Récursivité-Précision-Suites-Graphique

25 janvier 2013

Deux types de fichiers de programmes

Les fichiers exécutables

Un fichier exécutable est un fichier texte suffixé par `.sce`.

On appelle le fichier `machin.sce` en tapant :

```
exec('machin.sce')
```

Remarque : 'machin.sce' est le nom complet du fichier !
Pour moi, il s'agira par exemple de :

'/home/schaub/scilab/machin.sce'

Exécuter un fichier `.sce` est équivalent à entrer successivement toutes les commandes contenues dans le fichier. La commande *Pause* permet d'arrêter le déroulement et de reprendre à volonté. Il existe aussi différents modes (voir cette fonction).

Des bibliothèques de fonctions

C'est un fichier texte, suffixé avec un suffixe `.sci`.

Il contient des définitions de fonctions.
Chaque fonction est déclarée en suivant une syntaxe :

Syntaxe de fonction

```
function y = f(x)
    y=sin(x)+sin(2*x)
(il peut avoir autant de lignes qu'on veut)
endfunction
```

On charge aussi `truc.sci` par la même commande `exec('truc.sci')` (ce qui représente une nouveauté de la version 5.2 de Scilab).

Une fois la fonction chargée, la fonction `f` est disponible.
On peut voir un exemple avec le flocon de von Koch.

La boucle **for**

Calcul de la somme des entiers impairs de 1 à 99

```
s=0
for k=1 :2 :100
    s=s+k
end
```

Dans cet exemple, l'indice k parcourt toutes les valeurs des coefficients du vecteur `1:2:100`.

On notera que si `M` est une matrice, une commande commençant par `for i=M`, va parcourir tous les coefficients de `M`, et les affecter successivement à `i`.

La boucle **while**

Calcul de la somme des entiers impairs de 1 à 99

Même exemple que précédemment avec une boucle `while`

```
s=0,k=1
while k < 100
    s=s+k
    k=k+2
end
```

On peut inclure ces "boucles" dans des *bibliothèques de fonction*. Voyons des exemples avec les fichiers *cours2.sci* et *cours2.sce*.

L'opérateur conditionnel **if**

Test de positivité

```
if x=>0 then
    y=sqrt(x)
end
```

Les principaux opérateurs de test sont <, <=, >, >=, ==, <>. On peut les relier par des opérateurs logiques | (ou) et & (et). La syntaxe complète est la suivante

```
if condition then
    liste d'instructions
elseif autre-condition then
    autre liste d'instructions
else
    encore d'autres instructions
end
```

La structure conditionnelle **select**

Autre type de structure conditionnelle : **select/case**.

La syntaxe est la suivante :

```
select expr
 case expr1 then  instructions1
 case expr2 then  instructions2
 ...
 case exprn then  instructionsn
 else  autres instructions
end
```

Exemple à la fin de *cours2.sce*.

Le else est facultatif, et il faut veiller à ce que le mot clé then se trouve sur la même ligne que le case correspondant.

Cette commande effectue l'opération *i* si l'expression expri coincide avec expr.

Règles d'indentation

Pour assurer la lisibilité des programmes, respectez les règles suivantes :

- La déclaration d'une fonction function et le endfunction qui la termine sont écrites aussi à gauche que possible, et tout le contenu de la fonction est décalé d'un cran vers la droite.
- On écrit un opérateur de contrôle for ou while et le end qui le termine sur un même niveau, et toutes les actions situées entre l'opérateur et le end sont décalées d'un cran vers la droite.
- De même un if se trouve au même niveau que les else, elseif et end qui en dépendent, et toutes les actions conditionnées par cet if sont décalées vers la droite

Note : l'éditeur intégré comme Emacs (éditeur sous Unix le plus utilisé) en mode Scilab réalisent par eux-mêmes les indentations nécessaires (à condition de finir chaque ligne par "Entrée").

Variables locales et globales

Soit la fonction :

```
function [y1,y2,y3]=mafonction(x1,x2)
  y1=x1+1
  z=x1+2
  y2=z*y1
  y3=x2+1
endfunction
```

x1 et x2 sont des variables extérieures à la fonction.
Si on invoque la fonction : [a,b,c]=mafonction(1,2), on récupère dans [a,b,c] la valeur [2,6,3].
Pendant l'exécution de la fonction, *Scilab* utilise une variable z, qui prend à un moment la valeur 3 ; une fois que la fonction aura fini son travail, il ne restera rien de cette affectation provisoire.
Les variables x_i (et les y_j !) sont des variables globales ; à l'inverse, la variable z est une variable locale.

Programmation et débogage

Pour programmer en scilab, il faut ouvrir une fenêtre d'éditeur de texte (l'éditeur intégré de Scilab ou, pour ouvrir une fenêtre indépendante, un autre éditeur : emacs sous linux, par exemple) et une fenêtre scilab.

A chaque fois qu'on a écrit ou modifié une fonction, on enregistre le fichier, on le fait charger par Scilab et on le teste. Ensuite, on corrige les erreurs, et on recommence ...

En plus des messages d'erreurs, Scilab offre des fonctions de débogage.

Pour chercher une erreur, on peut insérer des instructions pause dans le fichier à exécuter.

Lors de l'exécution, si *Scilab* rencontre une pause, le programme s'interrompt et rend la main à l'utilisateur. Celui-ci peut alors regarder les valeurs des différentes variables locales à cet instant.

On reprend l'exécution par l'instruction resume (ou de manière équivalente return). Le programme reprend alors, sans tenir compte de tout ce qu'a fait l'utilisateur pendant l'interruption.

Voir aussi l'aide concernant les fonctions abort et break.