

# Information, Calcul, Communication (partie programmation) : Fonctions (2)

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle  
Faculté I&C

# Rappel du calendrier

|             | MOOC                 | décalage / MOOC | exercices prog.<br>1h45<br>Jeudi 8-10 | cours prog.<br>45 min.<br>Jeudi 10-11 |
|-------------|----------------------|-----------------|---------------------------------------|---------------------------------------|
| 1 11.09.25  | --                   | -1              | prise en main                         | Bienvenue/Introduction                |
| 2 18.09.25  | 1. variables         | 0               | variables / expressions               | variables / expressions               |
| 3 25.09.25  | 2. if                | 0               | if – switch                           | if – switch                           |
| 4 02.10.25  | 3. for/while         | 0               | for / while                           | for / while                           |
| 5 09.10.25  | 4. fonctions         | 0               | fonctions (1)                         | fonctions (1)                         |
| 6 16.10.25  |                      | 1               | fonctions (2)                         | fonctions (2)                         |
| - 23.10.25  |                      |                 |                                       |                                       |
| 7 30.10.25  | 5. tableaux (vector) | 1               | vector                                | vector                                |
| 8 06.11.25  | 6. string + struct   | 1               | array / string                        | array / string                        |
| 9 13.11.25  |                      | 2               | structures                            | structures                            |
| 10 20.11.25 | 7. pointeurs         | 2               | pointeurs                             | pointeurs                             |
| 11 27.11.25 |                      | -               | entrées/sorties                       | entrées/sorties                       |
| 12 04.12.25 |                      | -               | erreurs / exceptions                  | erreurs / exceptions                  |
| 13 11.12.25 |                      | -               | révisions                             | théorie : sécurité                    |
| 14 18.12.25 | 8. étude de cas      | -               | révisions                             | Révisions                             |

# Objectifs du cours d'aujourd'hui

- ▶ Suite des rappels sur les fonctions en C++ :
  - ▶ (rappel de la méthodologie)
  - ▶ surcharge
  - ▶ fonctions récursives
- ▶ Etudes de cas

# Méthodologie pour construire une fonction

- ① clairement identifier ce que **doit faire** la fonction
  - 👉 ne pas se préoccuper ici du *comment*, mais bel et bien du **quoi** !  
(ce point n'est en fait que conceptuel, on n'écrit aucun code ici !)
- ② que doit recevoir la fonction pour faire cela ?
  - 👉 identifie les **arguments** de la fonction
- ③ pour chaque argument : doit-il être modifié par la fonction ?  
(si oui 👉 passage par référence)  
Optionnel : se demander si cela a un sens de donner une valeur par défaut au paramètre correspondant
- ④ que doit « retourner » la fonction    👉 type de retour  
Se poser ici la question (pour une fonction nommée *f*) :  
est-ce que cela a un sens d'écrire :  
$$z = f(\dots);$$
  
Si oui    👉 le type de *z* est le type de retour de *f*  
Si non    👉 le type de retour de *f* est *void*
- ⑤ (maintenant, et seulement maintenant) Se préoccuper du *comment* :  
c'est-à-dire comment faire ce que doit faire la fonction ?  
c'est-à-dire écrire le corps de la fonction

# La surcharge des fonctions

En C++, *les types des paramètres font partie intégrante de la définition d'une fonction.*

Il est de ce fait possible de définir **plusieurs fonctions de même nom** si ces fonctions *n'ont pas les mêmes listes de paramètres* : nombre ou types de paramètres différents.

Ce mécanisme, appelé **surcharge des fonctions**, est très utile pour écrire des fonctions « *sensibles* » au type de leurs paramètres, c'est-à-dire des fonctions correspondant à des traitements de même nature, mais s'appliquant à des entités de types différents.

# La surcharge des fonctions : exemple

```
void affiche(int x) {  
    cout << "entier : " << x << endl;  
}  
void affiche(double x) {  
    cout << "reel : " << x << endl;  
}  
void affiche(int x1, int x2) {  
    cout << "couple : " << x1 << x2 << endl;  
}
```

`affiche(1)`, `affiche(1.0)` et `affiche(1,1)` produisent alors des affichages différents.

## Remarque :

```
void affiche(int x);  
void affiche(int x1, int x2 = 1);
```

est interdit !

👉 ambiguïté

# Fonctions récurives

Rappel (ICC) :

Principe de l'approche récurive :

*ramener le problème à résoudre à un sous-problème,  
version simplifiée du problème d'origine.*



**Attention !** Pour que la résolution récurive soit **correcte**, il faut une **condition de terminaison**

sinon, on risque une boucle infinie.

# Exemple

Calculer la somme des  $n$  premiers entiers.

Si je sais le faire pour  $n$ , je sais le faire pour  $n + 1$  :

$$S(n + 1) = (n + 1) + S(n)$$

**Condition d'arrêt :**

Je sais le faire pour  $n = 0$  :  $S(0) = 0$

Algorithme :

|                                                                                          |
|------------------------------------------------------------------------------------------|
| <b>somme</b>                                                                             |
| entrée : $n$<br>sortie : $S(n)$                                                          |
| <b>Si</b> $n \leq 0$<br>  <b>Sortir</b> : 0<br><b>Sortir</b> : $n + \text{somme}(n - 1)$ |

# Code de l'exemple

```
// prototype
int somme(int n);

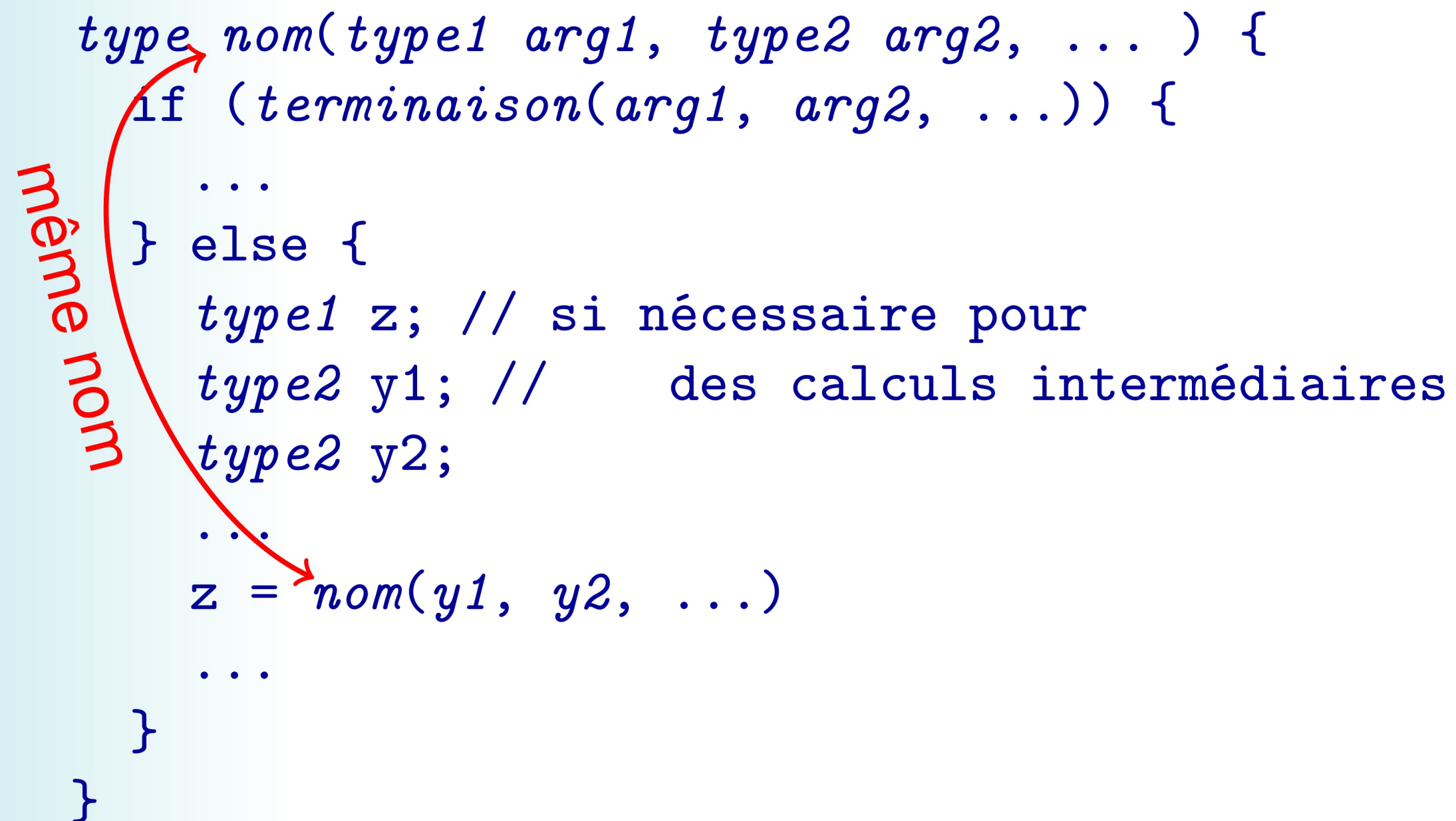
// définition
int somme(int n) {
    if (n <= 0) // condition d'arrêt
        return 0;
    return n + somme(n-1);
}
```

Sortie

j = somme(24);

# Les fonctions récurives

Le schéma général d'une fonction réursive est donc le suivant :



```
type nom(type1 arg1, type2 arg2, ... ) {  
  if (terminaison(arg1, arg2, ...)) {  
    ...  
  } else {  
    type1 z; // si nécessaire pour  
    type2 y1; //      des calculs intermédiaires  
    type2 y2;  
    ...  
    z = nom(y1, y2, ...)   
    ...  
  }  
}
```



# Les fonctions



Prototype (à mettre **avant** toute utilisation de la fonction) :

*type nom ( type1 arg1, ..., typeN argN [ = valN ] );*

*type* est *void* si la fonction ne retourne aucune valeur.

Définition :

```
type nom ( type1 arg1, ..., typeN argN )  
{  
    corps  
    return valeur;  
}
```

Passage par **valeur** :

*type f(type2 arg);*

*arg* ne peut pas être modifié par *f*

Passage par **référence** :

*type f(type2& arg);*

*arg* peut **être modifié** par *f*

Surcharge (exemple) :

*void affiche (int arg);*

*void affiche (double arg);*

*void affiche (int arg1, int arg2);*

# Etudes de cas

- conversion de décimal en binaire, version récursive

$f(5)$

$f$ : entrée :  $n \in \mathbb{N}$  (valeur décimale)  
Sortie : rien  
= en base 10

↳ afficher l'écriture binaire de  $n$

5

① Condition d'arrêt

Si  $n \leq 1$

afficher  $n$   
Sortir

$f(2)$

afficher  $n \% 2$   
Sortir

attendu:  
10

②

$f(5)$   
 $f(2)$

101

|     |    |
|-----|----|
| $n$ |    |
| 0   | 0  |
| 1   | 1  |
| 2   | 10 |

# Etudes de cas

- conversion de décimal en binaire, version récursive

- « augmentation » :

- si entier : **ajouter** le 2<sup>e</sup> argument, 1 par défaut
- si caractère : **passer en** majuscule si c'est une minuscule, sinon ne rien faire

augmente( **int** , int = 1 );

augmente( **char** );

? modifier ?

# Etudes de cas

- ▶ conversion de décimal en binaire, version récursive
- ▶ « augmentation » :
  - ▶ si entier : ajouter le 2<sup>e</sup> argument, 1 par défaut
  - ▶ si caractère : passer en majuscule si c'est une minuscule, sinon ne rien faire

```
augmente( int , int = 1 );  
augmente( char );
```

## Question :

```
void augmente(int&, int = 1);   et   void augmente(char&);
```

ou

```
int augmente(int, int = 1);     et   char augmente(char);
```

?

Passage par référence ou retour d'une nouvelle valeur ?

👉 question de point de vue !