Fonctions et procédures

Hervé Locteau
Université de Nancy 2
locteau@loria.fr

L'intérêt des fonctions et des procédures

- Regrouper un ensemble d'instructions au sein d'un même algorithme afin de pouvoir y faire appel autant de fois que nécessaire.
- Intérêt : lisibilité (et identification des erreurs), réutilisation.
- On distingue la définition d'une fonction de son appel (son utilisation).
- Il est inutile de connaître comment est définie une fonction / une procédure pour pouvoir y faire appel. Par contre, il est nécessaire de connaître sa déclaration (son manuel d'utilisation) :
  - Quel est son rôle?
  - Quels sont les arguments à fournir?
  - Quel est la valeur retournée (signification, type)?

Fonctions versus Procédures

- On effectue parfois la distinction entre les fonctions et les procédures.
- À l'image des fonctions mathématiques, une valeur est retournée par les fonctions algorithmiques, à l'inverse des procédures.
- Exemples
  - ecriture(« bonjour ») constitue un appel à la procédure ecriture; on lui fournit en paramètre la valeur d'une chaîne de caractères; aucune valeur n'est retournée,
  - A ← lecture() constitue un appel à la fonction lecture; aucun argument est fourni (mais on écrit les parenthèses systématiquement; une valeur est retournée.
- On utilisera le terme « fonction » indifféremment.

Définition d'une fonction

- Une fonction peut avoir aucun, un ou plusieurs paramètres.
- Exemple

```
Fonction nf( Entree a en Entier, Entree b en Chaine )
    delivre Reel
Debut
    variables …, res en Reel
    …
    delivre res
Fin
```

(1) (2) (3)

  - (1) déclaration/prototype; (2) corps; (3) définition=déclaration+corps
  - Correspond à la définition d'une fonction identifiée par le nom nf,
  - Cette fonction admet deux paramètres; le premier doit être un entier tandis que le second est une chaîne de caractère,
  - Cette fonction retourne la valeur réelle d'une variable,
  - Le corps de la fonction constitue un algorithme dans lequel les variables a et b sont disponibles.

Fonction « utilisateur » – Exemple complet

```
0.  Programme SuiteArithmetique
1.      Fonction terme(Entree rang en Entier, Entree raison en Entier,
                       Entree terme0 en Entier ) delivre Entier
2.      Debut
3.          variable val en Entier
4.          val ← terme0 + raison * rang
5.          delivre val
6.      Fin
7.  Debut
8.      variable N, u0, uN, r en Entier
9.      ecriture( « Premier terme? » )
10.     u0 ← lecture()
11.     ecriture( « Raison? »)
12.     r ← lecture()
13.     ecriture(« termes u0 .. u10: »)
14.     Pour N dans 0..10 Faire
15.         uN ← terme(N,r,u0)
16.         ecriture( uN )
17.         ecriture(«  »)
18.     FinFaire
19.     ecriture(« \n »)
20. Fin
```

Remarque: les valeurs de N, r et u0 sont transmises, pas les variables!

Paramètres d'une fonction

- Dans sa déclaration, ils sont appelés paramètres formels,
- Au sein d'un appel, ils sont appelés paramètres effectifs,
- À chaque paramètre formel correspond un paramètre effectif, de même type,
- Le nom d'un paramètre effectif peut être différent de celui du paramètre formel associé.
- Exemple

  Soit la déclaration:
  Fonction Somme(Entree a en Entier, Entree b en Entier) delivre Entier

  Soit un algorithme pour lequel nous disposons des variables a, b, c, d:
  variables a, b, c, d en Entier

  Nous pouvons avoir les appels suivants:
  Somme(a,b) ou Somme(b,a) ou Somme(a,a) ou Somme(a,c+d), ...

Fonctions – Exercices

- Écrire la fonction poidsTotalAutorise qui accepte en entrée deux réels et retourne un booléen. Le premier paramètre concerne le poids actuel des passagers d'un ascenseur, le second, celui d'un nouveau passager. La valeur de retour de cette fonction doit être Vrai si le poids total avec ce nouveau passager est inférieur à 300 kg, Faux sinon.
- Écrire le programme faisant appel à cette fonction au sein d'une boucle où une lecture clavier permet de saisir le poids d'un passager. On sortira de la boucle avec l'affichage « portes fermées » en refusant l'accès au passager pour lequel le poids autorisé de la cabine serait dépassé.

Remarque: au sein du programme, nous sommes obligés de « gérer » le poids de la cabine...

Fonctions – Corrigés

1. **Fonction** poidsTotalAutorise(Entree poidsActuel en Reel, Entree poidsPassager en Reel) **delivre** booleen
2. **Debut**
3. **delivre** (poidsActuel + poidsPassager) < 300
4. **Fin**

0. **Programme** ascenseur
- - déclaration de la fonction poidsTotalAutorise
5. **Debut**
6. variable cage, passager en Reel
7. cage ← 0
8. passager ← 0
9. **TantQue** poidsTotalAutorise(cage, passager) **Faire**
10. cage ← cage + passager
11. passager ← lecture()
12. **FinFaire**
13. ecriture(« portes fermees »)
14. **Fin**

Une unique information délivrée par une fonction?

- Il est possible de définir également des paramètres de sorties en employant le mot clef correspondant **Sortie**.
- Exemple

0. **fonction** caractRectangleA(**Entree** L en Reel, **Entree** H en Reel, **Sortie** aire en Reel, **Sortie** perimetre en Reel)
1. **Debut**
2. aire ← L * H
3. perimetre ← (L+H)*2
4. **Fin**

0. **fonction** caractRectangleB(**Entree** L en Reel, **Entree** H en Reel, **Sortie** aire en Reel) **delivre** Reel
1. **Debut**
2. variable perimetre en Reel
3. aire ← L * H
4. perimetre ← (L+H)*2
5. **delivre** perimetre
6. **Fin**

- Un paramètre effectif de sortie doit être une variable!

Gestion de la cabine d'ascenseur, suite

```
1.  Fonction poidsTotalAutorise( Entree ancienPoids en Reel, Entree
poidsPassager en Reel, Sortie nouveauPoids en Reel) delivre booleen
2.  Debut
3.      nouveauPoids ← poidsActuel
4.      Si ( poidsActuel + poidsPassager ) < 300 Alors
5.          nouveauPoids ← nouveauPoids + poidsPassager
6.          delivre Vrai
7.      Sinon
8.          delivre Faux
9.      FinSi
10. Fin
```

```
0.  Programme ascenseur
11. Debut
12.     variable cage, passager, cageMaj en Reel
13.     cage ← 0
14.     passager ← 0
15.     TantQue poidsTotalAutorise( cage, passager,cageMaj ) Faire
16.         cage ← cageMaj
17.         passager ← lecture()
18.     FinFaire
19.     ecriture( « portes fermees » )
20. Fin
```

Des entrées, des sorties, … des entrées-sorties?

- Un paramètre peut jouer à la fois le rôle de paramètre d'entrée et de sortie. Le paramètre effectif correspondant est alors nécessairement une variable initialisée, pour que son rôle de sortie puisse être accompli.

- Exemple

```
Fonction doubler( Entree-Sortie x en Reel)
Debut
    x ← x * 2
Fin
```

- Exercice

  Écrire un programme permettant de saisir trois réels (variables a, b, c) et, suite à plusieurs appels à une fonction permuter, permettant d'échanger les valeurs de deux variables, conduit à avoir a <= b <= c, quelles que soient les valeurs saisies.

Gestion de la cabine d'ascenseur, fin

```
1.  Fonction poidsTotalAutorise( Entree-Sortie PoidsCabine en Reel, Entree
    poidsPassager en Reel) delivre booleen
2.  Debut
3.      Si ( PoidsCabine + poidsPassager ) < 300 Alors
4.          PoidsCabine ← PoidsCabine + poidsPassager
5.          delivre Vrai
6.      Sinon
7.          delivre Faux
8.      FinSi
9.  Fin
```

Il faut mémoriser ces valeurs ou les renvoyer immédiatement. Après le « FinSi », nous ne savons plus si PoidsCabine a été mis à jour, et « delivre PoidsCabine < 300 » y est toujours Vraie

```
0.  Programme ascenseur
10. Debut
11.     variable cage, passager en Reel
12.     cage ← 0
13.     passager ← 0
14.     TantQue poidsTotalAutorise( cage, passager ) Faire
15.         passager ← lecture()
16.     FinFaire
17.     ecriture( « portes fermees » )
18. Fin
```

Trier les valeurs de trois variables

```
0.  Programme triVariables
1.  Fonction Permuter( Entree-Sortie x en Reel, Entree-Sortie y en Reel )
2.  Debut
3.      variable z en Reel   4.   z ← y   5.   y ← x   6.   x ← z
7.  Fin
8.  Debut
9.      variable a, b, c en Reel
10.     a ← lecture()
11.     b ← lecture()
12.     c ← lecture()
- - trouver la bonne valeur de a
14.     Si a > b Alors
15.         Permuter( a,b )
16.     FinSi
17.     Si a > c Alors
18.         Permuter( a,c )
19.     FinSi
- - trouver la bonne valeur de b
- - et donc également de c
20.     Si b > c Alors
21.         Permuter( b,c )
22.     FinSi
23. Fin
```

Remarque: les instructions « Permuter(a,b) » et « Permuter(b,a) » sont strictement équivalentes.

- Vérifier le bon déroulement du programme en complétant le tableau ci-contre. Pour chaque cellule, donnez les valeurs de (a,b,c).

saisie	instructions						
	14	15	17	18	20	21	23
(1,2,3)							
(1,3,2)							
(2,1,3)							
(2,3,1)							
(3,1,2)							
(3,2,1)							

Raisonnement par récurrence

Raisonnement par récurrence

Soit une propriété P sur $\mathbb{N}$ telle que P(0) est vraie et l'implication $P(n) \Rightarrow P(n+1)$. Alors P(n) est vraie pour tout n de $\mathbb{N}$.

Suite récurrente

Soit un ensemble E, on note (a_n) une suite d'éléments de E telle que:

$$\begin{cases} a_0 \text{ est connue} \\ a_n = f(a_{n-1}), \text{ pour } n>0 \text{ et } f:E \rightarrow E. \end{cases}$$

La suite (a_n) avec $n \in \mathbb{N}$ est une suite récurrente. On dit qu'elle est d'ordre 1 si la fonction f ne fait intervenir que a_{n-1}. Elle est entièrement déterminée pour tout n de $\mathbb{N}$.

Fonction récursive – Racine carrée (1)

- La valeur de la racine carrée d'un nombre A peut être approximée en employant la suite récurrente d'ordre 1 suivante:

$$\begin{cases} X_0 = 1 \\ X_n = \frac{1}{2} \left(X_{n-1} + A / X_{n-1} \right) \end{cases}$$

  Une valeur convenable est atteinte lorsque $|X_m^2 - A| < \varepsilon$, où ε est une valeur très faible.

- Racine de 4
  on obtient $x_0=1$, $x_1=(1+4/1)/2=2.5$, $x_2=(2.5+4/2.5)/2=2.05$, $x_3=(2.05+4/2.05)/2 \approx 2.000609756$

- Racine de 9
  on obtient $x_0=1$, $x_1=(1+9/1)/2=5$, $x_2=(5+9/5)/2=3.4$, $x_3=(3.4+9/3.4)/2 \approx 3.023$, $x_4 \approx (3.023+9/3.023)/2 \approx 3.00008745$

Fonction récursive – Racine carrée (2)

- Version itérative du calcul de la racine carrée

```
0. Fonction racine_it( Entree A en reel ) delivre reel
1. Debut
2.     variable X en reel
3.     X ← 1
4.     TantQue Abs(X*X-A) >= 0.001 Faire
5.         X ← ( X + A / X ) / 2
6.     FinFaire
7.     delivre X
8. Fin
```

Fonction récursive – Racine carrée (3)

- Version récursive du calcul de la racine carrée

```
0.  Fonction racine_rec_est( Entree A en reel, Entree x ) delivre reel
1.  Debut
2.      variable res en Reel
3.      Si Abs( x*x-A) < 0.001 Alors
4.          res ← x
5.      Sinon
6.          res ← racine_rec_est( A, (x+A/x)/2 )
7.      FinSi
8.      delivre res
9.  Fin
10. Fonction racine_rec( Entree A en reel ) delivre reel
11. Debut
12.     delivre racine_rec_est( A, 1 )
13. Fin
```

Fonction récursive – Factorielle (1)

- On peut écrire la factorielle de X par une suite:

$$\begin{cases} f_0 = 1 \\ f_n = n * f_{n-1} \end{cases}$$

On a X!=f_X. Sous cette forme, f_n dépend de f_{n-1} et de n et la suite (f_n) n'est pas récurrente d'ordre 1 puisque n intervient dans le calcul...

On peut toutefois définir deux suites (n_i) et (F_i) avec

$$\begin{cases} n_0 = 0 \\ n_i = 1 + n_{i-1} \end{cases} \qquad \begin{cases} F_0 = 1 \\ F_i = n_i * F_{i-1} \end{cases}$$

ou encore $(n,F)_0=(0,1)$ et $(n,F)_i=(1+n_{i-1},n_i*F_{i-1})$ qui est une suite récurrente d'ordre 1.

- Rappel : la factorielle de n, notée n!, vaut : 1*2*...*(n-1)*n

Fonction récursive – Factorielle (2)

- Écrire l'algorithme de la fonction factorielle en employant la récursivité.

Fonction récursive – Factorielle (3)

```
0. Fonction factorielle_rec( Entree n en Entier ) delivre Entier
1. Debut
2.     variable fn en Entier
3.     Si n = 0 OU n == 1 Alors
4.         fn ← 1
5.     Sinon
6.         fn ← n * factorielle_rec( n-1 )
7.     FinSi
8.     delivre fn
9. Fin
```

Fonction récursive – Suite de Fibonacci

- Soit la suite définie par (suite récurrente d'ordre 2):

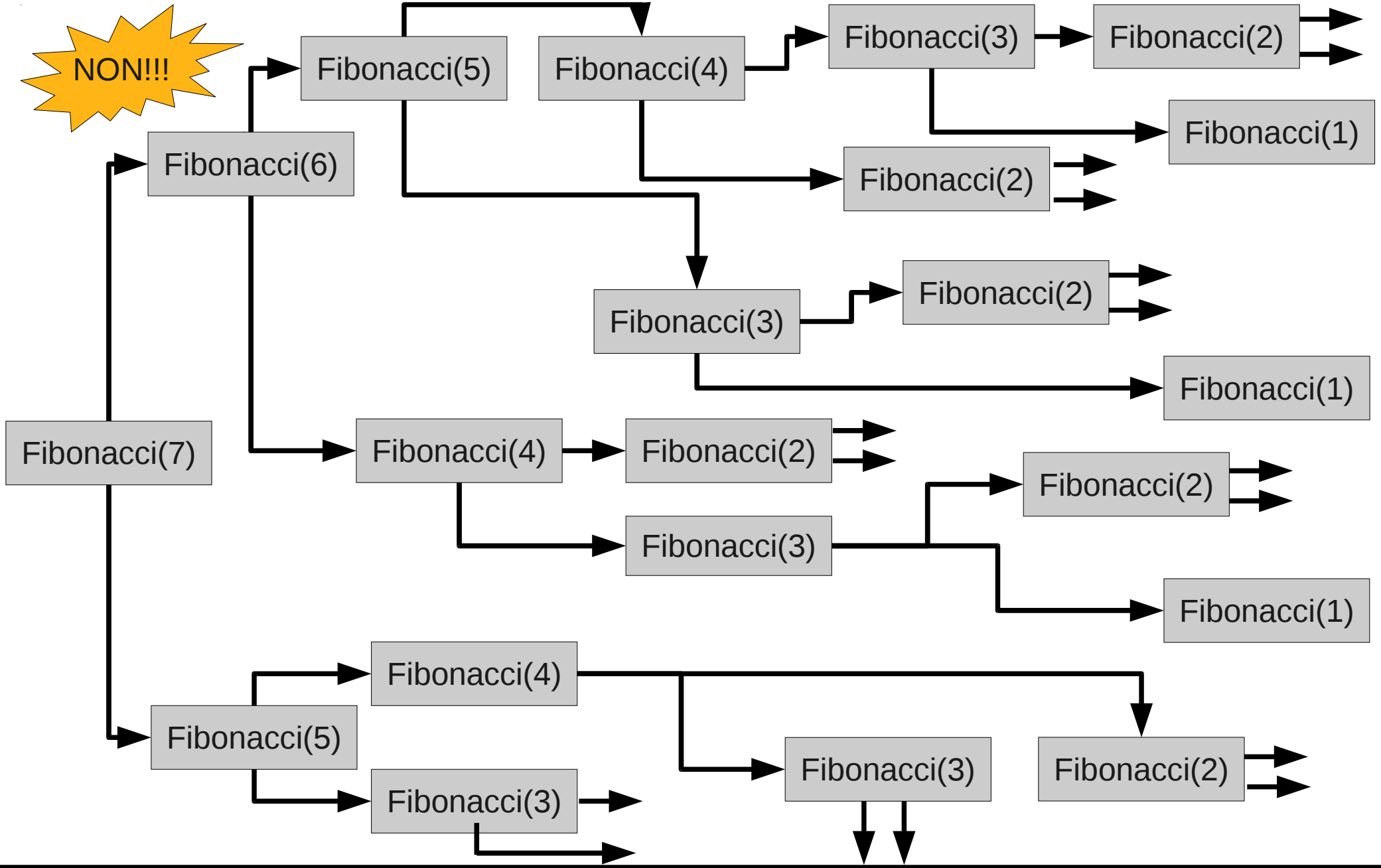
$$\begin{cases} u_0 = u_1 = 1 \\ u_n = u_{n-1} + u_{n-2} \end{cases}$$

  On a les valeurs 1,1,2,3,5,8,13,21,34, …

- Algorithme naïf

```
0. Fonction fibonacci( Entree n en Entier ) delivre Entier
1. Debut
2.     variable res en Entier
3.     Si n = 0 ou n = 1 Alors
4.         res ← 1
5.     Sinon
6.         res ← fibonacci(n-1)+fibonacci(n-2)
7.     FinSi
8.     delivre res
9. Fin
```

Suite de Fibonacci – Solution naïve



Suite de Fibonacci – Solution linéaire

- Principe : transmettre deux valeurs consécutives.
- Algorithme

```
0.  Fonction Fibo_rec( Entree rang en Entier, Entree moins2 en Entier, Entree moins1 en Entier ) delivre Entier
1.  Debut
2.      variable res en Entier
3.      Si rang = 0 ou rang = 1 Alors
4.          res ← moins1
5.      Sinon
6.          res ← Fibo_rec( rang-1, moins1, moins1+moins2 )
7.      FinSi
8.  Fin
9.  Fonction Fibo( Entree rang en Entier ) delivre Entier
10. Debut
11.     delivre Fibo_rec( rang, 1, 1 )
12. Fin
```

- Écrire quels sont les appels successifs à Fibo_rec pour Fibo(7)

Suite de Fibonacci – Déroulement

- Fibo(7)
  - → Fibo_rec(7,1, 1)
    - → Fibo_rec(6, 1, 1+1)
      - → Fibo_rec(5, 2, 1+2)
        - → Fibo_rec(4, 3, 2+3)
          - → Fibo_rec(3, 5, 3+5)
            - → Fibo_rec(2, 8, 5+8)
              - → Fibo_rec(1, 13, 8+13)