TD Algorithmique n°2

PTSI B Lycée Eiffel

2 octobre 2012

Principes de base de programmation

Nous avons déjà vu lors du premier TD ce qu'était un algorithme (suite d'instructions mécaniques permettant de résoudre un problème mathématique donné), et constaté sur un ou deux exemples que l'écriture d'algorithme nécessiterait de pouvoir donner à la machine des instructions faisant par exemple intervenir des disjonctions de cas (si une condition est vérifiée, alors on effectue un calcul ; sinon on en effectue un autre), ce qui suppose que nous apprenions la syntaxe correspondant à ce genre d'instructions dans le langage de programmation de Maple.

Un autre concept important sur lequel nous allons tout de suite revenir est celui de variable. Une variable en Maple est, comme dans tout langage de programmation ou même en mathématiques en général, une lettre (ou plus généralement un groupe de caractères) désignant un nombre (ou tout autre type d'objet) destiné à prendre plusieurs valeurs au cours de l'exécution de la commande ou du programme. En fait, informatiquement parlant, une variable est simplement une adresse d'emplacement dans la mémoire de la machine, où sera stockée la valeur.

Pour affecter une valeur à une telle variable, on utilise le :=, par exemple [> $a := 3$; créée une variable de type entier et y affecte la valeur 3. On peut ensuite utiliser la variable dans un calcul, qui sera effectué en la remplaçant par sa valeur au moment du calcul. Il est par ailleurs fortement conseillé de toujours commencer une feuille de calcul par un [> *restart*; qui réinitialise toutes les variables, pour éviter de se retrouver avec des valeurs définies lors de sessions précédentes. On peut de la même façon définir des fonctions sous la forme [> $f := x- > 2t^2 - \cos t$ (on peut mettre plusieurs variables, bien entendu). On peut aussi utiliser l'opérateur de composition @ dans la définition des fonctions : [> $f := t- > \sin t : g := t- > t^2 : h := g@f$ définit la fonction $h : t \mapsto \sin^2 t$.

Toutes les variables sont accompagnées d'un type qui restreint les possibilités de calcul qu'on peut effectuer avec la variable en question. On ne pourra ainsi évidemment pas dériver un nombre entier, mais dériver une fonction ne posera pas de problème. PNous étudierons en cours de semestre plusieurs types de variables classiques, contentons-nous pour l'instant d'une petite liste de types numériques courants : **integer** (nombres entiers) ; **fraction** (ou **rational**) pour les nombres rationnels ; **float** pour les nombres réels (sous forme décimale) ; **complex** pour les nombres complexes. Autres types utiles (mais non numériques) : le type **boolean** qui ne peut prendre que les valeurs true (vrai) et false (faux), et sera le type des tests. Par exemple, la commande [> $a := 3 > 2$; crée une variable a de type booléen qui prend en l'occurence la valeur vrai. Remarquons au passage la distinction effectuée en Maple entre le symbole := qui est utilisé pour l'affectation des variables, et le simple = qui constitue un test d'égalité (si vous tapez la commande [> $a = 2$;, Maple renverra true ou false selon que la valeur de la variable a est 2 ou non). Un autre type utile est le type **string** (chaine de caractères) qui permet d'utiliser des variables qui sont tout simplement des mots ou des phrases (ça n'a plus grand chose de mathématique).

Structures de contrôle

Structure if then else :

L'exemple vu la semaine dernière (résolution d'équations du second degré) montre déjà la nécessité d'introduire dans le moindre algorithme une possibilité de tester une condition. La structure correspondante en Maple a la syntaxe suivante :

if condition **then** instruction
elif condition2 **then** instruction2
else condition3 **end if** ;

Les deux dernières lignes sont optionnelles, on peut n'avoir qu'une possibilité. On peut aussi se passer d'elif si on n'a que deux possibilités, et bien sûr ajouter d'autres elif si besoin est. Un exemple d'instruction Maple faisant intervenir une boucle if : [> if $a < b$ then $c := b$ else $c := a$ end if ; (on affecte à la variable c la plus grande valeur parmi a et b).

Structure for :

Il est également extrêmement fréquent à l'intérieur d'un programme d'avoir à enchaîner un certains nombres de calculs similaires, la quantité de calculs pouvant par ailleurs dépendre d'un paramètre. Pour cela, on utilise une boucle for, dont la structure est la suivante :

for variable **from** valeurmin **to** valeurmax **by** pas **do** instruction **end do** ;

Le pas et la valeur de départ valent par défaut 1, il est inutile de les préciser si on ne veut pas les modifier. Un exemple de calcul faisant intervenir une boucle for :
[> $s := 0$: for k from 0 to 100 do $s := s + k$ end do : s;
affiche la somme des entiers de 0 à 100 (qui peut par ailleurs se faire plus simplement avec un sum).

Structure while do :

Cette dernière structure ressemble beaucoup à la précedente puisqu'on va également effectuer une boucle d'instructions, mais cette fois-ci, le nombre de passages dans la boucle n'est pas spécifié, on l'arrête simplement quand une certaine condition est vérifiée :

while condition **do** instruction **end do** ;

Cette structure est utile quand on ne sait pas le nombre d'étapes nécessaires avant d'atteindre une certaine condition, par exemple
[> $a := 1 : n := 0$: while $abs(a - sqrt(2)) > 10^{-10}$ do $a := 0.5 * (a + 2/a) : n := n + 1$ end do : n;
Ce petit programme calcule les valeurs successives de la suite définie par $u_0 = 1$ et $u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right)$, jusqu'à obtenir une valeur un terme dont la distance à $\sqrt{2}$ soit inférieure à 10^{-10}. Il affiche alors le nombre de termes à calculer avant d'atteindre cette précision. Il va de soi que le programme ne va fonctionner que parce que cette suite converge vers $\sqrt{2}$; si on met dans une boucle while une condition qui est toujours vérifiée, le programme ne terminera jamais.

Procédures

Disposer uniquement des structures de contrôle sans structure englobante est complètement improductif : essayez d'écrire le petit programme de la semaine dernière sur la résolution d'équations du second degré uniquement avec ce qu'on a vu jusqu'ici et vous serez très embêtés. Une telle structure existe en Maple, c'est la procédure. La syntaxe pour en définir une est la suivante :

[> nom := proc(paramètre1 : :type1, . . ., paramètrez : :typez)
local variable1 . . .variablek ;
instructions ;
end proc ;

La première ligne n'est rien d'autre qu'une affectation, une procédure étant en fait une variable comme les autres (mais d'un type un peu particulier). Entre parenthèses, on indique les paramètres utilisés par la procédure (ainsi que leur type si on le souhaite, mais c'est facultatif). Ces paramèters correspondent aux données du problème, autrement dit à des valeurs qui seront choisis par l'utilisateur de la procédure, extérieurement au fonctionnement de la procédure. Dans le cas de la résolution de l'équation du deuxième degré, les paramètres seraient les trois coefficients a, b et c. Ces paramètres ne peuvent absolument pas être modifiés à l'intérieur de la procédure, si vous souhaitez le

faire malgré tout, il faudra d'abord copier leur valeur dans une variable auxiliaire et modifier cette denrière.

Justement, les variables intervenant à l'intérieur de la procédure sont déclarés sur la deuxième ligne à l'aide du mot-clé **local**. Ce sera le cas par exemple de la variable Δ pour les équations de second degré. Le mot local signifie que ces variables ne sont définies et destinées à être utilisées qu'à l'intérieur de la procédure. Si vous refaites appel à une variable locale hors procédure, Maple ne va pas comprendre. On peut également utiliser ce qu'on appelle des variables globales dans une procédure (alors déclarées via le mot-clé global), mais c'est en général déconseillé (on peut même faire des horreurs du genre donner le même nom à une variable locale et à un paramètre, mais là c'est vraiment chercher les ennuis). La déclaration du type de ces variables est facultative, et même la déclaration des variables elle-même peut être omise en Maple, mais pour prendre de bonnes habitudes, on la fera toujours.

Après les déclaration, on peut aligner autant d'instructions qu'on le souhaite, jusqu'à conclure la procédure par l'instruction end proc ; (obligatoire). Par défaut, la procédure renverra à l'écran le résultat du dernier calcul effectué. Si on souhaite modifier cela, on ajoute des print en cours de procédure (pour forcer un affichage), on met l'instruction NULL ; avant la fin de procédure (dans ce cas, elle ne renverra rien), ou on insère un return(x) ; qui renverra la valeur de x si on l'atteint mais qui a aussi l'intéressante propriété d'interrompre illico la procédure.

Dernière remarque pour finir, comme signalé la semaine dernière, Maple sait gérer les procédures récursives, vous pouvez donc à l'intérieur même d'une procédure écrire une instruction faisant appel à cette procédure. On fera dans ce cas très attention à ne pas écrire de procédure récursive qui boucle sans jamais s'arrêter.

Un petit exemple de procédure déterminant le plus grand parmi trois nombres donnés par l'utilisateur :

```
[> maxi := proc (a,b,c)
local d ;
if a > b then d := a else d :=b ;
if d > c then d else c ;
end proc ;
```

1 Petits exercices

1. Écrire en Maple une procédure effectuant la résolution des équations du second degré.
2. Écrire une procédure Maple récursive calculant la valeur de $n!$.
3. Écrire une procédure Maple calculant $\displaystyle\sum_{k=1}^{n} k^3$.
4. On définit deux suites a_n et b_n par $a_0 = 1$, $b_0 = 2$, et pour $n \geq 1$, $a_{n+1} = \dfrac{a_n + b_n}{2}$ et $b_{n+1} = \sqrt{a_n b_n}$. Écrire une procédure Maple qui calcule les valeurs de a_n et b_n (on essaiera d'écrire une version non récursive et une version récursive).
5. La suite de Syracuse est définie de la façon suivante : u_0 est un entier positif, et ensuite on a $u_{n+1} = \dfrac{u_n}{2}$ si u_n est pair, $u_{n+1} = 3u_n + 1$ sinon. Écrire une procédure Maple qui, une fois donné le premier terme, calcule le rang du premier terme de la suite valant 1, ainsi que la plus grande valeur obtenue lors des calculs des termes successifs de la suite.
6. Écrire une procédure Maple calculant la valeur du plus petit entier n pour lequel $\displaystyle\sum_{k=1}^{n} \frac{1}{k} > a$, le réel a étant un paramètre de la procédure.