

Développement Web (R112)

MMI, IUT1, UGA

Gwen Salaün

Algorithmique

(cours 4)

Plan

- Programmation
 - Algorithmique
 - Notion de variable
 - L'instruction d'affectation
 - Les instructions de lecture/écriture
 - Le choix
 - Les boucles
 - Factorisation du code
-
- Cours 1
- Cours 2
- Cours 3
- Cours 4

Factoriser le code

- Objectifs :
 - Ne pas réécrire plusieurs fois la même chose (factoriser)
 - Maintenir le code
- Les fonctions et procédures au S1
- Les objets et classes au S2 et S3

Un exemple

Un algorithme qui décrit le fonctionnement d'une machine à café

Algo cafe

var c : caractere

Début

Ecrire("Voulez vous un café (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant que

Si c = 'o' **Alors**

Ecrire ("Voulez vous de sucre (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

Lire(c)

Fin Tant que

Ecrire ("Voulez vous un café long (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant Que

Ecrire ("Veuillez patienter votre café se prépare")

Fin Si

Fin

Quel code peut être factorisé ?

Un exemple

Un algorithme qui décrit le fonctionnement d'une machine à café

Algo cafe

var c : caractere

Début

Ecrire("Voulez vous un café (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant que

Si c = 'o' **Alors**

Ecrire ("Voulez vous de sucre (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

Lire(c)

Fin Tant que

Ecrire ("Voulez vous un café long (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

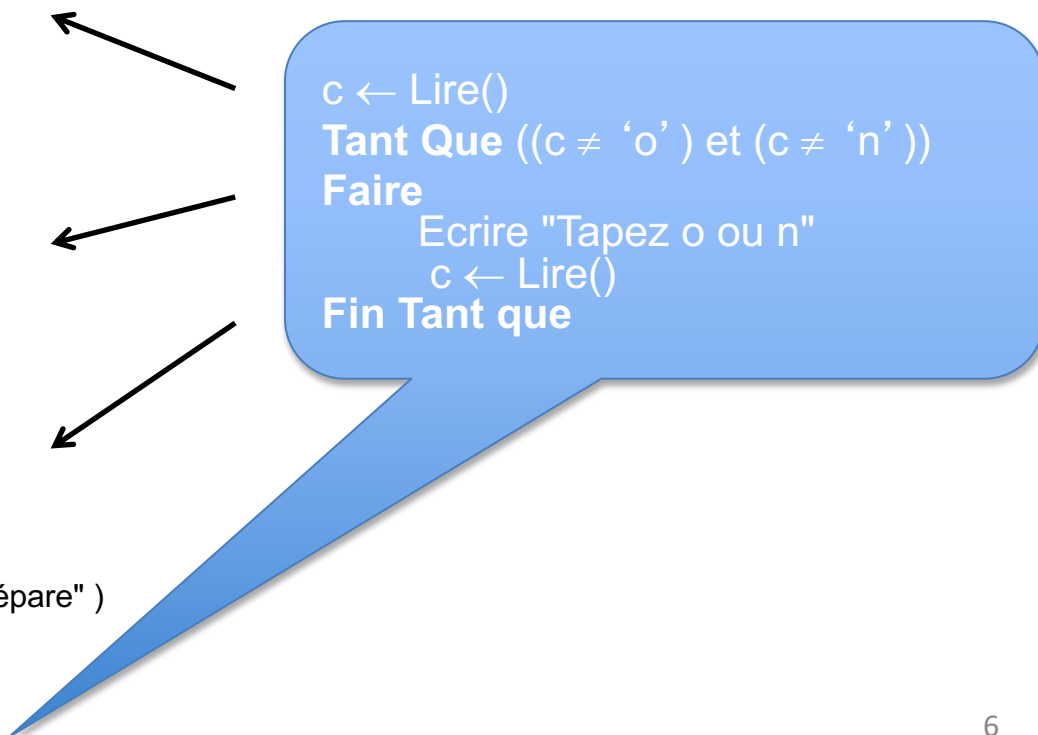
c ← Lire()

Fin Tant Que

Ecrire ("Veuillez patienter votre café se prépare")

Fin Si

Fin



```
c ← Lire()  
Tant Que ((c ≠ 'o') et (c ≠ 'n'))  
Faire  
    Ecrire "Tapez o ou n"  
    c ← Lire()  
Fin Tant que
```

Quel code peut être factorisé ?

Algo avec fonction

Algo cafe

var c : caractere

Début

Ecrire("Voulez vous un café (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant que

Si c = 'o' **Alors**

Ecrire ("Voulez vous du sucre (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

Lire(c)

Fin Tant que

Ecrire ("Voulez vous un café long (o/n)");

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant Que

Ecrire ("Veuillez patienter votre café se prépare")

Fin Si

Fin

Fonction oui-non() : caractere

var c: caractère

c ← Lire()

Tant Que ((c ≠ 'o') et (c ≠ 'n')) **Faire**

Ecrire "Tapez o ou n"

c ← Lire()

Fin Tant Que

Renvoie c

Fin Fonction

Algo cafe

var c : caractere

Début

Ecrire("Voulez vous un café (o/n)");

c ← oui-non()

Si c='o' **Alors**

Ecrire ("Voulez vous du sucre (o/n)");

c ← oui-non()

Ecrire ("Voulez vous un café long (o/n)");

c ← oui-non()

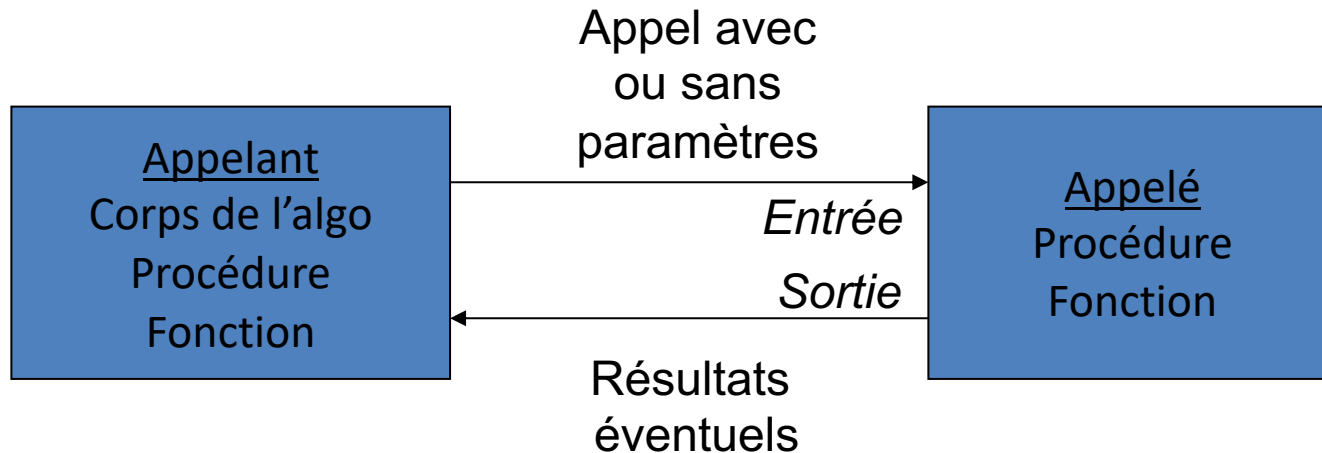
Ecrire ("Veuillez patienter votre café se prépare")

Fin Si

Fin



Fonctions et procédures



- Les procédures sont des sous-programmes qui ne renvoient rien
- Les fonctions sont des sous-programmes qui renvoient quelque chose

Pseudo-code pour les fonctions

Deux notions :

- La définition : le code du sous-programme
 - La signature (nom + paramètres),
 - Le corps (les instructions).

```
Fonction nomFonc (param1 : type1 [, param2 : type2] ) : typeRetour
    instructions
    Renvoie (resultat) // si nécessaire
Fin Fonction
```

- L'appel : l'utilisation du sous-programme
 - Notion de passage de paramètres

```
Algo Appel
Début
```

```
    nomFonc(val1,val2)           // sans valeur de retour
    X ← nomFonc(val1,val2)       // avec valeur de retour
```

```
Fin
```

Fonctions avec paramètres

Nous souhaitons améliorer notre fonction oui-non() pour qu'elle prenne le texte à afficher en paramètre

Fonctions avec paramètres

Nous souhaitons améliorer notre fonction oui-non() pour qu'elle prenne le texte à afficher en paramètre

```
Fonction oui-non (s : chaine) : caractere
    var c : caractère
    Ecrire(s)
    c ← Lire()
    Tant que ((c ≠ 'o' ) et (c ≠ 'n' )) Faire
        Ecrire "Tapez o ou n"
        c ← Lire()
    Fin Tant que
    Renvoie c
Fin Fonction
```

```
Algo cafe
    var c : caractere
Début
    c ← oui-non(« Voulez vous un café (o/n)" )
    Si c='o' Alors
        c ← oui-non("Voulez vous du sucre (o/n)" )
        c ← oui-non("Voulez vous un café long (o/n)" )
        Ecrire ("Veuillez patienter votre café se prépare" )
    Fin Si
Fin
```

Passage de paramètres

- le passage par valeur
 - Le paramètre est « copié » dans le sous-programme => les modifications du paramètre ne modifient pas la variable passée en paramètre mais sa copie.
- le passage par référence
 - La variable passée en paramètre n'est pas « copiée » mais utilisée directement dans le sous-programme => la variable passée en paramètre peut être modifiée.

Passage par valeur

Fonction plusplus (i : entier) : entier

$i \leftarrow i + 1$

renvoie(i)

Fin Fonction

Algo testplusplus

var j, k : entier

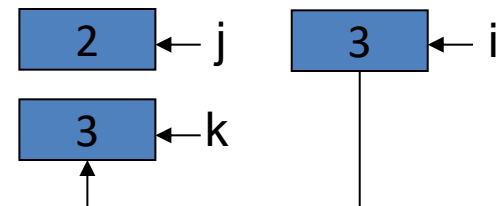
Debut

$j \leftarrow 2$

$k \leftarrow \text{plusplus}(j)$

Fin

Que valent 'j' et 'k' à la fin de l'algo ?



Passage par référence

Fonction plusplus (*i : entier*) : *entier*

i ← *i* + 1

renvoie(*i*)

Fin Fonction

Algo testplusplus

var *j, k* : *entier*

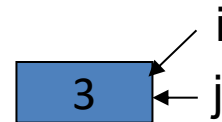
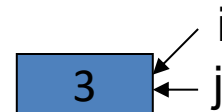
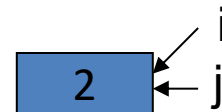
Debut

j ← 2

k ← plusplus(*j*)

Fin

Que valent 'j' et 'k' à la fin de l'algo ?



Passage de paramètres

Pendant les cours/TD d'algorithmique 'papier' nous supposerons que le passage de paramètres se fait par valeur.

Portée des variables

Fonction plusplus (*i : entier*) : *entier*

$i \leftarrow i + 1$

renvoie(*i*)

Fin Fonction

Algo testplusplus

var *i, k : entier*

Debut

$i \leftarrow 2$

$k \leftarrow \text{plusplus}(i)$

Fin

Les deux *i* sont totalement
indépendants

Fonction plusplus (*i : entier*) : *entier*

var *j : entier*

Debut

$j \leftarrow i + 1$

renvoie(*j*)

Fin Fonction

Algo testplusplus

var *j, k : entier*

Debut

$j \leftarrow 2$

$k \leftarrow \text{plusplus}(j)$

Fin

Les deux *j* sont totalement
indépendants

Récurtivité

Les fonctions peuvent s'appeler elles-mêmes pour répéter un traitement => appels récursifs

Exemple : calcul de la factorielle

$$\text{Fac}(0) = 1$$

$$\text{Fac}(1) = 1$$

$$\text{Fac}(n) = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

ou

$$\text{Fac}(n) = n * \text{Fac}(n-1) \quad // \text{ définition récursive}$$

Exemple : la factorielle

Version itérative

```
Fonction fac (i : entier) : entier  
    Var res : entier  
Debut  
    res ← 1  
    Tant que i > 1 Faire  
        res ← res * i  
        i ← i - 1  
    Fin Tant que  
    renvoie res  
Fin
```

Version récursive

```
Fonction fac (i : entier) : entier  
    Var res : entier  
Debut  
    Si (i=0) ou (i=1) Alors  
        res ← 1  
    Sinon  
        res ← i * fac (i-1)  
    Fin si  
    renvoie res  
Fin
```

Exécution de factorielle de 3

Version récursive

Fonction fac (*i* : entier) : entier

Var res : entier

Debut

Si (*i*=0) ou (*i*=1) **Alors**

res ← 1

Sinon

res ← *i* * fac (*i*-1)

Fin si

renvoie res

Fin

Fac(3) <= 3 *
fac(2) <= 2*
fac(1) <= 1

Fac(3) <= 3 *
fac(2) <= 2*
1

Fac(3) <= 3 *
fac(2) <= 2

Fac(3) <= 3 *
2

Fac(3) <= 6

Récurtivité

- Dans un algorithme récursif, vous aurez toujours une ou des **conditions d'arrêt** (**Si** (i=0) ou (i=1) **Alors** ...).
- Un **appel récursif** qui permet de converger vers les conditions d'arrêt (**res** \leftarrow i * **fac**(i-1)).

Fonction **fac** (i : *entier*) : *entier*

Var res: *entier*

Debut

Si (i=0) ou (i=1) **Alors**

 res \leftarrow 1

Sinon

 res \leftarrow i * **fac**(i-1)

Fin si

renvoie res

Fin

Exemple : somme d'entiers

Donner, en itératif puis en récursif, un algorithme permettant de calculer la somme des n premiers entiers.

Somme en itératif

Fonction somme ($n : \textit{entier}$) : *entier*

Var res, i : *entier*

Debut

res \leftarrow 0

pour i **de** 1 **à** n **pas de** 1

res \leftarrow res + i

renvoie res

Fin

Somme en récursif

Fonction somme ($n : \textit{entier}$) : *entier*

Var res : *entier*

Debut

Si ($n \leq 1$) **Alors**

 res \leftarrow 1

Sinon

 res \leftarrow n + somme (n-1)

Fin si

renvoie res

Fin

Conclusion

- Notions vues et à savoir : variables, entrée/sortie, choix, boucles, fonctions
- La suite pour ce semestre : comment coder nos algorithmes en PHP
- Le semestre prochain : les structures de données, la programmation objet, le développement web avancé

Au prochain cours

- Les bases de PHP