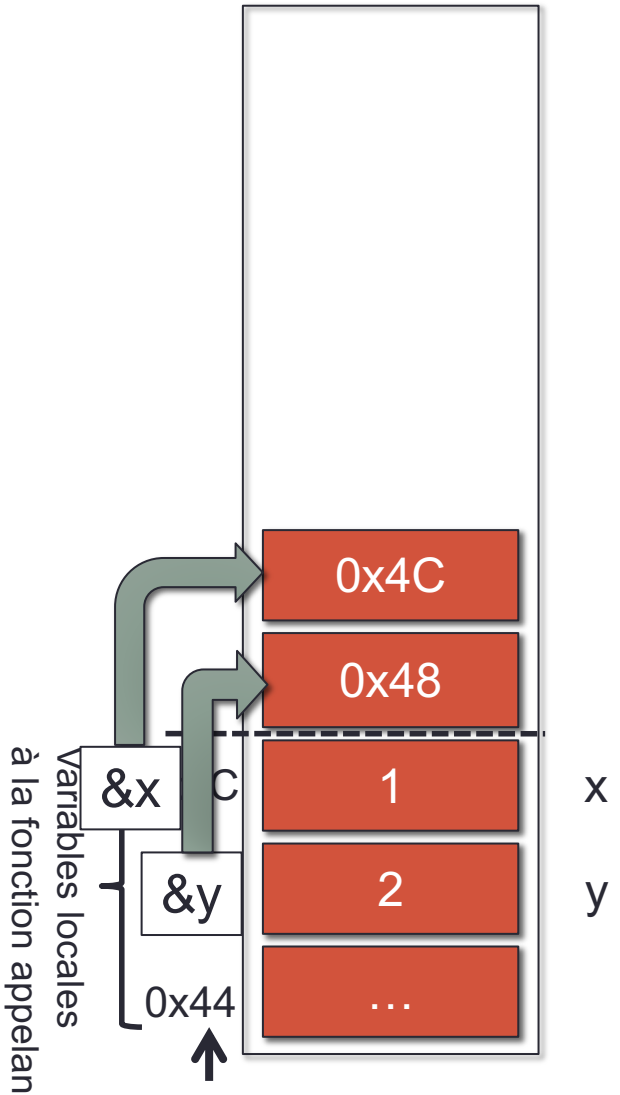
```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```



Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
1       echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```



1. Appel fonction:
   - copie des valeurs des arguments effectifs sur la pile
     == **copie des adresses** des variables `x` et `y`

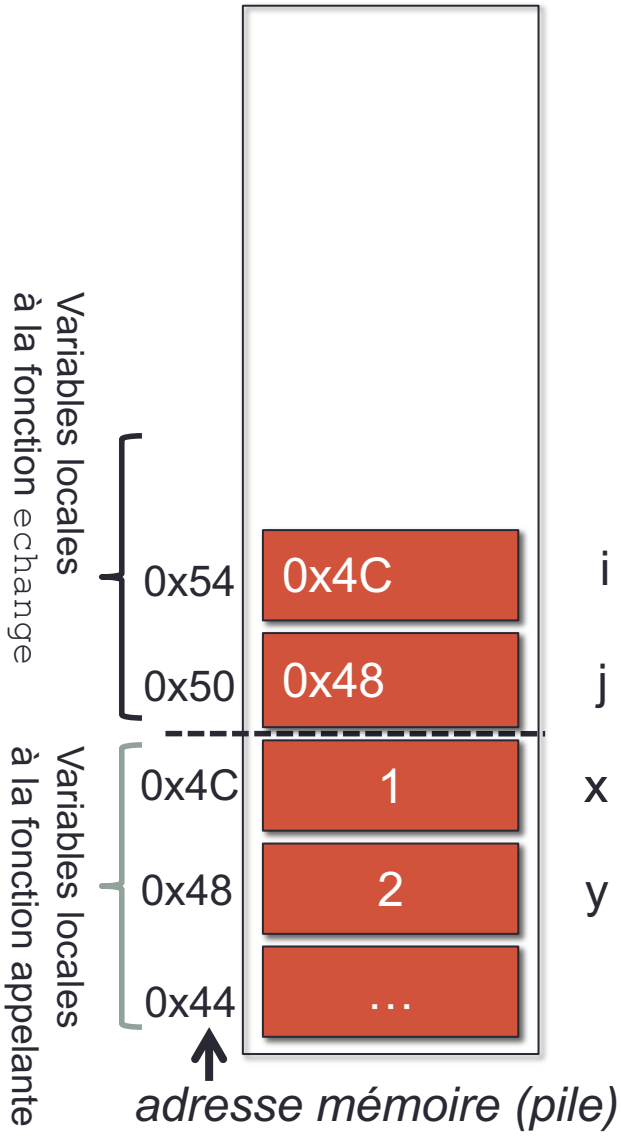
Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

2

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```

2. Entrée dans la fonction:
   - valeur des variables pointeurs `i` et `j` (arguments formels) trouvée sur la pile

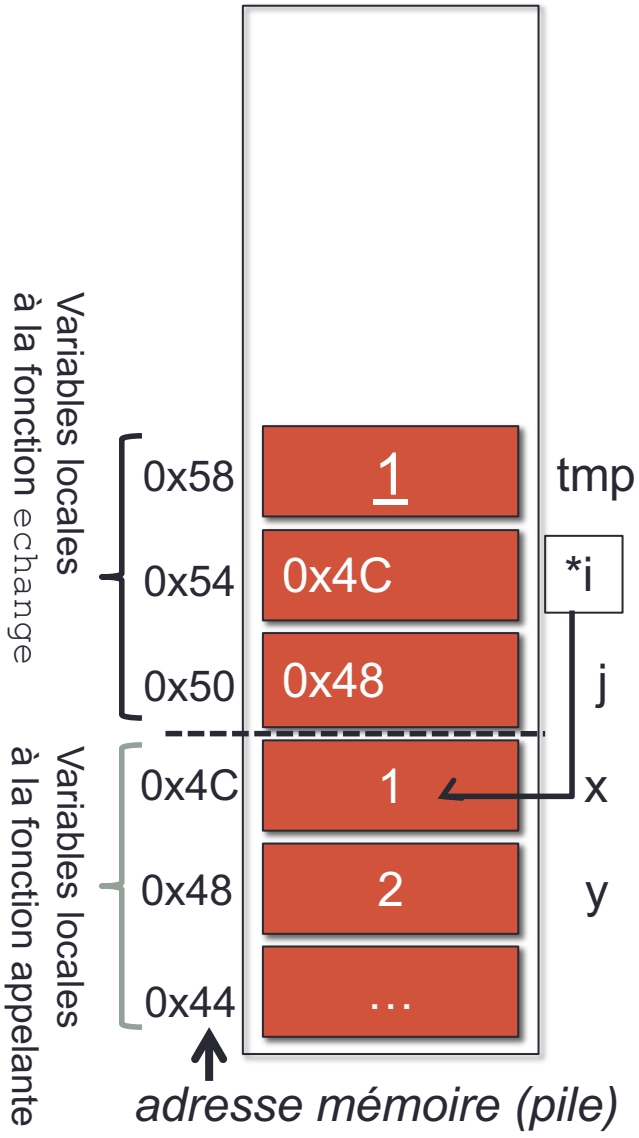


Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```

3



3. Exécution de la fonction:
   - -création variables locales
   - -exécution de l'instructions `tmp=*i`

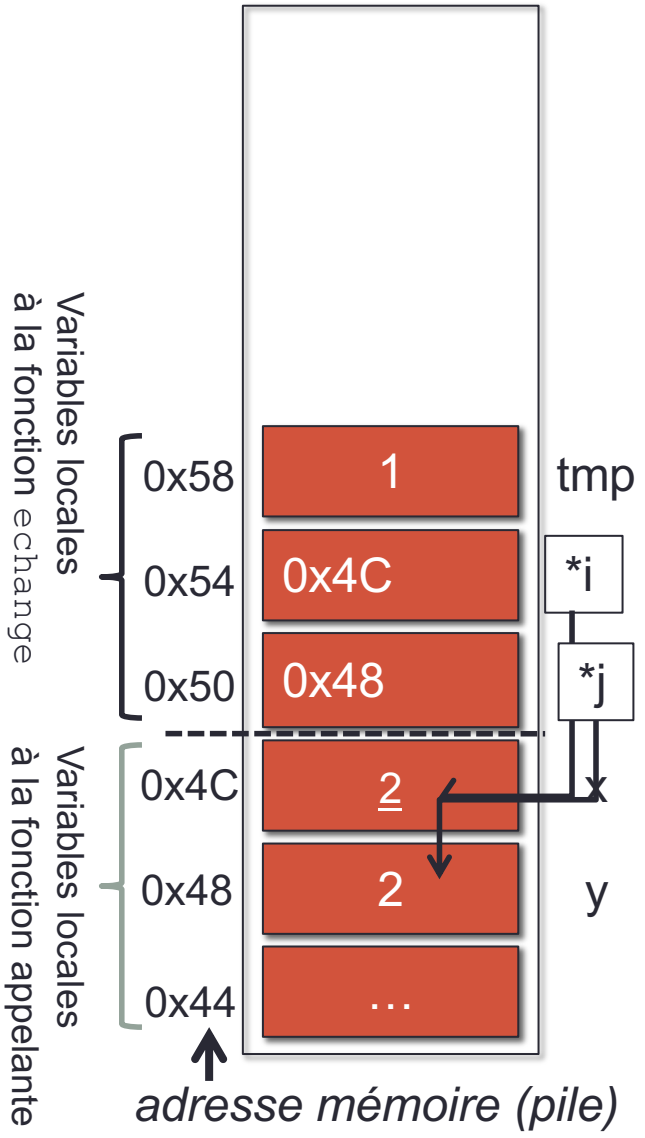
Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
        *j = tmp ;
}
```

3

3. Exécution de la fonction:
   -exécution de l'instruction `*i = *j`



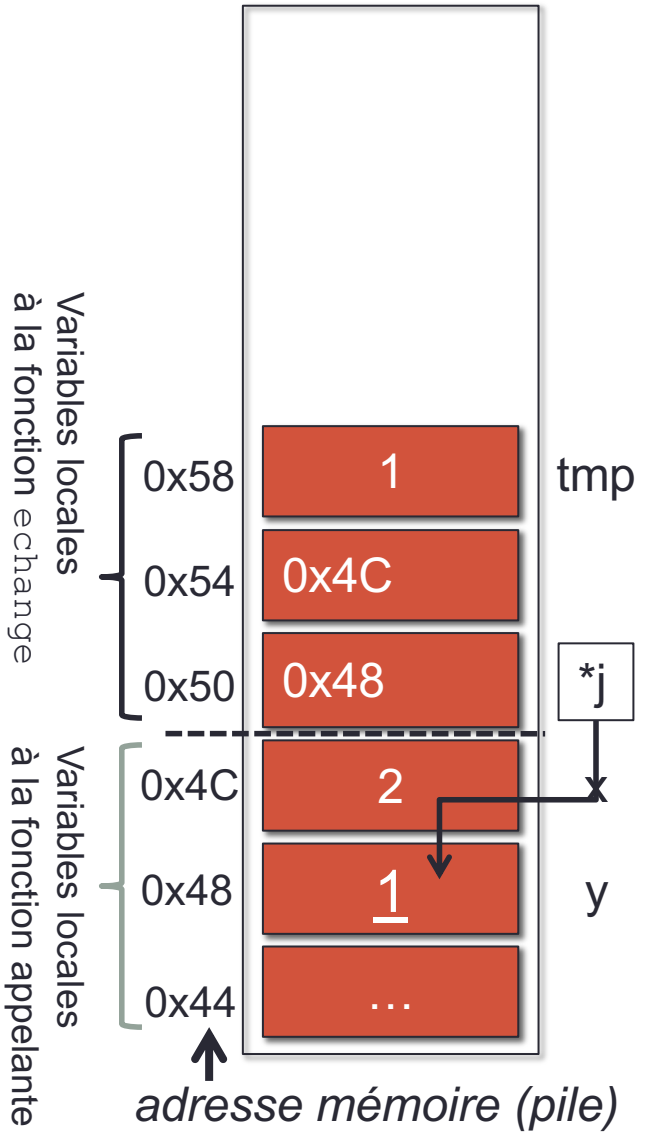
Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
        *j = tmp ;
}
```

3

3. Exécution de la fonction:
   -exécution de l'instruction `*j = tmp`



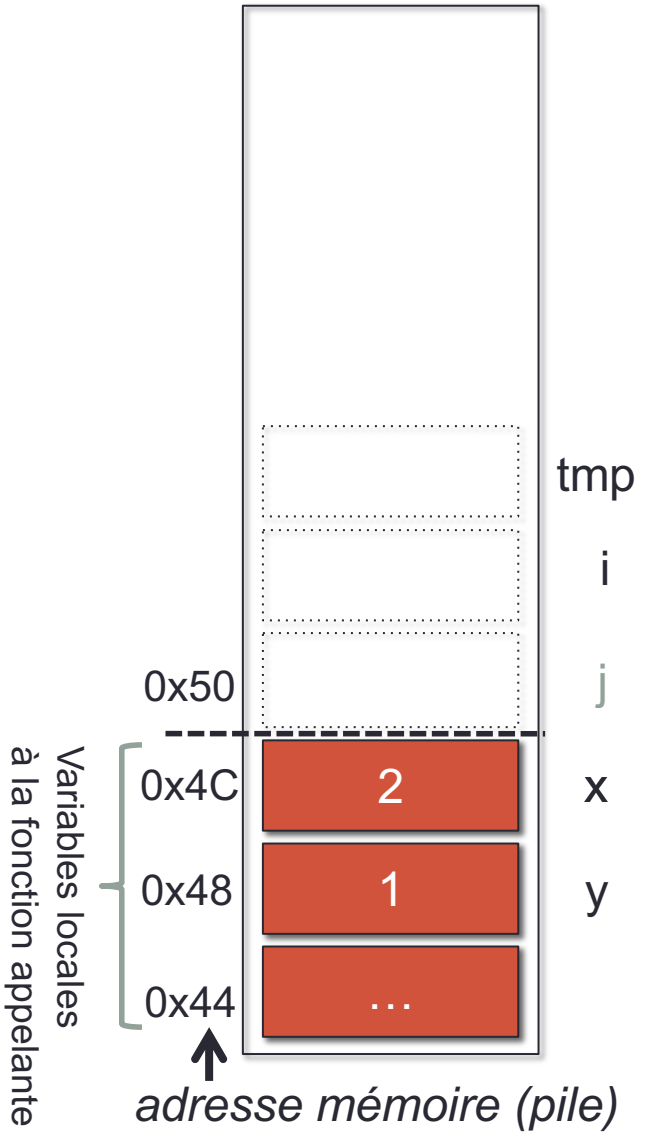
Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
        echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```

4

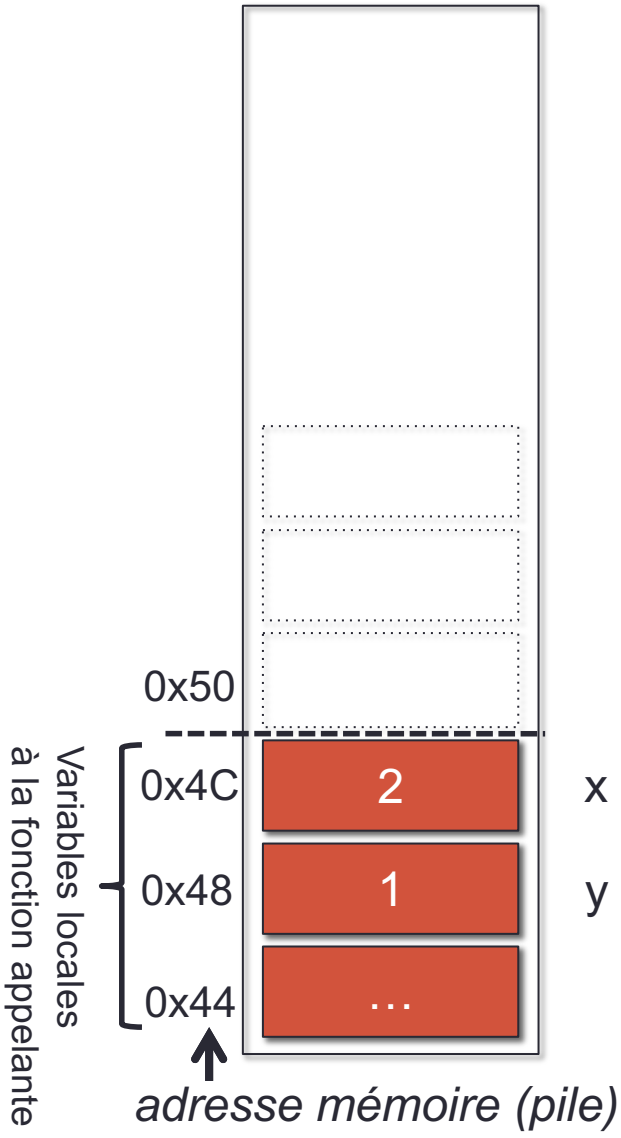
4. Sortie de la fonction :
   - suppression des variables et arguments locaux



Passage de paramètres par adresse

```
{
        int x = 1, y = 2;
        ...
5       echange(&x,&y);
        ...
}
```

```
void echange(int *i, int *j)
{
        int tmp;
        tmp = *i ;
        *i = *j ;
       *j = tmp ;
}
```



5. Retour à la fonction appelante:
   → **`x` et `y`, variables de la fonction appelante/ arguments effectifs, ont été échangés**

Passage par adresse

```
void  echange ( int *i, int *j )
{    int tmp;
     tmp = *i;
     *i = *j ;
     *j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                          x, y);
     echange(&x, &y);
     printf("apres echange:%d %d\n",
                          x, y);
     return 0;
 }
```

Passage par adresse

```
void  echange ( int *i, int *j )
{    int tmp;
     tmp = *i;
     *i = *j ;
     *j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(&x, &y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

1. affiche: `avant echange 1 2`

Passage par adresse

```
void  echange ( int *i, int *j )
{    int tmp;
     tmp = *i;
     *i = *j ;
     *j= tmp;
}

int main()
{    int x=1, y=2;

     printf("avant echange:%d %d\n",
                              x, y);
     echange(&x, &y);
     printf("apres echange:%d %d\n",
                              x, y);
     return 0;
 }
```

2. copie des adresses `&x` et `&y` dans { variables `i` et `j` / pointeurs `i` et `j` }

1. affiche: `avant echange 1 2`

Passage par adresse

```
void  echange ( int *i, int *j )
{     int tmp;
      tmp = *i;
      *i = *j ;
      *j= tmp;
}

int main()
{     int x=1, y=2;

      printf("avant echange:%d %d\n",
                              x, y);
      echange(&x, &y);
      printf("apres echange:%d %d\n",
                              x, y);
      return 0;
 }
```

2. copie des adresses `&x` et `&y` dans { variables `i` et `j` / pointeurs `i` et `j` }

3. vaut **1** : la valeur { stockée à l'adresse mémoire de `x` / de la variable pointée par `i`, i.e. `x` }

1. affiche: `avant echange 1 2`

Passage par adresse

```
void  echange ( int *i, int *j )
{     int tmp;
      tmp = *i;
      *i  = *j ;
      *j= tmp;
}

int main()
{     int x=1, y=2;

      printf("avant echange:%d %d\n",
                              x, y);
      echange(&x, &y);
      printf("apres echange:%d %d\n",
                              x, y);
      return 0;
 }
```

2. copie des adresses `&x` et `&y` dans { variables `i` et `j` / pointeurs `i` et `j`

3. vaut `1` : la valeur { stockée à l'adresse mémoire de `x` / de la variable pointée par `i`, i.e. `x`

4. affectation dans la variable { pointée par `i`: la variable `x` du `main` / d'adresse mémoire `&x`

1. affiche: `avant echange 1 2`

Passage par adresse

```
void  echange ( int *i, int *j )
{     int tmp;
      tmp = *i;
      *i  = *j ;
      *j= tmp;
}

int main()
{     int x=1, y=2;

      printf("avant echange:%d %d\n",
                              x, y);
      echange(&x, &y);
      printf("apres echange:%d %d\n",
                              x, y);
      return 0;
 }
```

2. copie des adresses `&x` et `&y` dans { variables `i` et `j` / pointeurs `i` et `j`

3. vaut **1** : la valeur { stockée à l'adresse mémoire de `x` / de la variable pointée par `i`, i.e. `x`

4. affectation dans la variable { pointée par `i`, la variable `x` du `main` / d'adresse mémoire `&x`

1. affiche: `avant echange 1 2`

4. affiche: `apres echange 2 1`

➢ valeurs de `x` et `y` échangées !!

Passage de paramètres par adresse

- Pour passer une adresse à une fonction :
  - l'adresse de la variable : `&x`
  - un pointeur, contenant l'adresse : `p_x` (avec `p_x = &x;`)

- Ainsi pour `scanf()`, qui veut un pointeur en dernier paramètre[1] :

```
int nombre;
int *p_nbr=&nombre;
scanf("%d",&nombre);   peut aussi s'écrire   scanf("%d",p_nbr);
```

[1] Passage par adresse, sinon la valeur de `nombre` ne pourrait être modifié par la fonction `scanf`!

Passage de paramètres par adresse

- pour retourner plus d'un résultat (`return` ne suffit pas):

les variables prévues pour stocker les résultats sont passées par adresse à la fonction.

Exemple: la fonction `decoupeMinute()` donne le nombre d'heures et minutes correspondant à la durée totale d'un film en minutes

```
int decoupeMinute( int* p_heures, int* p_minutes,
                                          int full_minutes)
{
   *p_heures = full_minutes/60;
   *p_minutes = full_minutes%60;
   return 0;                  /* <-- optionnel: 0=tout est OK*/
}
```

Passage de paramètres par adresse

- pour retourner plus d'un résultat (`return` ne suffit pas):

les variables prévues pour stocker les résultats sont passées par adresse à la fonction.

Exemple: la fonction `decoupeMin` ... nutes correspondant à la durée totale d'

pointeurs sur les variables du `main()` prévues pour les résultats

```
int decoupeMinute( int* p_heures, int* p_minutes,
                                          int full_minutes)
{
   *p_heures = full_minutes/60;
   *p_minutes = full_minutes%60;
   return 0;                    /* <-- optionnel: 0=tout est OK*/
}
```

Passage de paramètres par adresse

- pour retourner plus d'un résultat :

```
int decoupeMinute(int*,int*, int); /*prototype*/

int main ()
{
   int heures=0, minutes=0;
   int all_minutes=134;

   decoupeMinute(&heures, &minutes, all_minutes);

   printf("Le film dure %i h et %i\n", heures, minutes);
   return 0;
}
```

affiche : `Le film dure 2 h 14`

Passage de paramètres par adresse

- pour retourner plus d'un résultat :

```
int decoupeMinute(int*,int*, int); /*prototype*/

int main ()
{
   int heures=0, minutes=0;
   int all_minutes=134;

   decoupeMinute(&heures, &minutes, all_minutes);

   printf("Le film dure %i h et %i\n", heures, minutes);
   return 0;
}
```

variables pour les résultats

passage des adresses des variables pour les résultats

affiche : `Le film dure 2 h 14`

En résumé :

- Si la fonction doit modifier la valeur de la variable `x` appartenant à la fonction appelante (ou retourner plusieurs résultats) :

  → passage par adresse et utilisation de l'indirection

```
function (type *i,…)
{    *i = … ;
     printf("%i", *i);   }
```

appel avec: `function(&x,…)`

- Sinon, → passage par valeur.

La fonction récupère une copie de la valeur de `x` dans une variable locale `i`.
La variable `x` de la fonction appelante est protégée, sa valeur est conservée.

```
function (type i,…)
{    i = 3 ;  …      }
```

appel avec: `function(x,…)`

Sommaire chapitre 6

Les fonctions II

Tableaux : passage à une fonction

- on passe l'adresse du premier élément du tableau
  soit simplement le nom du tableau : `tab` (↔ `&tab[0]`)
  ou un pointeur dessus (allocation dynamique)

- à partir de cette adresse, la fonction a accès à tous les éléments
  car rangés en mémoire les uns à la suite des autres.

❖ souvent on a besoin de fournir la taille du tableau à la fonction

Tableaux : passage à une fonction

- Exemple 1 : fonction d'affichage d'un tableau

```
void affiche ( int *tab, int n )
 {     int i;
       for (i=0; i<n ; i++)
            {  printf("%d\n",tab[i]); }
 }

int main()
{    int u[3]={1,-2,7};
     affiche(u,3);                    /*ou affiche(&u[0],3)*/
     return(0); }
```

Tableaux : passage à une fonction

```
void affiche ( int *tab, int n )
 {     int i;
       for (i=0; i<n ; i++)
            {  printf("%d\n",tab[i]); }
 }


int main()
{    int *u;
        u = (int*)malloc(3*sizeof(int));
        u[0]=1; u[1]=-2; u[2]=7;
     affiche(u,3);
     return(0);
}
```

Tableaux : passage à une fonction

- Une variante pour les tableau 1D :

  utiliser `int tab[]` au lieu de `int *tab` dans les arguments formels

  pour souligner qu'un pointeur (ou adresse) sur un tableau est attendu.

```
void affiche( int[],int);

void affiche( int tab[], int n )
{
   int i;
      for (i=0; i<n ; i++)
          {  printf("%d\n",tab[i]); }
}
```

Tableaux : passage à une fonction

- Exemple 2 : fonction de calcul de la somme des éléments

```
double somme_elts ( double *tab, int n )
{    int i; double somme=0;
        for (i=0; i<n ; i++)
       { somme += tab[i]; }            /*ou  += *(tab+i);*/
       return (somme);
}

int main()
{    double u[3]={1,-2,7};
     double sum_u;
     sum_u=somme_elts(u,3);
     return(0); }
```

Tableaux : passage à une fonction

- Exemple 3 : fonction d'initialisation à zéro d'un tableau

```
void tab_init( double *tab, int n )
{       int i=0;
        for (i=0; i<n ; i++)
        { tab[i]=0; }
}

int main()
{    double u[3];
     tab_init(u,3);
     return(0);
}
```