Haute École Léonard de Vinci
SCIENCES ET TECHNIQUES

I2011
Langage C : bases

Chap. 7
Fonctions

Anthony Legrand
Jérôme Plumat

Spécificités des fonctions C

Programmation procédurale

- Fonction
- Appel direct
- Données du contexte appelant
- Ex:
  Pascal, BASIC, C, Fortran

≠

Programmation orientée objet

- Méthode
- Appel sur un objet
- Données de l'objet courant
- Ex:
  PHP, C++, Java, Python

Structure d'un programme C

- Choix 1:

```
[ inclusion de fichiers headers: #include ]

[ autres directives au préprocesseur: #define #ifndef … ]

[ définition de types utilisateur: typedef ]

[ déclaration de variables globales, externes ou statiques ]

[ définition de fonctions ]

[ programme principal: main() ]
```

Structure d'un programme C

- Choix 2:

```
[ inclusion de fichiers headers: #include ]

[ autres directives au préprocesseur: #define #ifndef … ]

[ définition de types utilisateur: typedef ]

[ déclaration de variables globales, externes ou statiques ]

[ déclaration de fonctions (prototypes) ]

[ programme principal: main() ]

[ définition de fonctions ]
```

Déclarer une fonction

- Déclarer le **prototype** d'une fonction :

```
type fct (type1 param1, type2 param2, … );
```

- Exemple

```
int search (int* t, int sz, int e);
```

- Rem: seuls les types importent dans un prototype (les identifiants de paramètres sont optionnels)

Définir une fonction

- Type de retour + liste de paramètres + code
- Exemple

```
int search (int* t, int sz, int e) {
    int i = 0;
    while (i < sz && t[i] != e) {
        i++;
    }
    return i;
}
```

Type d'une fonction

- Une fonction retourne une **valeur** ou rien (**void**)
- Exemples

```
double sum (double a, double b) {
    return (a+b);
}

void itoa (int a) {
    printf("%d", a);
    return;   // (optionnel)
}
```

Type d'une fonction

- C n'exige pas de **`return`** mais le résultat peut dans ce cas être **indéterminé** !

- Exemple

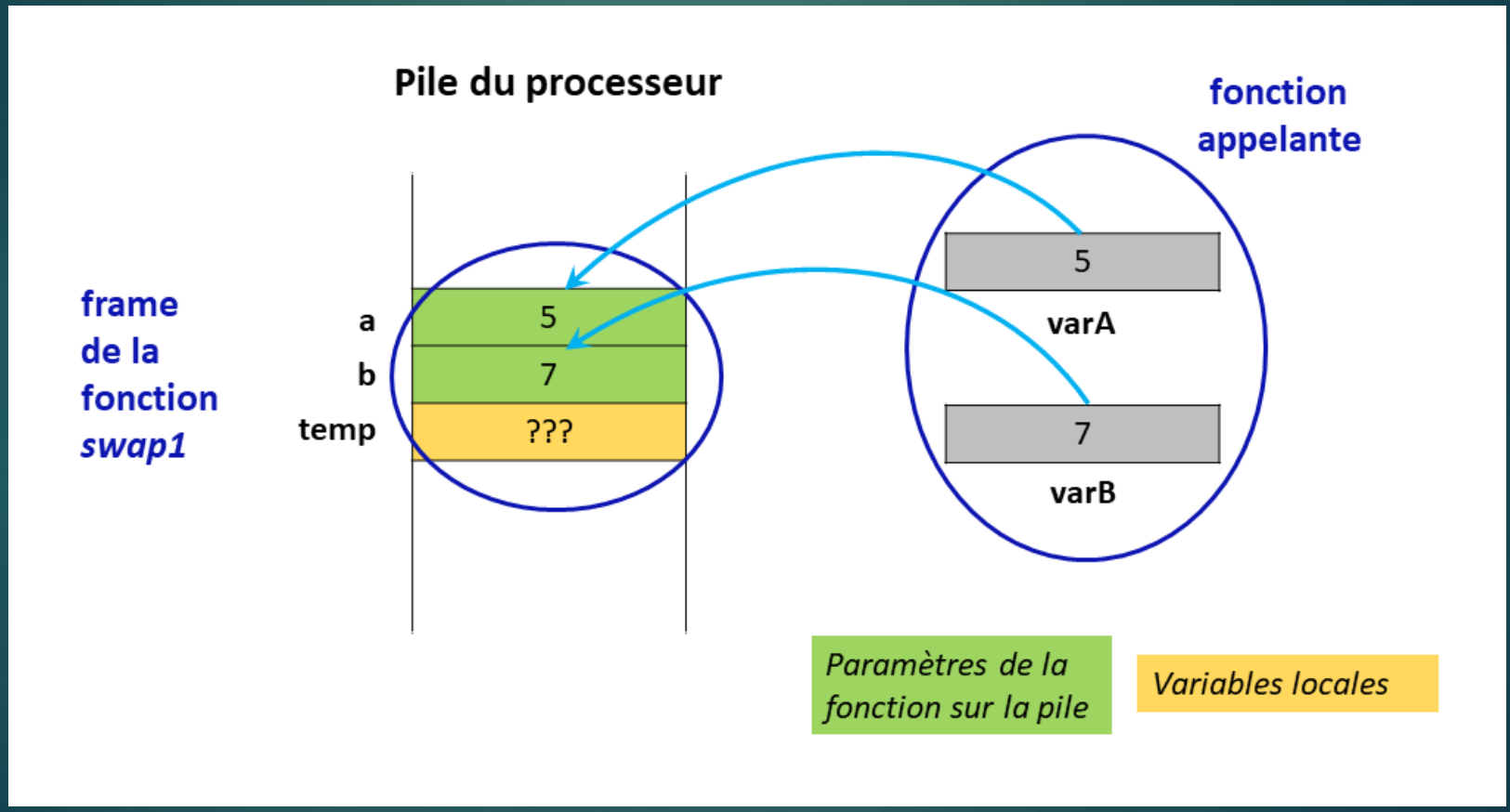
```
int divint (int a, int b) {
    if (b != 0) {
        return (a/b);
    }
}
```

Permuter deux valeurs

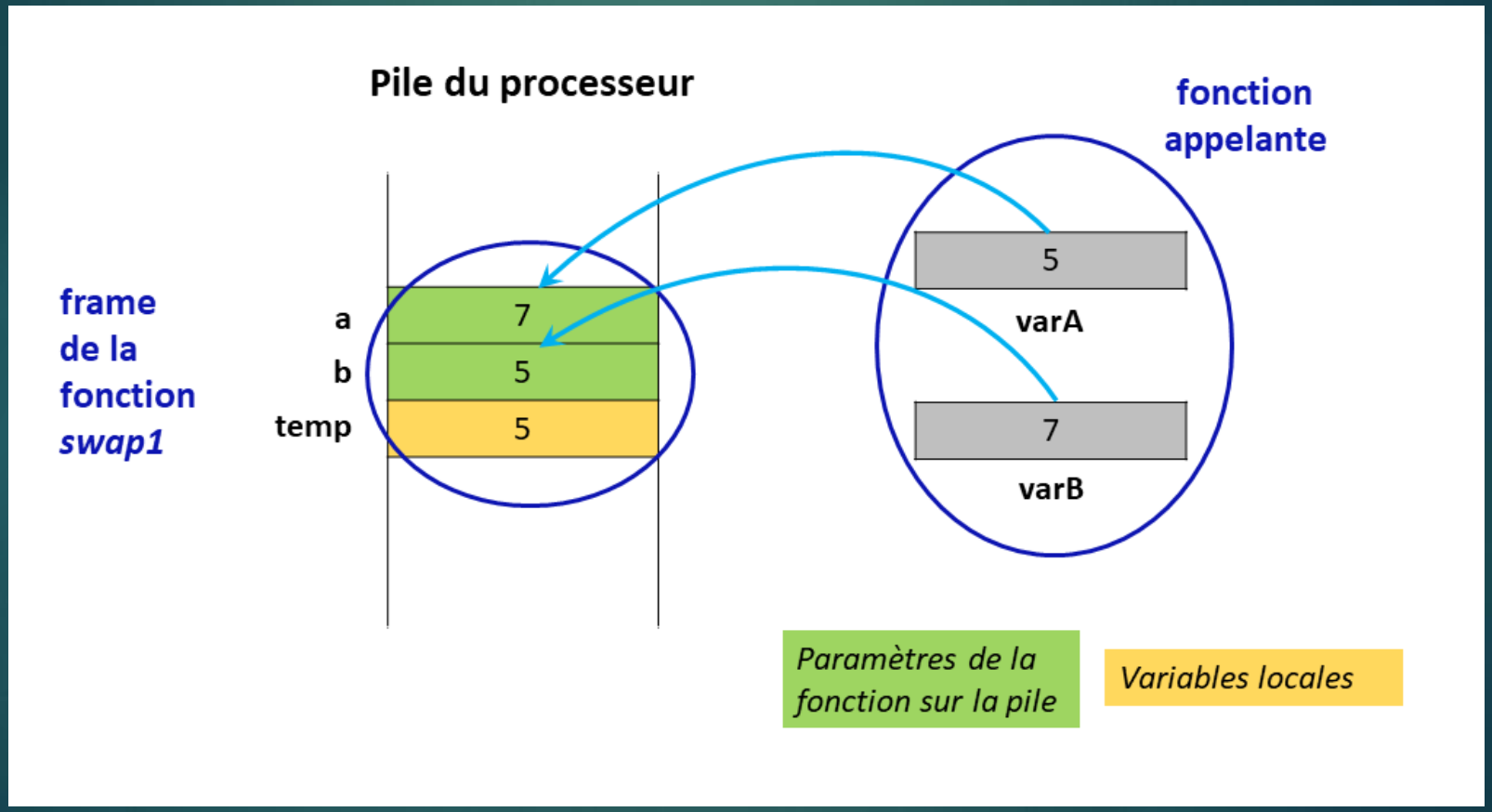

```
void swap1 (int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
int main () {
    int varA = 5, varB = 7;
    swap1(varA, varB);
    printf("varA=%d varB=%d", varA, varB);
}
```

Paramètres

Seul le passage par **valeur** est défini en C!

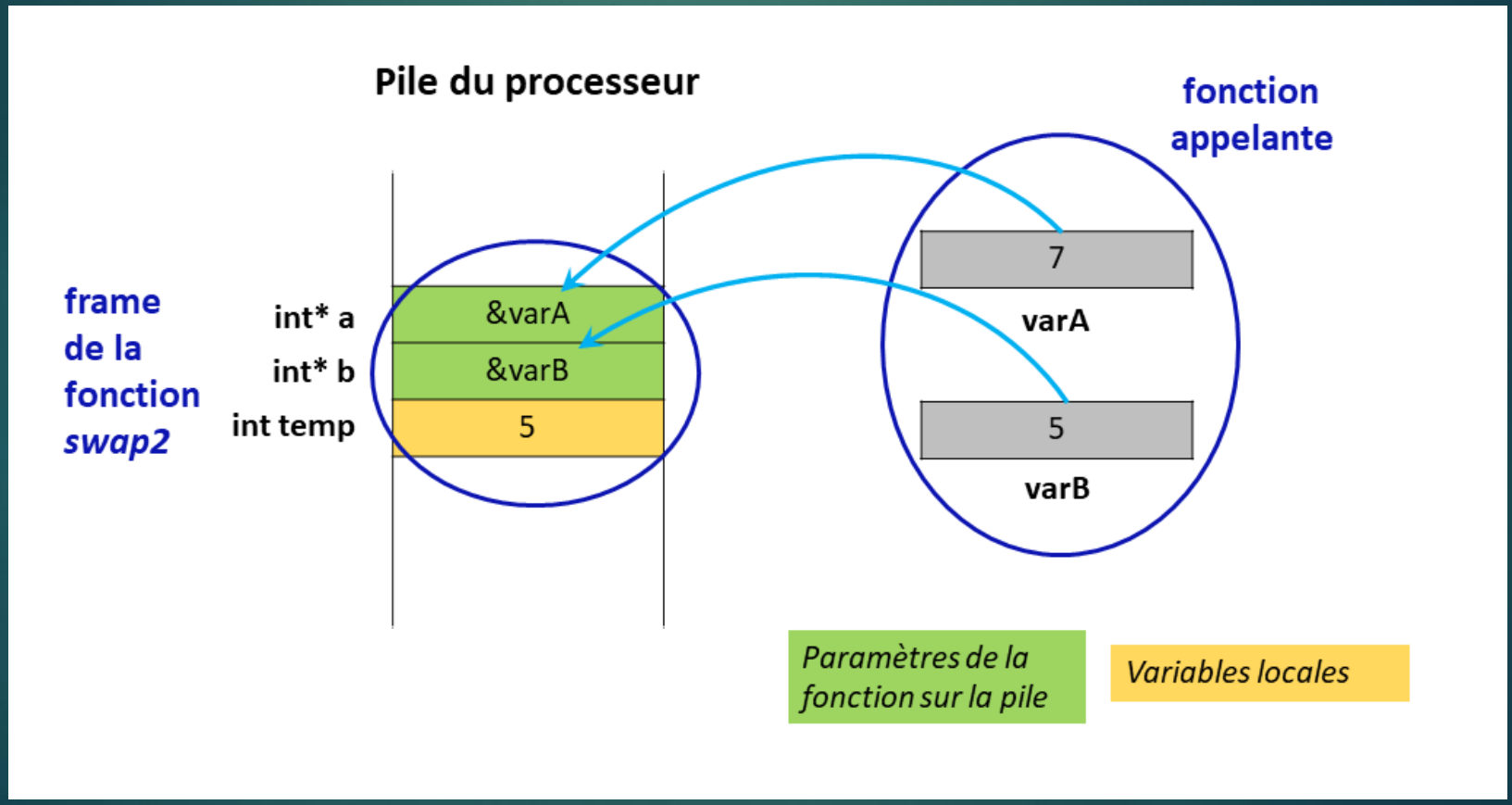


swap1 est **incorrect**



Paramètres et pointeurs

Passage par **valeur**, mais on peut passer des **pointeurs** !



Permuter deux valeurs (correct)


```
void swap2 (int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main () {
    int varA = 5, varB = 7;
    swap2(&varA, &varB);
    printf("varA=%d varB=%d", varA, varB);
}
```

Paramètres et pointeurs

- En C, pour qu'une fonction **modifie** la valeur d'un de ses paramètres, il faut lui fournir **l'adresse de ce paramètre**.

Fonctions et tableaux

- C convertit systématiquement les **tableaux** passés en paramètres en **pointeurs**

- Exemple

```
void initTable (int *tab, int taille) {
  for (int i=0; i<taille; i++) {
    tab[i] = i;
  }
}
```

Fonctions et tableaux

- Une fonction peut renvoyer un **tableau** mais il doit être alloué **dynamiquement**

```
int* creerTable (void) {
   int tab[TAILLE];
   for (int i=0; i<TAILLE; i++)
      tab[i] = i;
   return tab;
}
...
int* t = creerTable();
```

Fonctions et tableaux

- Une fonction peut renvoyer un **tableau** mais il doit être alloué **dynamiquement**

```
int* creerTable (int taille) {
  int* tab = (int*)malloc(taille*sizeof(int));
  for (int i=0; i<taille; i++)
     tab[i] = i;
  return tab;
}
...
int* t = creerTable(10);
```

Spécifications

- Les spécifications de fonctions sont des règles qui déterminent quels sont les **conditions** d'utilisation et les **résultats** de cette fonction.
- Ces règles forment un **contrat** qui précise les **responsabilités** entre le client (le programme appelant) et le fournisseur d'un morceau de code logiciel (la fonction).

Spécifications

- **Précondition:** phrase logique qui décrit les hypothèses sur l'état du programme au moment d'appeler la fonction.

  Si la précondition n'est pas vérifiée, le résultat de la fonction est indéterminé.

- **Postcondition**: phrase logique qui décrit comment la fonction a modifié l'état du programme.

- **Résultat:** le résultat renvoyé par la fonction.

Spécifications

```
int search (int* t, int sz, int e);
```

- **PRE:** `t` : tableau non-null de longueur `sz`
- **POST:** `t` n'est pas modifié
- **RES:** Si `e` est un élément de `t`, alors un entier `idx` est renvoyé de telle sorte que `t[idx] == e` ; sinon `-1` est renvoyé.

Spécifications

```
void print (int* t, int sz);
```

- **PRE:** `t` : tableau non-null de longueur `sz`
- **POST:** `t` n'est pas modifié et une ligne représentant le contenu de `t` est affichée sur stdout

Spécifications

Par défaut, on supposera que:

- **(PRE) tous les paramètres pointeurs sont non-nuls**
- **(POST) les paramètres ne sont pas modifiés**

```
void print (int* t, int sz);
```

- **PRE:** `t` : tableau ~~non-null~~ de longueur `sz`
- **POST:** ~~`t` n'est pas modifié~~ et une ligne représentant le contenu de `t` est affichée sur stdout

Spécifications

Par défaut, on supposera que:

- **(PRE) tous les paramètres pointeurs sont non-nuls**
- **(POST) les paramètres ne sont pas modifiés**

```
void print (int* t, int sz);
```

- **PRE:** `t` : tableau de longueur `sz`
- **POST:** une ligne représentant le contenu de `t` est affichée sur stdout