Algorithmique et Programmation, IMA

Cours 3 : Actions, Procédures

Université Lille 1 - Polytech Lille

Conception Structurée Descendante

Les Fonctions

Les Actions / les Procédures

Résumé

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 $\leftarrow$ **max(3,43)**
Que se passe-t-il lors de l'appel ?

- Les données sont remplacées par des valeurs (ou des expressions)
- Le code de la fonction est exécuté jusqu'au premier return.
- Le résultat **retourné** par la fonction est la valeur de l'expression du return.
- Ce résultat (valeur) est récupéré dans la variable x ici.

Fonctions en C - Syntaxe

Définition

```
type_de_retour nom_fonction(liste-params) {
    liste-declarations (optionnelle)
    liste_instructions
}
```

La liste d'instructions comprend **au moins** une instruction `return` (du type `type_de_retour`).

Appel

```
nom_fonction(liste-expressions)
```

Fonctions en C - Exemple

Définition d'une fonction

```
int max (int a, int b)
{
   int m;
   if (a>b) m=a; else m=b;

   return(m);
}
```

attention au type de retour !

Appel

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

Fonctions en C - Exercices

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.

Fonctions en C - Erreurs courantes

- Oubli du ; dans le if :

```
toto.c:9:17: error: expected ';' after expression
  if ( a>b ) m=a  else m=b ;
                ^
```

- Oubli du return :

```
toto.c:11:1: warning: control reaches end of non-void
 function [-Wreturn-type]
}
^
```

- appel avec des arguments du mauvais type : par exemple `max("tsoin",4)` :

```
toto.c:42:18: warning: incompatible pointer to integer conversion passing
      'char [6]' to parameter of type 'int' [-Wint-conversion]
  int toto = max("tsoin",4);
                 ^~~~~~~
toto.c:6:15: note: passing argument to parameter 'a' here
int max ( int a , int b )
```

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 $\leftarrow$ b
Sinon
maxi $\leftarrow$ 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.

Les actions/procédures sont surtout utiles pour :

- **imprimer** des valeurs, des structures, des messages . . .
- **modifier** des paramètres qui ne peuvent être retournés (tableaux, paires, structures compliquées) : ceux-ci 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 printmaxproc(int a, int b)
{ // impression du max
   int maxi;
   if(a<b) maxi=b; else maxi=a;
   printf("Le max est %d \n",maxi);
}
```

Paramètres R ? voir le chapitre « pointeurs »

Procédures en C - Syntaxe

Définition

```
void nom_action(liste-params) {
    liste-declarations (optionnelle)
    liste_instructions
}
```

Appel

```
nom_action(liste-expressions)
```

- on ne récupère pas le résultat d'une procédure, il n'y en a PAS.

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 :
  - $u_0 = 32$
  - $u_n = 3 * u_{n-1} + 19$
- Impression des k premiers termes de la suite, avec k passé en paramètre.

Les fonctions

Une fonction retourne un **et un seul** résultat.

Fonction *ajoute_un(a) : Entier*
| **D**: a : Entier
| **Retourner** *a+1*
FFonction

Appel :

Programme *Main*
| **L**: x,y :Entiers
| x ← 12
| y ← ajoute_un(x)
| Imprimer(y)
| **Retourner** *0*
FProgramme

Les actions

Une action ne retourne pas de résultat.

Action *imprime_succ(a)*
| **D**: a : Entier
| Imprime(a+1)
FAction

Appel :

Programme *Main*
| **L**: x :Entier
| x ← 12
| imprime_succ(x)
| **Retourner** *0*
FProgramme

Et si ?

- Et si je veux modifier un paramètre d'entrée ?

Action *inc(a)*
| **D/R**: a : Entier
| a ← a+1
FAction

Programme *Main*
| **L**: x :Entier
| x ← 12
| inc(x)
| Imprime(x)
| **Retourner** *0*
FProgramme

Et si ?

- Et si je veux “retourner” deux résultats ?

Action *quotient_et_reste(a,b,q,r)*
| **D**: a,b : Entiers
| **R**: q,r : Entiers
| .....
| (calcul de q et r)
FAction

Programme *Main*
| **L**: x,y,vq,vr :Entiers
| x $\leftarrow$ 37
| y $\leftarrow$ 7
| quotient_et_reste(x,y,vq,vr)
| Imprime(vq,vr)
| **Retourner** *0*
FProgramme

Algorithmique et Programmation, IMA

Cours 3c - Récursivité

Université Lille 1 - Polytech Lille

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, . . .) ► Prog avancéee (S6).

► 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 ! - Complexité en **nb d'appels** ?

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 1
    Sinon
        Si n=1 alors
            Retourner 1
        Sinon
            Retourner fibo(n-1)+fibo(n-2)      {Appel récursif}
        Fsi
    Fsi
FFonction
```

- Dérouler ! ▶ Complexité en **nb d'appels** ?
- L'implémentation impérative est meilleure, pourquoi ?

Quelques exemples classiques - 3

Que calcule somme(5,0) ?

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 !

La factorielle :

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

```
Fonction fact(n) : entier
|   D: n : entier positif ou nul
|   L: i,f :entier
|   f← 1
|   Pour i de 0 à n Faire
|   |   f← f*i
|   Fpour
|   Retourner f
FFonction
```

Attention

Toujours bien vérifier que votre algorithme termine !