Structures de données

Nicolas Gast

3 octobre 2005

Après avoir vu comment manipuler simplement des formules simples la semaine dernière, nous allons maintenant aborder des structures de données plus complexes.

1 Types de données

Une commande dans maple est une tapée sous forme de chaine de charactère. Le polynôme $x^2 - 1$ est représenté par le mot "`x^2-1`". Lorsque l'on tape une ligne en Maple, celle-ci est ensuite interprétée : Mapletransforme la ligne tapée en un ensemble de donnée qu'il va pouvoir traiter.

Afin de mieux représenter en mémoire ces données, celles-ci sont classée en plusieurs "types". Chaque type représente une brique de base permettant de construire des expressions beaucoup plus complexes.

La commande `whattype(...)` permet de connaitre le type d'une expression. Appliquée à une expression simple, celle-ci renvoie le type de l'expression en question ; appliquée à une expressions composée, celle-ci renvoie le type de l'oppération de plus haut niveau[1].

Les types sont très variables : type numériques (integer, float, fraction, ...), type algébrique (+, *, ^), type d'égalités (= , ¡¿, ¡=) relations logiques, ...

```
> #types numériques
  whattype(3); whattype(3.2); whattype(5/2); whattype(I);
                                    integer
                                     float
                                   fraction
                            complex(extended_numeric)

> #type algébriques
  whattype(x+y); whattype(x-y); whattype(x*y); whattype(x/y); whattype(x^y);
                                        +
                                        +
                                        *
                                        *
                                        ^


> # type d'égalités
  whattype(x<y); whattype(x=y); whattype(x <> y);
                                        <
                                        =
                                       <>

> #relations logiques
  whattype(A or B); whattype (A and B); whattype (not A);
                                       or
                                       and
                                       not

> #autres types
  whattype( f(x) ); whattype (sin(x)); whattype(proc(x) x^2; end);
                                    function
```

[1] **Oppération de plus haut niveau :**lorsqu'on a à faire avec une expression complexe, l'oppération de plus faut niveau est celle qui est effectuée en dernier. Par exemple dans le cas de $1 + 3 * 7$, on effectue d'abord $3 * 7$ puis le $+$ de $1 + (3 * 7)$. Dans le cas de $\int_1^x (4t^3 + 2)dt$, la dernière oppération est $\int$

```
                                    function
                                    procedure
> whattype (a[1]); whattype( 1..2 ); whattype('a'); whattype("a");
                                     indexed
                                       ..
                                     symbol
                                     string

> # ATTENTION, Maple évalue la fonction avant de retourner son type. Exemples :
> whattype(1+2); whattype(Pi); whattype(evalf(Pi));
                                     integer
                                     symbol
                                      float
```

2 Suites et Listes

Les suites (exprseq) et les listes (list) sont des stuctures de données simples et très importantes en Maple. Bien que se ressemblant fortement, elles ont différentes propriétés qu'il vous faudra exploiter par la suite.

2.1 Les Suites

Elles sont les expressions composées les plus simples. Une suite est un ensemble finie d'expressions, écrite dans un ordre donné et séparés par des virgules.

```
> a,b,sin(x),3,Pi, arctan(3*x+2);
  whattype(%);
                    a, b, sin(x), 3, Pi, arctan(3 x + 2)

                                    exprseq
```

On peut assigner rapidement plusieurs variables grace aux suites grâce aux listes.

```
> p,q,r := 34, evalf(Pi), 2.5;
                        p, q, r := 34, 3.141592654, 2.5

> # Lorsqu'il y a plusieurs Le résultat de la commande solve est une liste.
  racine1, racine2, racine3 := solve (x^3+x^2+x+1,x);
                    racine1, racine2, racine3 := -1, I, -I
```

2.2 Les Listes

Une liste se représente de la même façon qu'une suite mais est entourée par deux crochets. Elle décrits les mêmes ensembles d'éléments.

```
> [p,q,r];
  whattype(%);
                             [34, 3.141592654, 2.5]
                                      list
> #Attention~: on ne peux pas assigner plusieurs éléments à l'aide d'une liste
  [a,b,c] := [1,2,4];
Error, invalid left hand side of assignment
```

2.3 Quelques commandes sur les listes et les suites

Il est facile de passer d'une suite à une liste et inversement :

```
> suite := 1,a,b;
                              suite := 1, a, b

> liste := [suite]; whattype (liste);
                             liste := [1, a, b]
                                    list

> suite2 := op(liste);
                              suite2 := 1, a, b
```

Pour créer une suite, la commande `seq(...)` est très utile.

```
> seq (f(i), i=1..10); # crée une suite arbitraire
         f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), f(10)

> seq ( k^2 , k=5..20 ); # la suite des carrés de 5 à 20
   25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400

> seq ( cos(k*Pi/4), k=0..7); # la suite des cos(k*Pi/4)
                        1/2       1/2        1/2      1/2
                       2         2          2        2
                  1, ----, 0, - ----, -1, - ----, 0, ----
                       2         2          2        2
```

On peut facilement optenir le n$^{\text{ième}}$ terme d'une suite ou d'une liste ou même obtenir une sous-suite composée des termes n à m d'une suite ou liste.

```
> liste := [1,a,b]; suite := e,f,g;
                             liste := [1, a, b]
                              suite := e, f, g

> suite [1]; liste [1];
                                    e
                                    1

> suite [-1]; # le dernier terme de la suite (-2 pour l'avant dernier, etc...)
                                    g

> liste[2..3]; # la sous-suite composée des termes 2 à 3.
                                 [a, b]
```

On peut bien entenfu faire des listes de listes, concaténer deux suites (attention pour les listes ceci est plus délicat !).

```
> ll := [ 5, [1,2,3], [3,4] ];
                        ll := [5, [1, 2, 3], [3, 4]]
> suite1 := 1,2; suite2 := 3,4,5;
                             suite1 := 1, 2
                            suite2 := 3, 4, 5
> suite3 := suite1,suite2;
                          suite3 := 1, 2, 3, 4, 5

>  #Attention pour les listes, il faut passer par la commande "op"
> liste1 := [a,b]; liste2 := [4,k,t];
                             liste1 := [a, b]
                            liste2 := [4, k, t]
> liste3 := [ op (liste1), op(liste2)];
                          liste3 := [a, b, 4, k, t]
```

Pour substituer un élément dans une liste, utilisez la commande subs ou subsop.(ATTENTION : cela ne modifie pas la liste de départ !)

```
> liste := [1,a,4,b,c,4,a];
                        liste := [1, a, 4, b, c, 4, a]
> subs(4=quatre, liste); # je remplace les occurences de 4.
                        [1, a, quatre, b, c, quatre, a]
> subs(4=quatre, a=aahh, liste); # je remplace les occurences de 4 et de a
                     [1, aahh, quatre, b, c, quatre, aahh]
> subsop(4=quatre, liste); # je remplace le 4èm élément.
                          [1, a, 4, quatre, c, 4, a]
> liste; #notons que la liste n'est jamais modifié
                             [1, a, 4, b, c, 4, a]
```

On peut aussi utiliser la commande sort pour trier une liste.

3 Les ensembles

Maple sait manipuler des ensembles (`set`). Ils ont le même sens qu'un ensemble mathématique : c'est une suite d'éléments non ordonnés et sans répétitions.

Quelques commandes :

```
> e1 := {1,1,4,k};  #création d'un ensemble (il n'y a pas de répétition)
                                 e1 := {1, 4, k}
> suite := 4,3,5,1,a,4;
                            suite := 4, 3, 5, 1, a, 4
> {suite};  #création d'un ensemble à partir d'une suite.
                                 {1, 3, 4, 5, a}
> op (%);   #opération inverse
                                  1, 3, 4, 5, a
> entiers1_100 := { seq(i, i=1..100)}: #donne l'ensemble des entiers de 1 à 100

# on peut faire des union, intersection et différences d'ensembles.
> e1 := {4,50,102,8};
                               e1 := {4, 8, 50, 102}
> e1 minus entiers1_100;
                                     {102}
> e1 intersect entiers1_100;
                                   {4, 8, 50}
> e1 union {0,-1};
                              {-1, 0, 4, 8, 50, 102}

# la commande select permet de selectionner selon un critère. Elle
# s'utilise sous la forme   select ( boolean procedure, set);
> select ( proc(n) n mod 10 = 0; end ,entiers1_100) ;
                    {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
```

4 Les tableaux et les tables

4.1 Les tableaux

Un tableau ("array") de taille $m_1 \times m_2 \times \ldots \times m_p$ est une structure contenant $m_1 m_2 \ldots m_p$ entrées et numérotée par le o-uplet $(i_1, i_2, \ldots, i_n)$.

Si la taille est $1 \times n$ c'est un vecteur, si elle est $m \times n$ c'est une matrice.

Pour créer un tableau, on utilise la commande `array`.

```
# création d'un tableau de taille 2x2
> tableau := array (1..2, 1..2);
                      tableau := array(1 .. 2, 1 .. 2, [])
> print(tableau);
                        [tableau[1, 1]    tableau[1, 2]]
                        [                               ]
                        [tableau[2, 1]    tableau[2, 2]]
```

```
#on peut aussi remplir le tableau quand on le crée.
> tableau := array (1..2, 1..2, [ [1,2], [a,b]]);
                                          [1    2]
                              tableau := [      ]
                                          [a    b]
> array( [[1,2],[3,4]]);
                                    [1    2]
                                    [      ]
                                    [3    4]
```

Remarques : un tableau n'est pas une liste de listes, on peut utiliser la commande convert pour passer de l'un à l'autre. La principale différence est que la taille d'un tableau est fixée à l'avance et ne peut être modifié au cours du temps (et si, cela a un intérêt !)

4.2 Les tables

Une table ressemble beaucoup à un tableau mais les indices peuvent être une expression et non plus uniquement des nombres. Elles ne nous servirons pas ici mais peuvent être assez utiles dans le cadre de programmation plus avancée.

```
> masse := table( [ Pierre = 32, Paul = 56, Remi = 44]);
              masse := table([Remi = 44, Paul = 56, Pierre = 32])
> masse[Pierre];
                                    32
```

5 Exercices

Exercice 1

Note : pour définir une fonction dans Maple, utilisez une commande du type $f := x -> x^2 + x + 1$

1. Définir sous forme d'expression puis d'une fonction f le polynôme $x^3 + 4 * x - x + 5$.
2. Calculer la valeur de f en 1 et en 7. Développez et simplifiez f(a+b).
3. Dériver les expressions (commande `diff`) $\ln(x) + e^{5*x}$, $g(x) = \sin(x) - \cos(x) + tan(x)$
4. Trouver la dérivée troisème de $g(x)$.
5. Soit $f(x, y) = \frac{x^3 y^2}{x^4 + y^2 + 1}$. Calculer les dérivées partielles de f.
6. Déterminer un développement limité à l'ordre 4 de $cos(exp(x))$
7. Calculer une primitive de $f(x)cos(x)$ (f comme à la question 1)
8. Calculer l'intégrale entre 1 et 2 de cette même fonction.

Exercice 2

1. Construisez la liste (ordonnée) des 200 premiers nombres premiers (`isprime` permet de savoir si un nombre est premier ou non)
2. Divisez tous ces nombres par 17 et appliquez la fonction `frac(...)` pour ne garder que la partie fractionnaire des nombres.
3. Multipliez ces éléments par 17 et construisez une nouvelle liste composée du résultat.
4. Convertissez cette liste en un ensemble et donnez-en le nombre d'éléments.

Exercice 4

1. Créez une liste L2 constitué des carrés compris entre 1000 et 10000.
2. Créez une liste L3 constitué des cubes compris entre 1000 et 10000.
3. Combien y-a-t'il d'éléments en commun dans les deux listes L2 et L3. Combien y-a-t'il d'éléments dans les deux listes réunies ?
4. Répetez la même chose avec L3 et L4 où L4 est constitué des puissances $4^{\text{ièmes}}$.

Exercice 5 - Décomposition d'un entier en base b

Écrire un programme rendant une suite représentant la décomposition d'un entier $n > 0$ en base b.

Exercice 3

1. Écrivez une procedure `n_prem := proc(n)` rendant la liste de n premiers nombres premiers.
2. En utilisant la procedure ci dessus, écrivrez une procedure `double_somme := proc(n)` rendant la liste des sommes des paires de nombres premiers entre 1 et n.
3. En utilisant la procedure ci-dessus, montrez que tous les nombres pairs entre 1 et 1000 s'écrivent sous la forme d'une somme de deux entiers premiers.

Ce problème est connu sous le nom de "conjecture de Goldbach" : tout nombre pair strictement supérieur à 2, peut-il s'écrire comme somme de deux nombres premiers ? Pour l'instant, cela a été vérifié pour tous les nombres jusuq'à 2.10^{16}. Démontrez le et vous finirez riche !

Exercice 6 - Décomposition en facteur premiers

- Écrire un programme qui rend une liste représentant la décomposition d'un entier n en facteurs premiers.
- Écrire un programme prenant en entrée la décomposition d'un entier en facteur premier et rendant l'entier en question.
- Vérifier que vos deux programmes sont bien l'inverse l'un de l'autre.

Bonus

- Multiplication de polynômes
- Tri "bulles"