TD Info n°2

PCSI 1 Lycée Pasteur

13 septembre 2007

1 Variables et affectations

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 cessions précédentes. On peut de la même façon définir des fonctions sous la forme [$> f := x- > 2t^2 - \cos t$ (on peut mettre plusirurs 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 $h : t \mapsto \sin^2 t$.

2 Principes de programmation ; boucles

Qu'est-ce qu'un algorithme ?

C'est une succession de calculs élémentaires permettant la résolution mécanique d'un problème donné. Par élémentaires il faut comprendre que ce sont des calculs qui peuvent être effectués directement par la machine. Par mécanique, il faut comprendre qu'il ne doit y avoir à aucun moment un choix de la part de la machine. Par contre, et c'est la base de la programmation, on peut lui demander de vérifier certaines conditions et d'agir en fonction du résultat trouvé. Par exemple, pour résoudre une équation du second degré dans $\mathbb{R}$, on a l'algorithme suivant (écrit en pseudo-langage pour l'instant), les données étant les coefficients a, b et c du trinome :

- Calculer $d := b^2 - 4 * a * c$;
- Si $d < 0$, alors écrire « Pas de solution réelle » ;
- Si $d = 0$, alors afficher $x = -\dfrac{b}{2a}$;
- Si $d > 0$, alors afficher $x_1 = -\dfrac{b+\sqrt{d}}{2a}$ et $x_2 = -\dfrac{-b-\sqrt{d}}{2a}$;

En fait, il y a quelques subtilités supplémentaires (il faut vérifier que $a \neq 0$, notamment) et des améliorations possibles (si d n'est pas négatif ni nul il est forcément positif), mais l'idée est là.

Structures de contrôle

Structure if then else : Le premier exemple précédent 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 grade valeur parmi a et b).

Structure for : Il est également extrêmement fréquent à l'intérieur d'un programme d'avoir à enchainer 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 esy inutile de les préciser si on ne veut pas les modifier. Un exemple de calcul faisant inervenir 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.

3 Petits exercices

1. Écrire une instruction en Maple calculant la somme des cubes des entiers de 1 à 1000 (sans utiliser sum, bien entendu).
2. On définit deux suites a_n et b_n par $a_0 = 1$, $b_0 = 2$, et pour $n \geq 1$, $a_{n+1} = \frac{a_n + b_n}{2}$ et $b_{n+1} = \sqrt{a_n b_n}$. Écrire une ligne de commande Maple qui calcule les valeurs de a_15 et b_15.
3. La suite de Syracuse est définie de la façon suivante : u_0 est un entier positif, et ensuite on a $u_{n+1} = \frac{u_n}{2}$ si u_n est pair, $u_{n+1} = 3u_n + 1$ sinon. Écrire une ligne de commande Maple qui, une fois donné le premier terme, calcule le rang du premier terme de la suite valant 1 (il est conjecturé que, quelle que soit la valeur de u_0, la suite de Syracuse finit par atteindre la valeur 1, à partir de quoi elle boucle sur les valeurs 1, 4, 2, mais personne n'a encore réussi à le prouver !).