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

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

Rappel du calendrier

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 9-11	cours prog. 45 min. Jeudi 11-12
1	23.09.21	--	-1	prise en main	Bienvenue/Introduction
2	30.09.21	1. variables	0	variables / expressions	variables / expressions
3	07.10.21	2. if	0	if – switch	if – switch
4	14.10.21	3. for/while	0	for / while	for / while
5	21.10.21	4. fonctions	0	fonctions (1)	fonctions (1)
6	28.10.21		1	fonctions (2)	fonctions (2)
7	04.11.21	5. tableaux (vector)	1	vector	vector
8	11.11.21	6. string + struct	1	array / string	array / string
9	18.11.21		2	structures	structures
10	25.11.21	7. pointeurs	2	pointeurs	pointeurs
11	02.12.21		-	entrées/sorties	entrées/sorties
12	09.12.21		-	erreurs / exceptions	erreurs / exceptions
13	16.12.21		-	révisions	théorie : sécurité
14	23.12.21	8. étude de cas	-	**Examen final (2h45)**	

(ne sont pas sur le MOOC)

©EPFL 2021
Jean-Cédric Chappelier & Jamila Sam

Objectifs du cours d'aujourd'hui

- Suite des rappels sur les fonctions en C++ :
  - *rappel méthodologie*
  - cas particuliers :
    - `Type f();`
    - `void f(Type);`
  - *surcharge*
  - valeurs par défaut
  - *fonctions récursives*
- Etudes de cas

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

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 à 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(....);
```

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

©EPFL 2021
Jean-Cédric Chappelier & Jamila Sam

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.

©EPFL 2021
Jean-Cédric Chappelier & Jamila Sam

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

©EPFL 2021
Jean-Cédric Chappelier & Jamila Sam

Fonctions récursives

Rappel (ICC) :
Principe de l'approche récursive :

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

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

sinon, on risque une boucle infinie.

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

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 :

somme
entrée : n sortie : $S(n)$
Si $n \leq 0$ **Sortir :** 0 **Sortir :** $n +$ **somme**$(n-1)$

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

Code de l'exemple

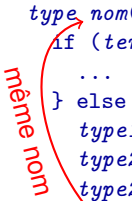
```
// prototype
int somme(int n);

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

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

Les fonctions récursives

Le schéma général d'une fonction récursive 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, ...)
    ...
  }
}
```

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam

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);
```

©EPFL 2021
Jean-Cédric Chappelier & Jamila Sam

Etudes de cas

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

©EPFL 2021
Jean-Cédric Chappelier
& Jamila Sam