Passage de paramètres

Amélie Lambert

DSP - USAL 34

Les sous-programmes

Notion de sous-programme

- **Problème :** Lorsqu'un programme est long on ne peut le coder en entier dans la fonction principale.
- **Idée :** Décomposer le programme en plusieurs sous-programmes plus petits.
- On utilise pour cela les sous-programmes (procédures ou fonctions).

Notion de sous-programme

- Un sous-programme possède un nom, des variables, des instructions, un début et une fin.

- l'exécution d'un sous-programme est invoquée directement dans la fonction `main`, ou dans un sous-programme invoqué dans le fonction `main`.

- **Définition :** Un sous-programme est un élément de programme nommé et éventuellement paramétré, que l'on définit afin de pouvoir ensuite l'appeler par son nom en affectant, s'il y a lieu, des valeurs aux paramètres.

Les sous-programmes

- **Les intérêts :**
  - le gain de place en mémoire pour le code du programme : lorsqu'un sous-programme est appelé plusieurs fois, dans une boucle,
  - les notions d'abstraction et de modularité : utiliser un appel de sous-programme permet de d'écrire une application en faisant abstraction (en dissimulant) les détails de la fonction que réalise ce sous-programme.

- **Quelques règles :**
  - Un sous-programme doit être homogène : il doit réaliser une tâche précise, formant un tout.
  - Il doit être de taille "raisonnable". La compréhension et la gestion du programme en dépend.

Les sous-programmes

- La définition du sous-programme est composée d'une spécification qui indique son nom, ses paramètres (ou arguments) avec leurs caractéristiques (nom, type) et d'un corps comprenant éventuellement des déclarations d'objets locaux au sous-programme et les instructions à exécuter.
- Déclaration d'un sous-programme en C :

```
typeRetour nomFonction (typeAr1 arg1, typeAr2 arg2, ...)
```

Exemple en C :

```
int perimetre(int largeur, int longueur)
```

Les types de sous-programmes

Types de sous-programmes

Il existe deux types de sous-programmes :

- **Les procédures :** sous-programme nommé, ne renvoyant pas de résultat, et modifiant généralement son environnement.
- **Les fonctions :** sous-programme nommé, renvoyant une valeur.

Les procédures

Les procédures : sous-programme nommé, ne renvoyant pas de résultat, et modifiant généralement son environnement.

Exemple :

```
#include "inout.h"

void afficher(int i) {
  EcrireInt(i);
}

void main()
{
  afficher(3);
}
```

L'appel d'une procédure constitue une instruction en lui-même

Les fonctions

Les fonctions : sous-programme nommé, renvoyant une valeur.

Exemple :

```
int carre(int n) {
  return n*n;
}

void main()
{
  int n;
  n = carre(2);
  afficher(n);
}
```

Les fonctions

Les fonctions exécutent des instructions et en plus retournent une valeur.

L'appel d'une fonction est remplacé dans le programme appelant par la valeur renvoyée.

L'appel d'une fonction représente une valeur qui doit être utilisée à l'intérieur d'une instruction.

Appel de sous-programme

Appel de sous-programmes

```
void afficher(int e) {
  ecrireInt(e);
}
```

La procédure `afficher` affiche l'entier e qui lui est passé en paramètre.

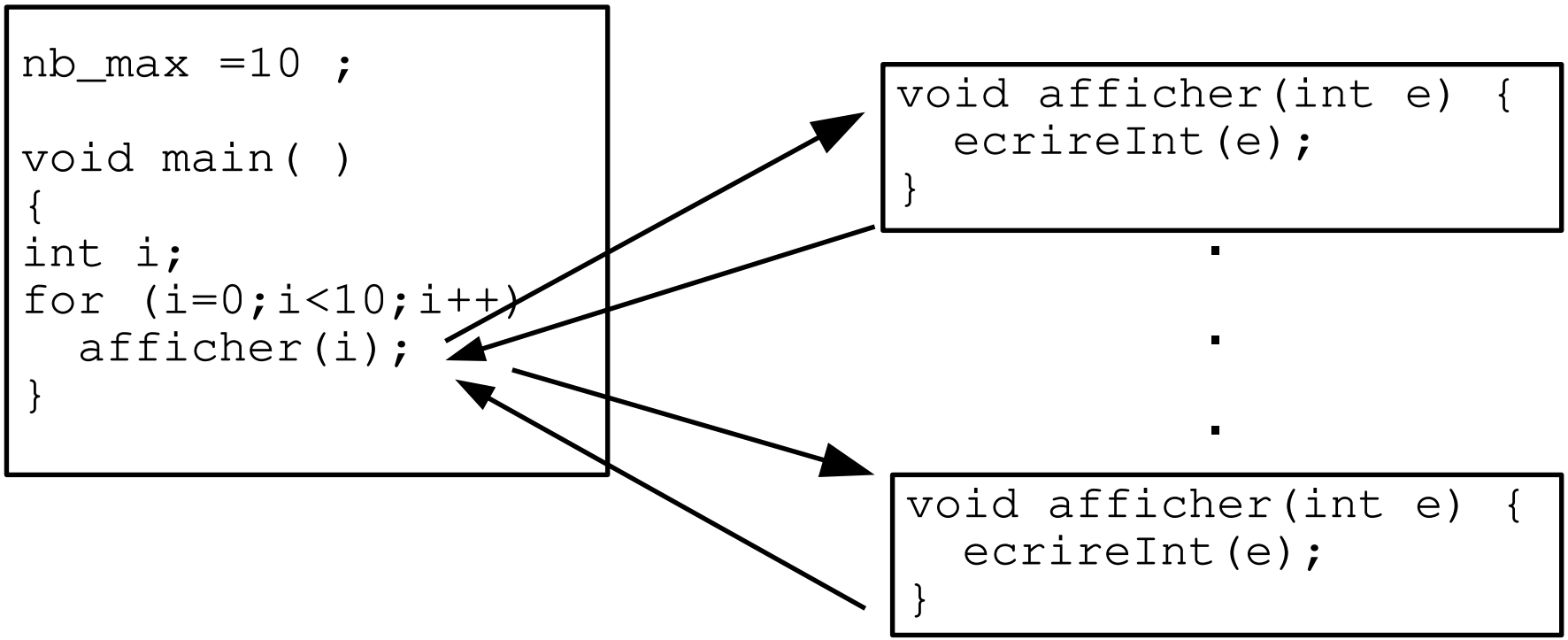
```
int nb_max=10;

void main()
{
  int i;
  for (i=0;i<nb_max;i++)
    afficher(i);
}
```

La fonction principale `main` appelle la procédure `afficher` 10 fois, pour les valeurs successives 0, 1, 2, ..., 9.

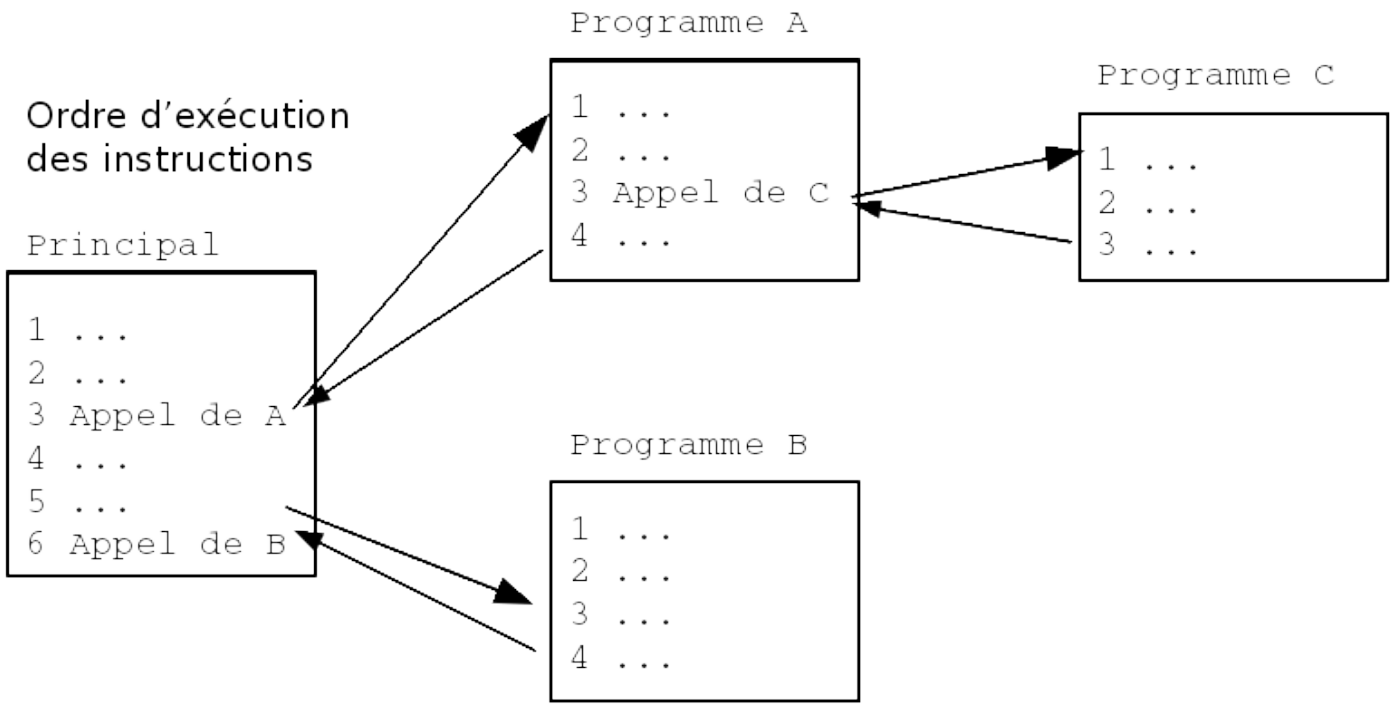
Appel sous-programmes

La procédure `afficher` est appelée 10 fois dans la boucle du programme principal :



Appel sous-programmes en cascade

Un sous-programme peut lui-même appeler un autre sous programme.
A chaque fin de sous-programme, on revient à l'exécution de l'instruction suivante dans le programme appelant.



Communication entre programme appellant et sous-programmes

Les données et les résultats d'un sous-programme proviennent d'une communication entre le programme appelant et le sous-programme.

3 modes de communication sont possibles :

- l'utilisation de variables communes à l'appelant et à l'appelé : ce sont les **variables globales**.
- **le passage de paramètres**, pour les procédures et les fonctions
- **la valeur de retour** pour les fonctions

Les variables locales ou globales

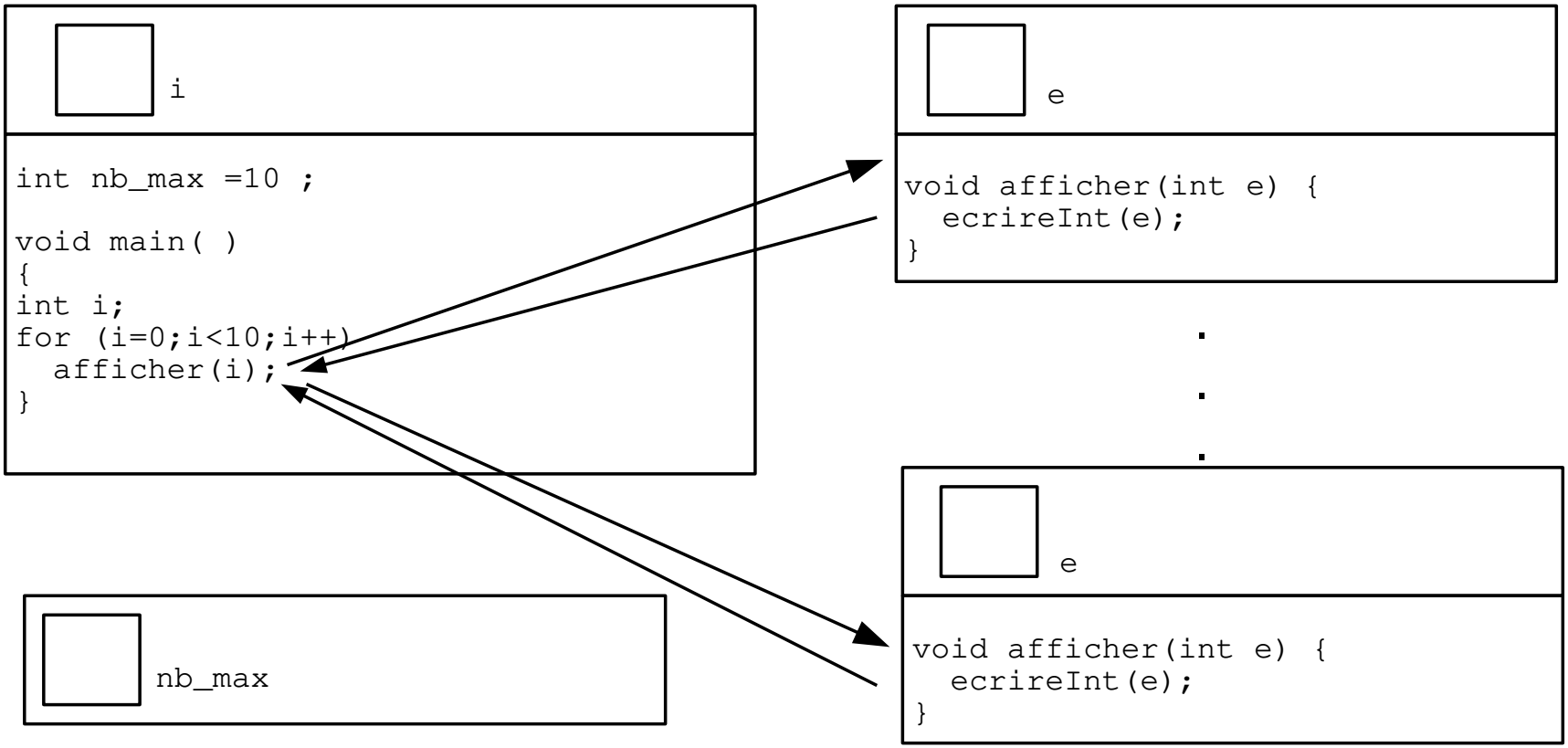
L'endroit où est déclaré une variable est très important car il détermine dans quel(s) sous- programme(s) elle va pouvoir être utilisée.

- **Une variable locale** est déclaré dans un sous-programme et n'est utilisable que dans le sous-programme où elle a été déclarée.

- **Une variable globale** est déclarée à l'extérieur du programme principal et des sous-programmes : elle est commune à l'ensemble des sous-programmes et du programme principal et est utilisable partout.

Les variables locales ou globales

Chaque procédure qui est appellée possède son propre espace d'adressage :



Passage de paramètre

Appel d'un sous-programme et passage de paramètre

L'appel d'un sous-programme se fait en mentionnant son nom, suivi des paramètres effectifs figurant entre parenthèses et séparés par des virgules.

Il existe deux mécanismes importants de substitution entre paramètres effectifs et paramètres formels :

- le passage **par valeurs,**
- le passage **par référence.**

Le passage de paramètre par valeur

La valeur du paramètre effectif est recopiée dans le paramètre formel à l'entrée du sous-programme. Le sous-programme travaille sur une copie du paramètre effectif. Celui-ci n'est pas modifié à l'issue du sous-programme.

Le passage de paramètre par valeur : Exemple

```
void echanger(int x, int y){
   int temp;
   temp=x;
   x=y;
   y=temp;
}
```

```
void main( ){
int a=3;
int b=5;
ecrireString("avant a=");
ecrireInt(a); ecrireString("b=");
ecrireInt(b);
echanger(a,b);
ecrireString("avant a=");
ecrireInt(a); ecrireString("b=");
ecrireInt(b);
}
```

La procédure `echanger` échange l'entier `x` et l'entier `y` qui lui sont passé en paramètre.

La fonction principale `main` appelle la procédure `echanger` sur les entiers `x` et `y`.

Le passage de paramètre par valeur : Exemple

Le résultat de l'exécution du programme est l'affichage suivant :

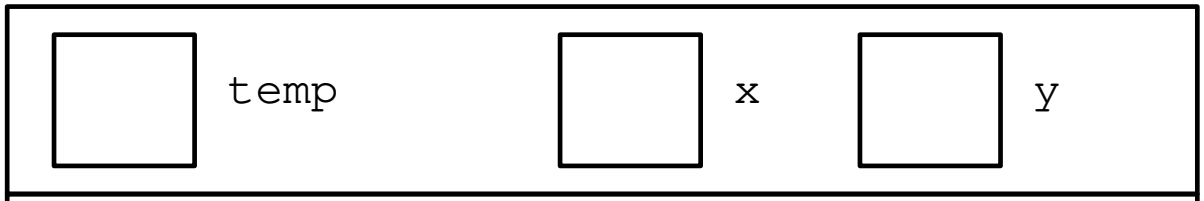
```
avant échange, a = 3, b= 5
après échange, a = 3, b= 5
```

L'appel à la procédure `echanger(a,b)` est un appel par valeur, les paramètres formels `x` et `y` sont remplacés par `a=3` et `b=5`, elle reçoit des copies des paramètres `a` et `b`.

Le passage de paramètre par valeur : Exemple

L'état de la mémoire avant le début de l'exécution du programme :

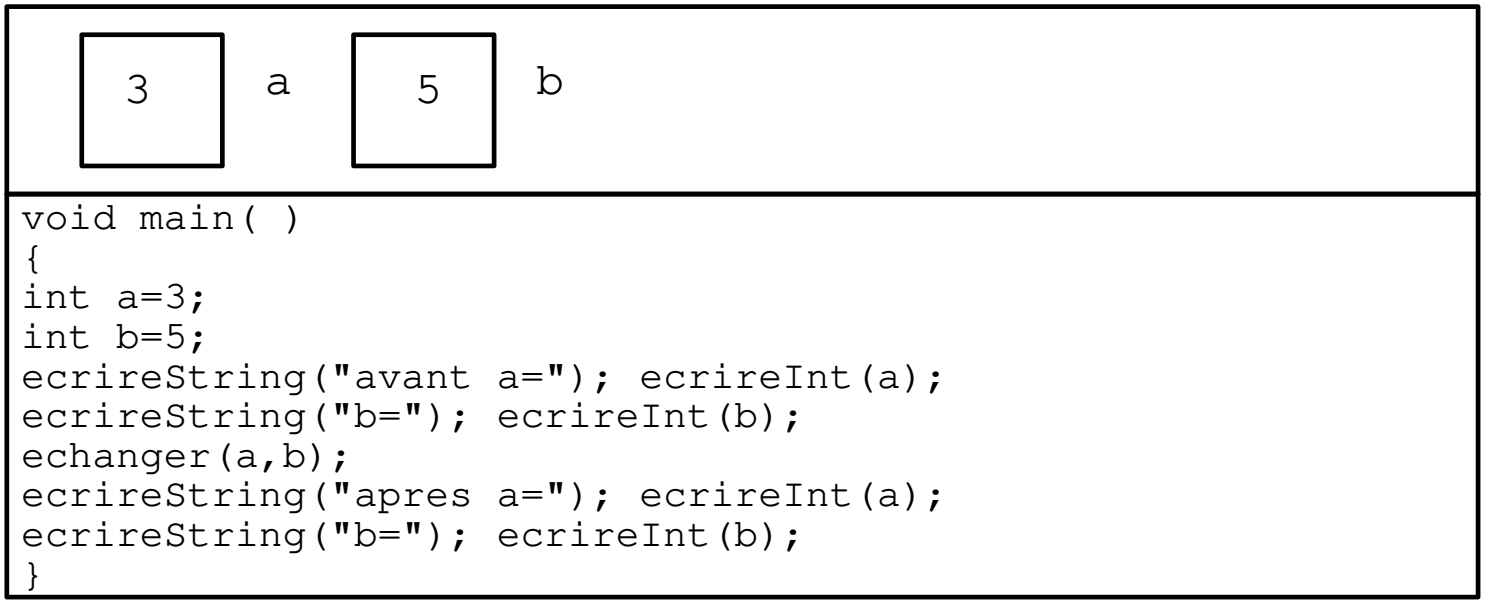

```
void main( )
{
int a=3;
int b=5;
ecrireString("avant a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
echanger(a,b);
ecrireString("apres a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
}
```



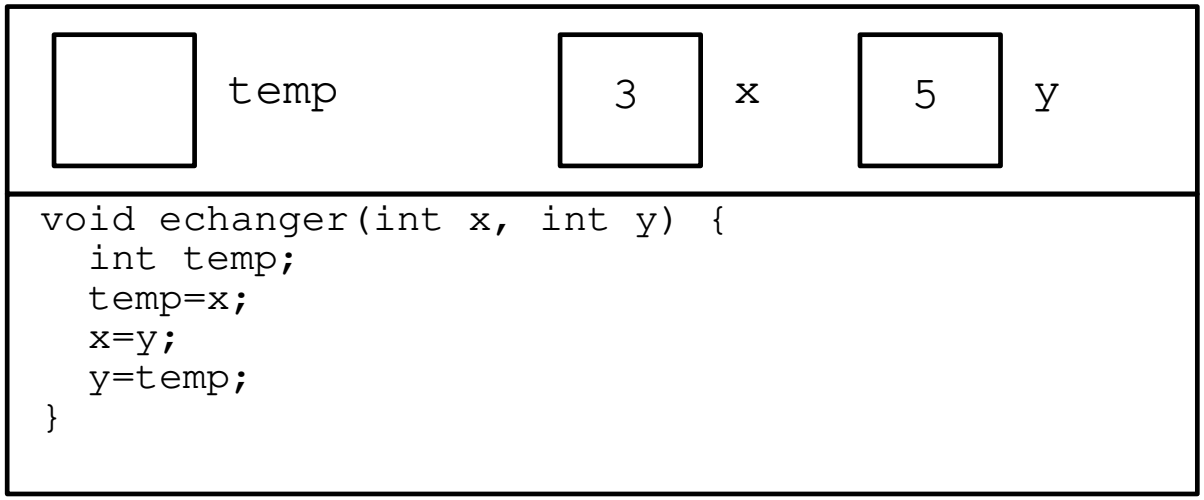

```
void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

On commence par exécuter les instructions de la fonction `main` :



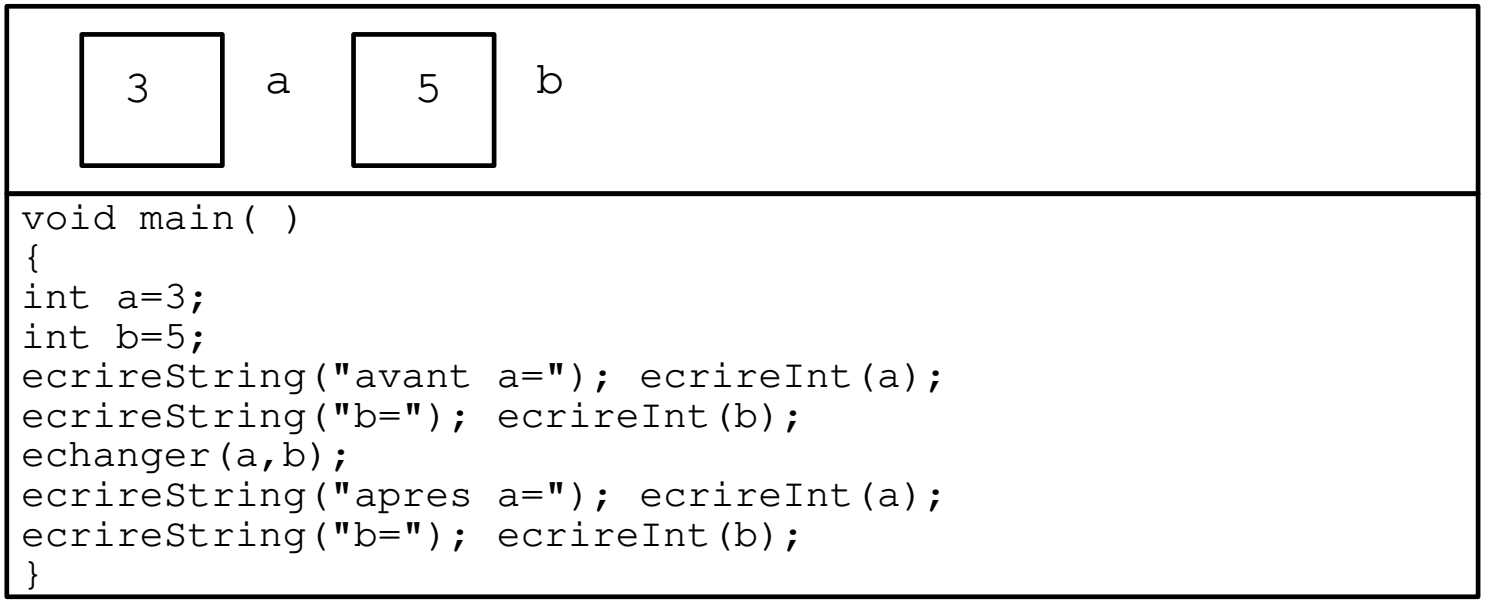

```
void main( )
{
int a=3;
int b=5;
ecrireString("avant a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
echanger(a,b);
ecrireString("apres a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
}
```



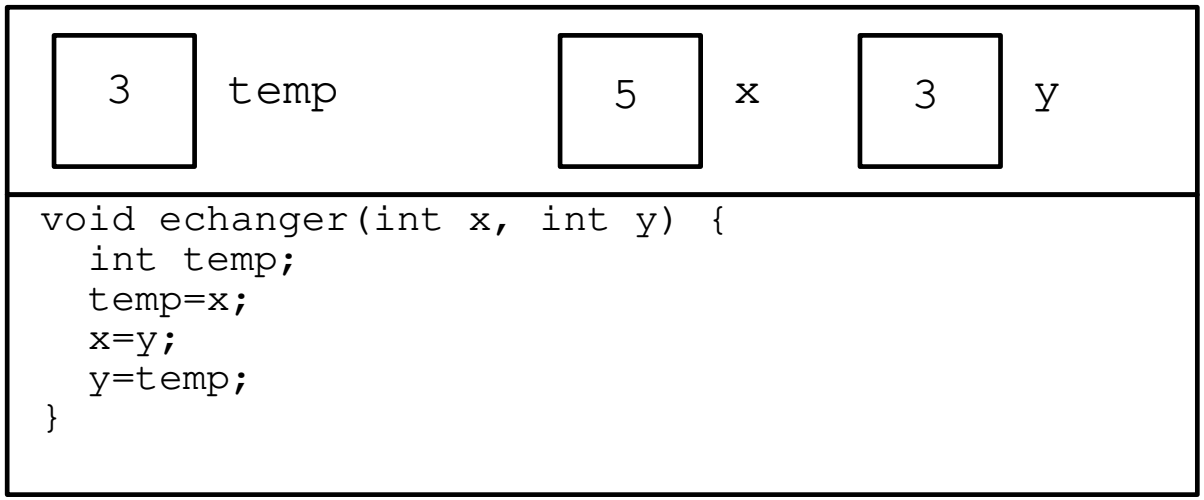

```
void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

On exécute la procédure `echanger` :



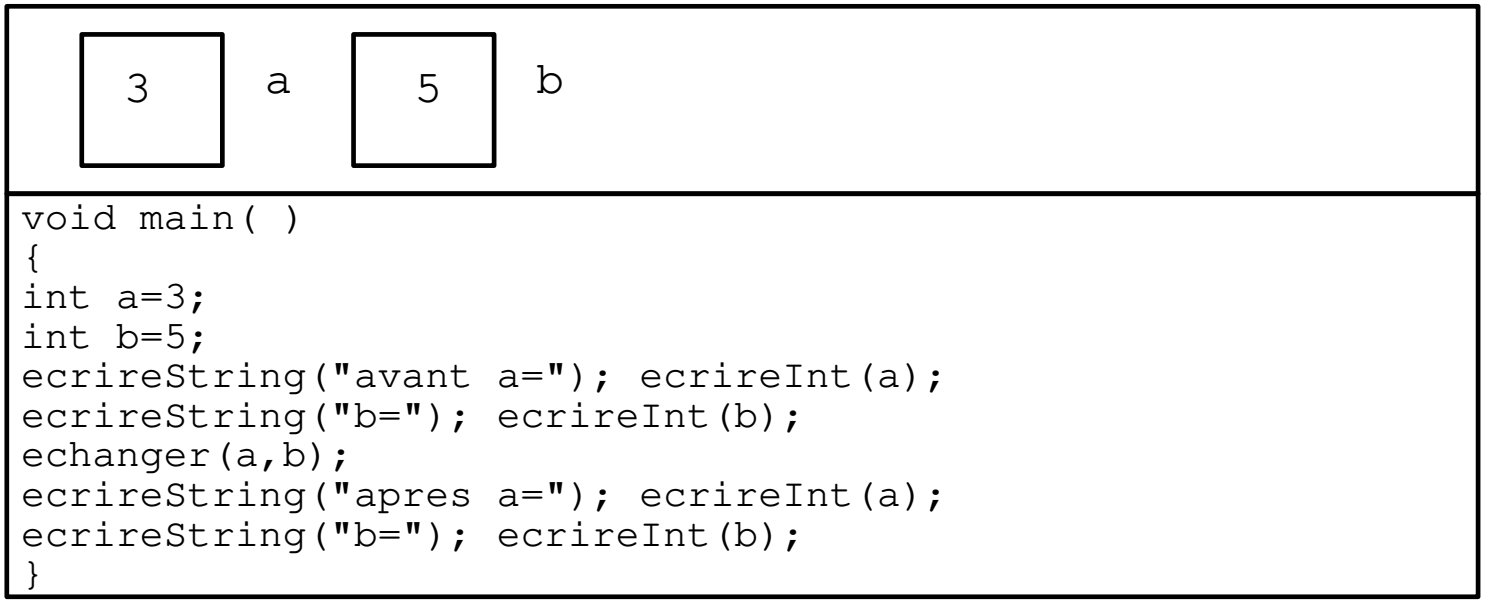

```
void main( )
{
int a=3;
int b=5;
ecrireString("avant a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
echanger(a,b);
ecrireString("apres a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
}
```



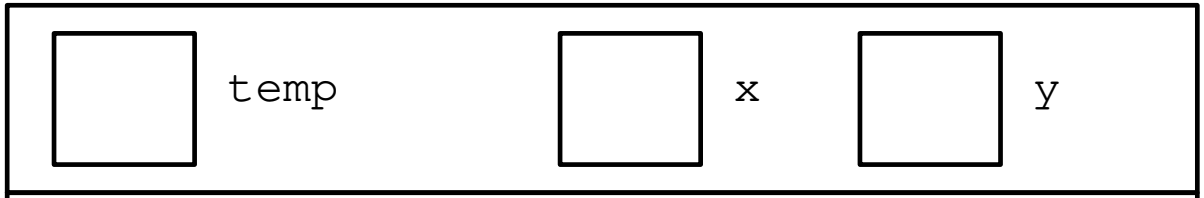

```
void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par valeur : Exemple

Après l'exécution de la procédure `echanger` :




```
void main( )
{
int a=3;
int b=5;
ecrireString("avant a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
echanger(a,b);
ecrireString("apres a="); ecrireInt(a);
ecrireString("b="); ecrireInt(b);
}
```




```
void echanger(int x, int y) {
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Le passage de paramètre par référence

Définition : une variable de type référence est une variable dans laquelle est stockée une adresse.

L'adresse du paramètre effectif est communiquée au sous-programme qui travaille alors directement sur la variable passée en paramètre et non sur une copie locale. Le sous-programme peut alors modifier la valeur d'une variable du programme qui l'a appelé.

Lorsqu'un tableau est passé en paramètre, c'est sa référence qui est en fait passée (i.e. son adresse), le tableau d'origine et celui vu par la procédure occupent donc le même espace mémoire.

Le passage de paramètre par référence : Exemple

La procédure `echangerTab` échange les élements des d'indices `i` et `j` du tableau `t` qui lui sont passé en paramètre.

```
echangerTab(int t[], int i, int j) {
   int temp;
   temp = t[i];
   t[i]=t[j];
   t[j]=temp;
}
```

La fonction principale `main` appelle la procédure `echangerTab` sur le tableau `t` et les indices `1` et `3`.

```
void main( ) {
   int t[]= {3, 4, 24};
   ecrireString("avant tab=");
   ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
   echangerTab(tab,1,2);
   ecrireString("avant tab=");
   ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
}
```

Le passage de paramètre par reférence : Exemple

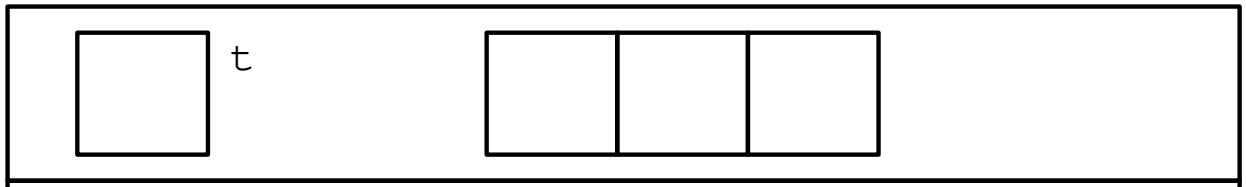
Le résultat de l'exécution du programme est l'affichage suivant :

```
avant tab=3,4,24
apres tab=3,24,4
```

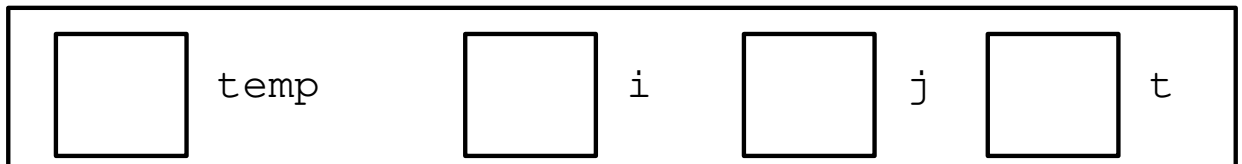
L'appel à la procédure `echangerTab(tab,1,2)` est un appel par valeur, mais la valeur de tab est sa référence, les paramètres formels `t`, `i` et `j` sont remplacés par `ref t`, `i=1` et `j=4`, elle reçoit des copies de ces paramètres.

Le passage de paramètre par référence : Exemple

L'état de la mémoire avant le début de l'exécution du programme :



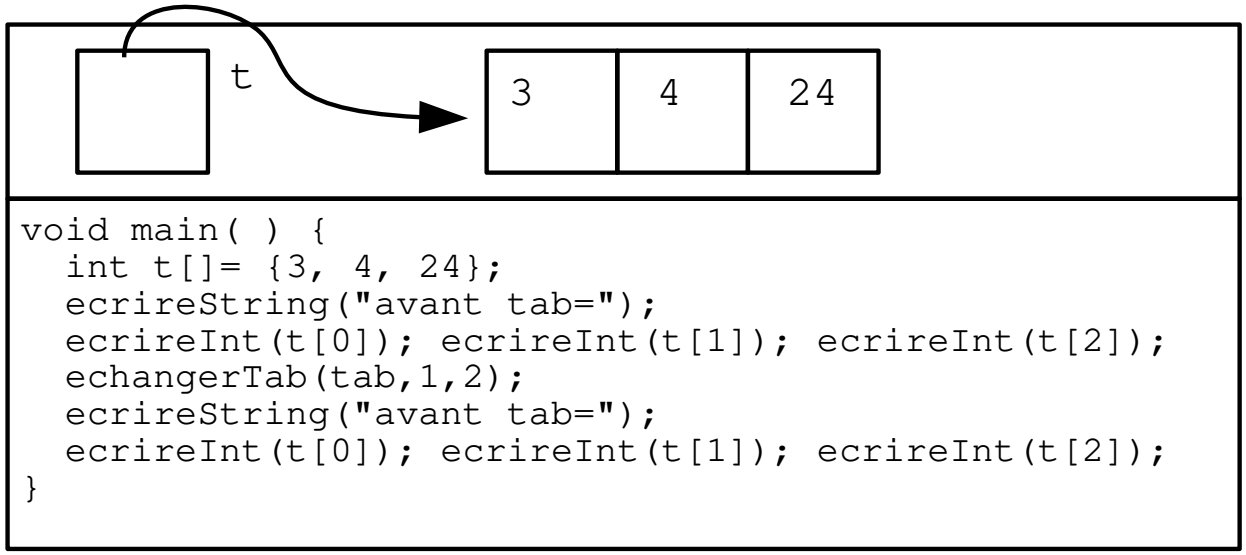

```
void main( ) {
  int t[]= {3, 4, 24};
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
  echangerTab(tab,1,2);
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
}
```



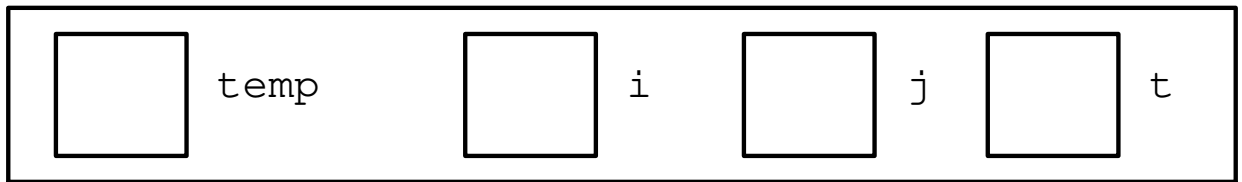

```
void echangerTab(int t[],int i,int j)
{
   int temp;
   temp = t[i];
   t[i]=t[j];
   t[j]=temp;
}
```

Le passage de paramètre par référence : Exemple

On commence par exécuter les instructions de la fonction `main` :



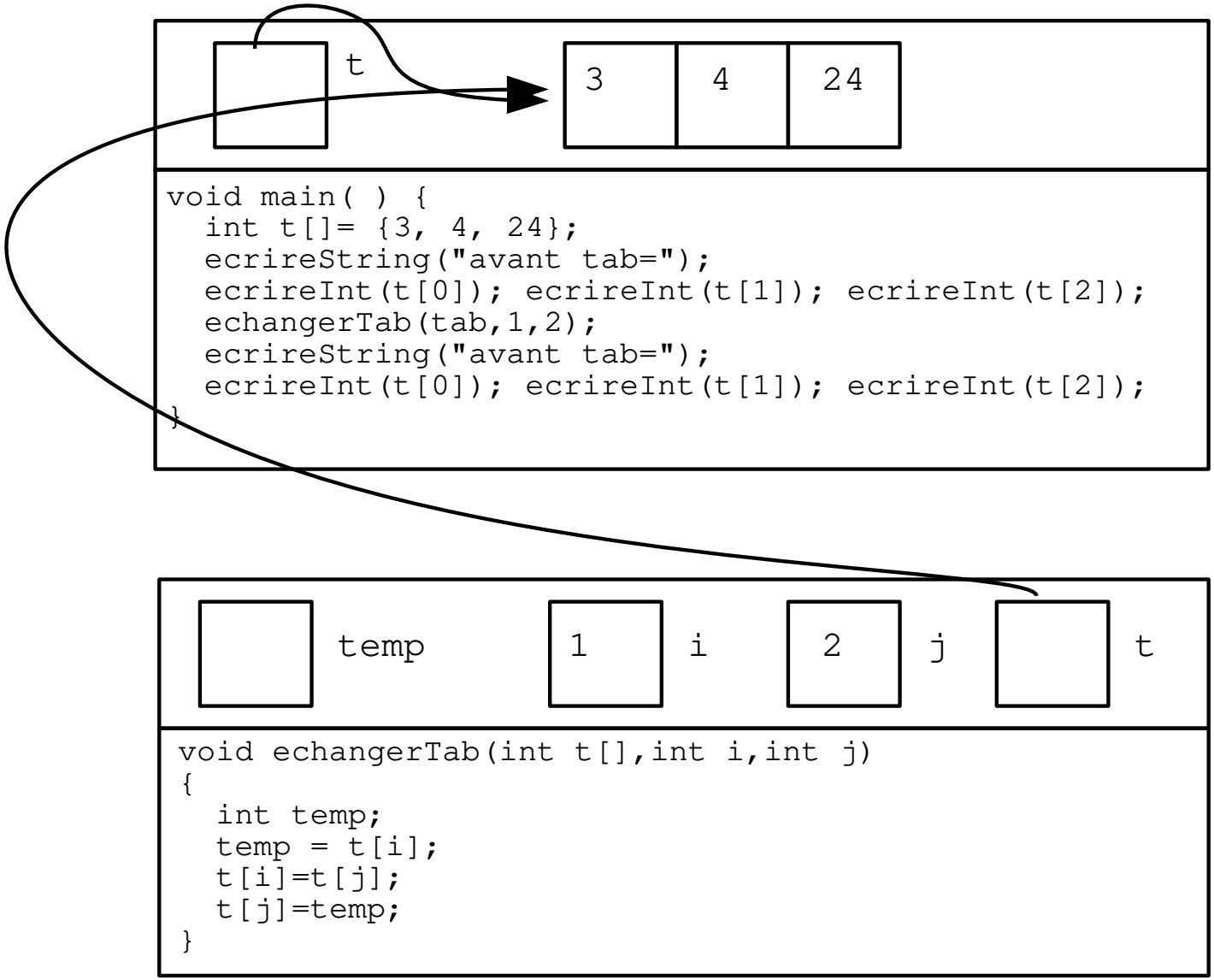

```
void main( ) {
  int t[]= {3, 4, 24};
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
  echangerTab(tab,1,2);
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
}
```




```
void echangerTab(int t[],int i,int j)
{
  int temp;
  temp = t[i];
  t[i]=t[j];
  t[j]=temp;
}
```

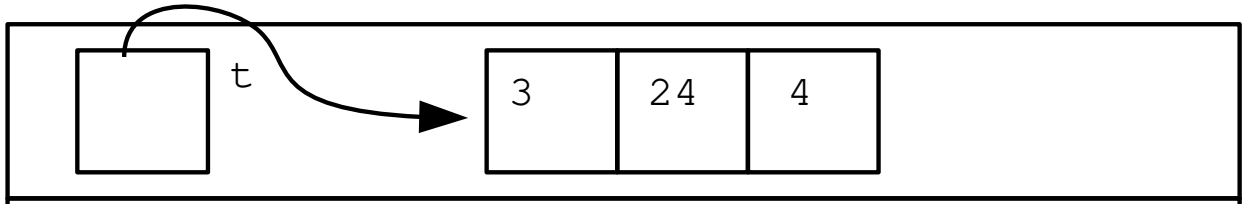
Le passage de paramètre par référence : Exemple

On exécute la procédure `echangerTab` :

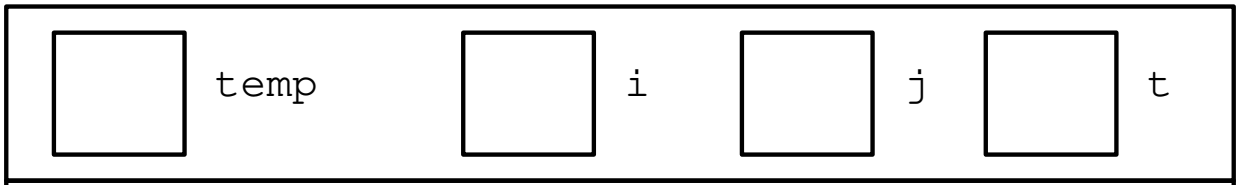


Le passage de paramètre par référence : Exemple

Après l'exécution de la procédure `echangerTab` :




```
void main( ) {
  int t[]= {3, 4, 24};
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
  echangerTab(tab,1,2);
  ecrireString("avant tab=");
  ecrireInt(t[0]); ecrireInt(t[1]); ecrireInt(t[2]);
}
```




```
void echangerTab(int t[],int i,int j)
{
   int temp;
   temp = t[i];
   t[i]=t[j];
   t[j]=temp;
}
```