Algorithmique et Programmation, IMA

Cours 3 : Actions, Procédures

Laure Gonnord
http://laure.gonnord.org/pro/teaching/
Laure.Gonnord@polytech-lille.fr

Université Lille 1 - Polytech Lille

Conception Structurée descendante - 1

Découper l'algorithme (action) en sous-algorithmes (sous-actions) plus simples, jusqu'à des opérations considérées primitives. Buts :

- Simplification
- Abstraction (ignorer les détails)
- Structuration
- Réutilisation

Conception Structurée descendante - 2

Exemple : sélectionner les entiers selon un certain critère

- une fonction de sélection qui dit "oui" ou "non" et qui peut être plus ou moins compliquée ;
- un appel dans le "main".

Outil : actions et fonctions paramétrées.

Fonctions - Définition

Une fonction est un sous-programme qui à partir de **données** produit un (et un SEUL) **résultat**.

Syntaxe Algo (exemple)

Fonction *max(a,b) : entier*
| **D**: a,b : entiers {Données}
| **L**: m : entier {Variable locale}
| **Si** *a<b* **alors**
| | m ← b
| **Sinon**
| | m ← a
| **Fsi**
| **Retourner** *m* {Obligatoire}
FFonction

Fonctions - Appel de fonction

Un **appel** de fonction est une expression du **type de retour** de la fonction.

Exemple :
x ← max(3,43)

- Les données sont remplacées par des valeurs (ou des expressions)
- Le résultat est récupéré dans une variable (par exemple)

Fonctions en C - Syntaxe

Déclaration

```
def_fun ::=   type identificateur (liste-params)
                {
                  liste-declarations (option)
                  liste_instructions
                }
```

La liste d'instructions comprend **au moins** une instruction `return`

Appel (poly p 189)

```
expression ::= ...
      | identificateur (liste-expressions )
      | expression_unaire op expression_affectation

expression_unaire ::= id | cte | ..
```


Fonctions en C - Exemple 1

Définition d'une fonction

```
int max (a:int,b:int)
{
  int m;
  if a<b then m=a else m=b;

  return(m);
}
```

attention au type de retour !

Appel

```
toto = max(3,45); //int declare avant !
```


Fonctions en C - deuxième exemple

Bons entiers

- Modifier le programme de sélection des entiers inférieurs à 100 pour utiliser la fonction d'entête :

  ```
  bool bon_entier(int n)
  ```

- Écrire la fonction `bon_entier` de façon à sélectionner les entiers multiples de 3.

Les actions - définition

Une action ne retourne pas de résultat.

Action *maxproc(a,b,maxi)*
- **D**: a,b : entiers {Données}
- **R**: maxi : entier {Résultat}
- **Si** *a<b* **alors**
  - maxi ← b
- **Sinon**
  - maxi ← a
- **Fsi**

FAction

Les variables résultats servent à propager les informations produites à l'extérieur de la définition.

Les actions - Utilisation (1)

L'**appel** d'une action est une **instruction**. Les paramètres formels sont TOUS remplacés par des paramètres effectifs **de même type**

- Une donnée par une valeur (ou une expression qui a une valeur)
- Un résultat par une variable dans laquelle la procédure doit ranger le résultat
- Une Donnée/Résultat par une variable valuée.

Les actions - Utilisation (2)

Exemple
resu : entier ;
maxproc(3,43,resu);

▶ Après l'appel, la variable `resu` contient le max des deux entiers.

Utilisation : fonctions d'impression, ou modification des paramètres, qui sont alors appelés *Données/Résultats* ou *entrées/sorties*.

Les actions en C - procédures

En C les actions/procédures sont des fonctions qui ne retournent rien (mot clef **void**).

```
void maxproc(int a, int b)
{ // impression du max
  int maxi;
  if a<b then  maxi=b else maxi=a;
  printf("Le max est %d \n",maxi);
}
```

Paramètres R ? voir le chapitre « pointeurs »

Procédures en C - Syntaxe

Déclaration

```
def_proc ::=   void identificateur (liste-params)
                 {
                   liste-declarations (option)
                   liste_instructions (option)
                 }
```

Appel (poly p 189)

```
expression ::= ...
      | identificateur (liste-expressions )
```


Procédures en C - Exemples

Écrire les procédures suivantes :

- Impression du maximum de 3 entiers en paramètres
- Impression des 100 premiers termes de la suite suivante :
  - $u_0 = 32$
  - $u_n = 3 * u_{n-1} + 19$

Procédures en C - Remarques

Les procédures sont surtout utiles pour :

- **imprimer** des valeurs, des structures, des messages . . .
- **modifier** des paramètres qui ne peuvent être retournés (tableaux, paires, structures compliquées) (voir pointeurs)

Définition

Un algorithme (une fonction, une procédure) est dit **récursif** si sa définition (son code) contient un appel à lui-même.

Un algorithme qui n'est pas récursif est dit **itératif**.

Utilisations usuelles

Utilisations variées (liste non exhaustive) :

- Calcul de suite **récursive** (numérique, graphique. . .)
- Calcul de type « diviser pour régner » : recherche, tri, . . .
- Calcul sur des structures de données **inductives** (listes, arbres, . . .) ▶ cours de SD.

▶ Dans tous les cas, une version itérative est possible.

Quelques exemples classiques - 1

Factorielle : $n! = n.(n-1)!$

Fonction *fact(n) : entier*
| **D**: n : entier positif ou nul
| **Si** *n=0* **alors**
| | **Retourner** *1*
| **Sinon**
| | **Retourner** *n*fact(n-1)* {Appel récursif}
| **Fsi**
FFonction

▶ Attention au type de retour et à l'orthographe du nom de la fonction.
▶ Dérouler !

Quelques exemples classiques - 2

Fibonacci : $Fibo(n) = Fibo(n-1) + Fibo(n-2)$

Fonction *fibo(n) : entier*
| **D**: n : entier positif ou nul
| **Si** *n=0* **alors**
| | **Retourner** *0*
| **Sinon**
| | **Si** *n=1* **alors**
| | | **Retourner** *1*
| | **Sinon**
| | | **Retourner** *fibo(n-1)+fibo(n-2)* {Appel récursif}
| | **Fsi**
| **Fsi**
FFonction

▶ Dérouler !
▶ L'implémentation impérative est meilleure, pourquoi ?

Quelques exemples classiques - 3

Que fait cette fonction ?

Fonction *somme (n,r) :entier*
- **D**: n,r : entier positifs ou nul
- **Si** *n = 1* **alors**
  - **Retourner** *r + 1*
- **Sinon**
  - **Retourner** *somme (n -1 , r + n)*
- **Fsi**

FFonction

▶ r est appelé paramètre d'**accumulation**.

Récursivité terminale (ou pas ?)

Un algorithme récursif est dit récursif **terminal** si l'appel récursif est la dernière instruction réalisée.

▶ Stockage non nécessaire de la valeur obtenue par récursivité.

- Factorielle : fact(n-1) puis multiplication par n, donc non récursif terminal.
- Somme : récursif terminal :
  $somme(5,0) = somme(4,5) = somme(...) \ldots = 15$

Dérécursivons !

Pour l'algorithme somme, la forme « accumulateur » et **l'invariant** $resu = \sum_{j=r}^{i} j$, fournit rapidement un algorithme itératif :

Fonction *somme2 (n)*
- **D**: n : entier positif ou nul
- r ← 0 {accumulateur}
- i ← n
- **Tq** *i⩾ 2* **faire**
  - r← r+i
  - i← i-1
- **Ftq**
- **Retourner** *r*

FFonction

▶ Autres exemples en TD

Et l'inverse ?

La procédure : **Action** *compter()*
- **L**: i :entier
- **Pour** i **de** *0* **à** *10* **Faire**
  - traitement(i)
- **Fpour**

FAction

se transforme en procédure récursive en :

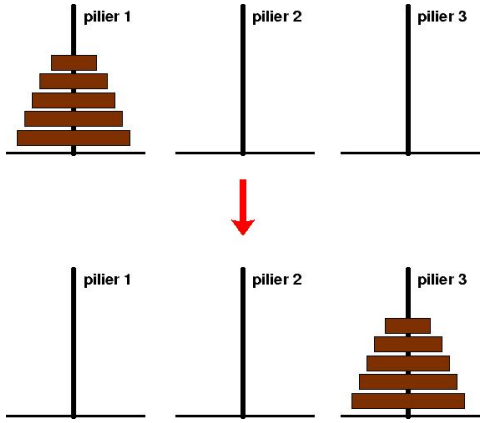
Action *compter(i)*
- **D**: i :entier
- **Si** $i \leqslant 11$ **alors**
  - traitement(i)
  - compter(i+1))
- **Fsi**

FAction

Attention

Toujours bien vérifier que votre algorithme termine !

Un exemple plus exotique ! Tours de Hanoi - 1



- déplacement d'une seule rondelle à la fois
- rondelle jamais au dessus d'une plus grosse.

Un exemple plus exotique ! Tours de Hanoi - 2

- Si 0 rondelle, je sais faire !
- Si je sais faire avec $n-1$ rondelles, comment faire pour traiter le cas n rondelles ?

- Laissé en exercice !