Travaux Pratiques sur le système MuPAD n°5

Cours d'Informatique de l'École Centrale d'Électronique

—Ingénieur Deuxième Année—

Cette feuille porte sur le système de calcul formel *MuPAD*. Les objectifs sont la découverte des possibilités et la mise en évidence des limitations d'un tel système de calcul.

Le système *MuPAD* est développé depuis 1989. La première version officielle (version 1.0) date d'août 1992. Ont suivi à peu près tous les ans des nouvelles versions : version 1.1 en Mars 1993, version 1.2 en Mars 1994. Son développement rapide tient essentiellement à sa conception particulière qui s'est appuyée sur des modules existants.

C'est un système de calcul formel—*computer algebra system*. Son intitulé de système de calcul formel tient au fait qu'il a la capacité d'effectuer à la fois des calculs numériques et symboliques. C'est aussi un langage de programmation qui vient compléter le moteur de calcul. Il se distingue pourtant de ses concurrents directs tels que *Maple* par exemple par quatre points :

- *MuPAD* dispose d'une interface graphique qui autorise l'accès à une aide en ligne hypertexte, aux graphiques et à un *debugger*) ;
- *MuPAD* est le premier système à offrir des primitives de calcul parallèle à ses utilisateurs ;
- *MuPAD* offre la possibilité d'inclure dynamiquement des modules objets développés par les utilisateurs ;
- et enfin, *MuPAD* est gratuit pour les institutions non commerciales.

MuPAD est ainsi un programme interactif de calcul symbolique, qui dispose d'un langage de programmation interprété. Il n'est pas le mieux adapté pour faire des calculs numériques intensifs à cause de la surcharge de travail amenée par son interpréteur, mais il est un bon outil pour effectuer des calculs numériques en grande précision, en comparaison avec les langages de programmation plus classiques comme *Pascal* ou *C*.

La partie suivante met en évidence, sous forme de plusieurs remarques, les points clés de l'utilisation et de la programmation de *MuPAD*. Ces quelques connaissances de base vous seront utiles pour la résolution des exercices qui suivent.

1 Utiliser et programmer

- La commande `mupad` permet de lancer une session *MuPAD* (`xmupad` est la version donnant accès à l'interface graphique).
- Chaque commande *MuPAD* doit se terminer par un '`;`'.
- Pour supprimer l'affichage, on remplace le '`;`' par '`:`'.
- Le caractère `%` désigne le dernier résultat. Suivi d'un entier i, il fait référence au i-ème dernier résultat (c'est la commande `last` qui est en fait appelée).
- L'opérateur `:=` est l'opérateur *MuPAD* d'affectation d'une variable.
- *MuPAD* réalise une arithmétique exacte sur entiers.
- La fonction `float` fournit une approximation décimale de son argument.
- Les commandes `sum` et `product` permettent de manipuler respectivement des sommes et des produits.
- Les guillements `"` délimitent une chaîne de caractères.
- La commande `?` doit être un premier réflexe en cas de problème. Elle donne de l'aide sur une commande donnée.

- La commande `loadlib` permet de charger une librairie de fonctions.
- La commande `quit` termine une session *Maple*.
- Programmer *MuPAD* pour manipuler des formules, c'est basiquement programmer des manipulations d'arbres syntaxiques.
- La fonction `type` : `type(f, t)` retourne `TRUE` si l'expression `f` est du type `t`.
- La fonction `type` retourne le type d'une expression.
- La fonction `nops` retourne le nombre d'opérandes d'une expression.
- La fonction `op` permet d'extraire une des opérandes d'une expression.
- La syntaxe d'un appel de fonction est $f(a_1, a_2, \ldots)$ où f est le nom de la fonction et $a_1, a_2, \ldots$ sont les arguments.
- Une instruction conditionnelle s'écrit :

  `if` *condition* `then`
      suite_d'instructions
  [`elif` *condition* `then` *suite_d'instructions*]*
  [`else` *suite_d'instructions*]
  `end_if`

- Une extension de cette instruction conditionnelle est l'instruction d'aiguillage :

  `case` *expression*
  [`of` *expression* `do` *suite_d'instructions*]+
  [`otherwise` *suite_d'instructions*]
  `end_case`

- Une instruction de boucle `for` s'écrit :

  [`for` *nom*] [`from` *expression*] [`to` *expression*] [`step` *expression*] `do`
      suite_d'instructions
  `end_for`

  où alors :

  [`for` *nom*] [`from` *expression*] [`downto` *expression*] [`step` *expression*] `do`
      suite_d'instructions
  `end_for`

- Une instruction de boucle `while` s'écrit :

  [`while` *condition*] `do`
      suite_d'instructions
  `end_while`

- Une instruction de boucle `repeat` s'écrit :

  [`repeat`
      suite_d'instructions
  [`until` *condition*] `end_until`

- Un `break` permet de sortir de la boucle la plus intérieure dans laquelle nous nous trouvons.
- Un `next` permet de passer à l'itération suivante dans la boucle la plus intérieure où nous nous trouvons.
- Une *séquence* est une suite d'expressions séparées par des virgules.
- L'instruction f`(i)$i=`m`..`n permet de créer une séquence. Une autre syntaxe est donnée par f`(i)$i=`a).
- Pour accéder à la `i`-ème entrée d'une séquence `s`, on utilise l'expression `s[i]`.
- Une *liste* est une structure de données permettant de regrouper des objets ensemble. Les crochets servent à créer une telle structure.
- La liste vide est notée `[]`.
- Un *ensemble* est une structure de données permettant de regrouper des objets ensembles, sans répétition. Les accolades servent à créer une telle structure.
- L'ensemble vide est noté $\{\}$.
- La fonction `table` permet de créer une table. une collection d'objets.
- Les *tableaux* sont des tables ayant des indices entiers pris dans un intervalle déterminé à la création du tableau et de dimension fixée à l'avance aussi.
- La fonction `array` permet de créer un tableau.

- La fonction `read` lit un fichier de commandes *MuPAD*.
- La fonction `write` permet de sauver des résultats dans un fichier.
- Une procédure se définit par :

```
proc( séquence_paramètres )
      [ local séquence_noms; ]
      [ option séquence_options; ]
      [ begin
            instructions
end_proc
```

- La commande `return` réalise une sortie directe de la fonction. La valeur retournée par la fonction est alors, la séquence d'expressions—éventuellement vide—passée en argument du `return`.
- La commande `error` peut être utilisée pour générer un message d'erreur au sein d'une procédure.
- Un appel de fonction avec des paramètres se fait en passant à la fonction la séquence de ces paramètres.
- La procédure `map` prend en argument une expression `e` et une procédure `f`. Elle applique alors `f` à chaque opérande de l'expression `e`.
- La commande `unapply` permet de créer une procédure, à partir d'une expression.
- L'expression `args(0)` est le nombre de paramètres correspondant à l'appel courant.
- La valeur de la variable `procname` est le nom de la procédure courante.
- La rubrique `option` dans la définition d'une nouvelle procédure, permet de positionner les propriétés de cette procédure.
- Lorsque l'option `remember` est positionnée, à la fin de chaque appel de la procédure, une entrée est ajoutée dans une table associée à la procédure. Cette entrée a pour objet de se souvenir du résultat calculé, en fonction des arguments spécifiés.

2 Exercices de découverte

Voici quelques exercices qui devraient vous aider à comprendre les différents points de la section précédente. Ces exercices doivent être compris avant d'envisager de passer à la suite.

▸ **Exercice 1.** *Comment commencer une session MuPAD ? Quel est le prompt proposé par le logiciel ?*

▸ **Exercice 2.** *Entrez l'instruction* `4/6;` *suivie d'un* <RETOUR CHARIOT>. *Quel affichage obtenez-vous ? Essayez la commande* `%;`, *puis une instruction plus compliquée comme* `2*% + 1;`. *Qu'affichent les lignes suivantes ?*

```
>> % + %2:
>> %;
                ...
>> r := %1 * %2;
              ......
```

▸ **Exercice 3.** *Il existe en MuPAD un certain nombre de constantes mathématiques prédéfinies. Quelles sont-elles ? Vérifiez votre réponse à l'aide de la fonction* `type` *qui retourne le type de son argument.*

▸ **Exercice 4.** *MuPAD fait la différence entre les minuscules et les majuscules donc nous devons être bien sûr d'utiliser les bonnes lettres par exemple pour référencer ces constantes. Vérifiez que les noms* `pi` *et* `PI` *ne sont pas équivalents à l'aide par exemple de la commande* `float` *qui fournit une approximation décimale de son argument.*

▸ **Exercice 5.** *MuPAD réalise une arithmétique exacte sur entiers. En fait, MuPAD réalise des calculs sur des entiers plutôt que sur des nombres décimaux. L'utilisation des décimaux est tout à fait possible mais correspond à un ordre explicite de l'utilisateur. Regardez le résultat des commandes suivantes :*

```
>> 2*3+2/7;
                    .....
>> 2.0*3 + 2/7;
                 ...........
>> sin(PI/3);
                    .....
>> float(%);
                 ...........
```

▸ **Exercice 6.** *Il n'y a pas de limite sur la longueur des entiers en MuPAD. Il est ainsi naturel de calculer avec des entiers de plusieurs centaines de chiffres. Calculez* 2^{100}. *D'un autre côté, il est possible de faire de l'arithmétique décimale avec plus que les 10 chiffres, par défaut. C'est la variable* `DIGITS` *qui stocke le nombre de chiffres utilisés pour les calculs décimaux. Fixez la variable* `DIGITS` *à 50 et lancez* `float(sin(PI/3));`.

▸ **Exercice 7.** *Les sorties de MuPAD ne sont pas toujours dans leur forme la plus simplifiée. Il existe un certain nombre de commandes telles que* `collect`, `combine`, `expand`, `Factor` *ou* `simplify` *qui sont à disposition pour simplifier ou arranger un résultat suivant une forme voulue. Développez avec MuPAD le polynôme* $(x+y)(x-y)-x^2$. *Factorizez le polynôme* $x^4+x^2y+2x^2+2x^3+2xy+2x+y+1$. *Simplifiez l'expression MuPAD* $exp(a+ln(b*exp(c)))$ *et l'expression* $(x^3-y^3)/(x^2+xy+y^2)$.

▸ **Exercice 8.** *Dérivez par rapport à* x *les expressions* $\ln(x)$, $\arcsin(ax)$, $x\ln(x)$. *Calculez les intégrales* $\int \sqrt{(}1-x^2)\,dx$, $\int_1^2 \ln x\,dx$.

▸ **Exercice 9.** *Calculez formellement les sommes des* `n` *premiers carrés, cubes, etc.*

▸ **Exercice 10.** *Analyser la réponse de MuPAD aux instructions suivantes :*

```
>> f:=(x^3+2*x^2-x)/(x^3-x^2+x-1):
>> int(f, x);
          ....................
>> diff(%, x);
          ....................
>> % - f;
          ....................
```

Comment obtenir le résultat espéré ?

▸ **Exercice 11.** *Une chaîne de caractères consiste en un certain nombre de caractères entourés par des guillements* `"`. *Que produisent les commandes suivantes ?*

```
> "Voila une chaine de caracteres";
    ............................
> "123abc";
               ........
> "prog.txt";
               ........
```

Certains caractères spéciaux tels que le `+`, *le* `.` *ou encore le* `/` *peuvent être présents dans une chaîne de caractères dès lors que celle-ci est entourée de guillements. Si ces guillements sont absentes, ces caractères spéciaux seront tout naturellement traités comme des opérateurs MuPAD normaux. Essayez les instructions :*

```
> 3+abc+4;
                .......
> "3+abc+4";
                .......
> repertoire/fichier;
               .........
> prog.txt;
               .........
```

Expliquez les deux derniers résultats.

▸ **Exercice 12.** *Si nous avons besoin d'information sur une fonction MuPAD, nous pouvons utiliser la commande MuPAD* `?`. *Cette commande donne accès à une foultitude d'informations sur le système MuPAD en général et sur les fonctions MuPAD en particulier. Entrez la commande* `?` *et lisez cette première page de documentation.*

▸ **Exercice 13.** *Les librairies de fonctions viennent augmenter le noyau de MuPAD. On parle habituellement de package pour référencer un tel ensemble de fonctions spécifiques. Une des plus classiques mais aussi des plus imposantes de ces librairies de part le nombre de ses fonctions concerne l'algèbre linéaire :* `linalg`. *Chargez ce package et calculez le déterminant de la matrice*

$$\begin{bmatrix} x-1 & 2 & 3 \\ 0 & x-2 & 2 \\ 2 & 1 & x-3 \end{bmatrix}$$

3 Exercices de révision

Cette section présente quelques exercices de révision pour voir si les notions présentées ont été assimilées.

▸ **Exercice 14.** *Nous avons vu que pour commencer une session MuPAD, nous utilisions la commande* UNIX `mupad` *par exemple. Cette commande accepte quelques options que nous nous proposons de regarder. Quelle est donc la commande qui devrait nous permettre d'obtenir de telles informations et quelles sont ainsi les options acceptées par la commande* UNIX `mupad` *par exemple ?*

▸ **Exercice 15.** *Une des premières remarques que nous avons faite au sujet du langage MuPAD, concernait l'utilisation du point-virgule (voire du deux-points). Nous avons vu que toute commande MuPAD doit se terminer par un de ces deux caractères. Il existe pourtant une exception à cette règle que nous ne nous sommes déjà pas privés d'utiliser.*

1. *Quelle est donc cette exception à la règle ?*
2. *Quelle commande MuPAD pouvons-nous lancer pour confirmer notre idée ?*

▸ **Exercice 16.** *Dans l'exercice 15, nous avons vu une exception à la règle de l'utilisation du point-virgule. En fait, il existe une autre exception qui concerne une commande MuPAD que l'on ne peut utiliser, par nature, qu'une seule fois par session. Quelle est donc cette commande ?*

▸ **Exercice 17.** *Le logiciel MuPAD met à disposition une librairie de fonctions concernant l'algèbre linéaire : la libraire (ou package)* `linalg`.

1. *Examinez la documentation de la librairie* `linalg`.
2. *Comment charge-t-on cette librairie dans MuPAD ?*
3. *Comment obtenir de l'aide sur une fonction d'un package ?*
4. *Essayez l'exemple donné dans l'aide concernant la fonction* `transpose` *du package* `linalg`.

▸ **Exercice 18.** *Nous avons vu qu'en MuPAD, le signe d'affectation est* `:=`. *À la différence des langages de programmation classiques, il est possible d'affecter à une variable une expression contenant d'autres variables ne possédant aucune valeur (variables libres ou formelles). Les polynômes sont un cas particulier de telles expressions. Un polynôme en une indéterminée x est considéré comme une expression avec une variable libre x. Il existe de nombreuses fonctions très utiles pour manipuler par exemple les polynômes en une indéterminée. La fonction* `Factor` *(*`factor`*) permet de factoriser un polynôme, alors que les commandes* `simplify` *et* `expand` *permettent respectivement de simplifier et développer une expression.*

Ces commandes sont très utilisées lors d'une session MuPAD et nous conseillons de tout de suite consulter leurs aides en ligne respectives.

- *Entrez le polynôme $p(x) = x^2 - x - 1$. Affectez-le à une variable* `p` *sans l'afficher.*
- *Calculez $p(x)^5$, développez puis refactorisez le résultat.*

- *Substituez y à x dans* `p` *en utilisant la commande* `subs`.
- *Trouvez les solutions de l'équation $p(y) = 0$ à l'aide de la commande* `solve`.
- *Vérifiez que les deux solutions calculées sont effectivement les racines du polynôme.*

▸ **Exercice 19.** *Refaites les questions de l'exercice précédent pour le polynôme $p(x) = x^2 + ax + b$.*

▸ **Exercice 20.** *En MuPAD, il est possible d'utiliser des variables muettes telles les indices de sommation ou les variables d'intégration par exemple. Calculez la somme*

$$s_1 := \sum_{i=1}^{n} i^3 .$$

▸ **Exercice 21.** *Calculez la somme*

$$s_2 := \sum_{j=0}^{n} j^2 \binom{n}{j} ,$$

où $\binom{n}{j}$ est le coefficient binomial. Ce coefficient est accessible en MuPAD par la commande `binomial(n,j)`.

▸ **Exercice 22.** *Nous nous proposons de faire quelques calculs avec la fonction* `int`.

1. *Calculez une primitive de la fonction $\sin^2(x)$.*
2. *Vérifiez que le résultat est correct en dérivant puis en simplifiant.*

▸ **Exercice 23.** *Faites de même avec les fonctions $\sin(1/x)$, $1/p$[1] et $1/(x^2+1)^k$.*

▸ **Exercice 24.** *Calculez la séquence des coefficients binomiaux $\binom{10}{i}$ pour $i = 0, \ldots, 10$. On utilisera la commande* `seq`.

[1] Où p est le polynôme introduit dans l'exercice 18.

▸ **Exercice 25.** *Les polynômes de Bernoulli $B_n(x)$ sont définis par leur série génératrice :*

$$\frac{te^{xt}}{e^t - 1} = \sum_{n=0}^{\infty} B_n(x) \frac{t^n}{n!} ,$$

où $x \in \mathbb{R}, t \in C, |t| < 2\pi$. Les nombres de Bernoulli sont par définition les nombres $B_n = B_n(0)$ pour $n \geq 0$. Ces nombres sont rationnels. Nous nous proposons de calculer les premiers nombres de Bernoulli en admettant que $B_0 = 1$ et $B_1 = -1/2$. Ces nombres peuvent être calculés par récurrence en utilisant la formule

$$B_n = \sum_{k=0}^{n} \binom{n}{k} B_k, \quad (n \geq 2) ,$$

sous la forme

$$B_{n-1} = -\frac{1}{n} \sum_{k=0}^{n-2} \binom{n}{k} B_k, \quad (n \geq 3) .$$

Nous donnons le début du calcul pour B_0 et B_1 :

```
>> B.0 := 1;

                    B0 := 1

>> B1 := -1/2;

                    B1 := -1/2
```

Quelle propriété évidente sur les nombres de Bernoulli pouvons-nous déduire de ces premiers calculs ?

▸ **Exercice 26.** *Calculez une approximation de l'irrationnel π avec 2000 décimales.*

▸ **Exercice 27.** *π peut être calculée par la formule de Chudnovsky suivante :*

$$\frac{1}{\pi} = \frac{6541681608}{640320^{3/2}}$$

$$\sum_{k=0}^{\infty}\Big(\frac{13591409}{545140134}+k\Big)$$
$$\frac{(6k)!}{(3k)!(k!)^3}\frac{(-1)^k}{640320^{3k}}.$$

Vérifiez cette assertion par un calcul effectif de π utilisant cette expression.

4 Structures de données et structures de contrôle

Avant de commencer, nous devons mettre en évidence une différence essentielle entre *MuPAD* et les langages traditionnels de programmation. Si un identificateur n'a pas été assigné d'une valeur, sa valeur est formellement lui-même. C'est ce que nous appelons un *symbole*. Les symboles sont utilisés pour représenter des inconnues dans des équations, des variables (indéterminées) dans des polynômes ou encore des indices de sommation par exemple.

▸ **Exercice 28.** *Entrez les instructions suivantes et analysez les résultats :*

```
>> p := x^2 + 4*x + 4;
              ....................
>> p;
              ....................
>> x;
                     ...
>> x := 3:
>> p;
                     ...
```

Suite aux intructions précédentes, calculez l'intégrale $\int p\,dx$ et conclure. Comment faire en sorte que `x` *retrouve sa valeur formelle ?*

▸ **Exercice 29.** *Les formules en MuPAD sont représentées par des arbres syntaxiques ou, pour être tout à fait précis, par des DAGs—Directed Acyclic Graphs. Comment sont codées les opérations de soustraction et de division ? Quels sont les différents types arithmétiques en MuPAD ? Notez la différence cruciale de résultat entre les deux commandes* `2/3;` *et* `2/3.0;`.

▸ **Exercice 30.** *On considère le polynôme $p := x^2y + 3x^3z + 2$. Comme nous le constatons,* `p` *est une somme de termes. Combien y en a-t-il ? Essayez d'extraire chacun des termes à l'aide de la fonction* `op`. *Analysez complètement l'expression* `p`.

▸ **Exercice 31.** *Les fonctions se comportent quelque peu comme des expressions quelconques. Commentez les résultats des commandes suivantes :*

```
>> nops(f(x,y,z));

                    .....

>> op(f(x,y,z), 1..3);

                    .....

>> op(f(x,y,z), 0);

                     ...
```

▸ **Exercice 32.** *Nous sommes désormais capables d'analyser la structure de n'importe quelle formule MuPAD car nous connaissons l'action de chacune des fonctions* `nops`, `op`, *etc. sur les différents objets que nous pouvons rencontrer dans une formule. Quel est le type du premier terme de la formule : $\sin(x1)^2(1-cos(\pi x2))$.*

▸ **Exercice 33.** *Nous nous attardons sur un problème d'évaluation qui peut arriver si nous ne faisons pas assez attention aux commandes entrées. Nous considérons le polynôme* `p` *que nous avions défini précédemment. Ici, nous avons assigné au nom* `p` *une formule qui contenait explicitement le symbole* `x`. *Que se passe-t-il si nous utilisons désormais le même nom* `p` *des deux côtés de l'assignation, par exemple pour l'instruction* `p := p + x;` *? Qu'en est-t-il si vous entrez la commande* `q:=q+x;` *et demandez ensuite la valeur de* `q`.

▸ **Exercice 34.** *Affichez à l'aide d'une boucle, les 25 premiers carrés.*

▸ **Exercice 35.** *Recherchez avec une boucle le plus petit entier premier supérieur à $10^{10}+1$. Proposez une version avec une boucle* `for` *puis avec une boucle* `while`*.*

▸ **Exercice 36.** *La notion de séquence est tout à fait essentielle. Elle apparaît réellement partout et c'est tout à notre avantage. Par exemple, nous avons déjà vu que les arguments d'une fonction étaient séparés par des virgules, et bien c'est parce qu'en fait ils sont simplement une séquence—au sens MuPAD—d'expressions. Entrez l'instruction* `s := 1, 4, 9, 16, 25;` *et demandez le type de* `s`*. Que conclure du résultat des instructions* `u := 1, (4, 9, 16), 25;` *et verb+u, u;+. Calculez le maximum de la séquence* `u`*.*

▸ **Exercice 37.** *Créez à l'aide d'une boucle la séquence des 10 premiers carrés. Faites de même avec l'opérateur* `$`*. Que pensez-vous des temps de création de la séquence lorsque l'on considère des très grandes séquences ? Que conclure ?*

▸ **Exercice 38.** *On considère l'expression $a := 3x^3+yx-11$. Créez la séquence des coefficients des puissances de x de cette expression.*

▸ **Exercice 39.** *Calculez la dérivée partielle 22^{eme} par rapport à x, de la dérivée partielle 23^{eme} par rapport à y de $sin(xy)$.*

▸ **Exercice 40.** *Complétez l'éxécution suivante :*

```
>> ll := [x, 1, 1-z, x];

                    . . . . . . . . . . . . . . . . . . . . . .

>> type(ll);

                    . . . . . .
>> ll[3];

                    . . . . . .
>> op(ll, 3);

                    . . . . . .

>> op(ll, 1..nops(ll));

                . . . . . . . . . . . . . . .
>> type(%);
                . . . . . . . . . . . . . . .
>> [%2];
                . . . . . . . . . . . . . . .
```

▸ **Exercice 41.** *Que produisent les instructions suivantes ?*

```
>> i := 4:
>> ll := [ op(ll, 1..i-1),
          op(ll, i+1..nops(ll)) ];

              . . . . . . . . . . . . . . . . . . .

>> ll := subsop(ll, 2=34);

              . . . . . . . . . . . . . . . . . . .

>> ll[2]:=NIL: ll;

              . . . . . . . . . . . . . . . . . . .
```

▸ **Exercice 42.** *Complétez l'éxécution suivante :*

```
>> s := {x, 1, 1-z, x};

              . . . . . . . . . . . . . . . . . . .
>> type(s);

                          . . .
```

Trouvez les opérations possibles sur les ensembles.

▸ **Exercice 43.** *Partant d'une séquence quelconque, créez une liste puis transformez-la en un ensemble (pour par exemple éliminez les répétitions d'éléments) et enfin reconstruisez une liste.*

▸ **Exercice 44.** *L'enjeu de cet exercice est la découverte des objets de type* `table`*. Complétez l'exécution suivante et commentez la nature des résultats :*

```
>> couleur[rouge] := red, rot;
          ..............................
>> couleur[bleu] := blue, blau;
          ..............................
>> couleur[jaune] := yellow, gelb;
          ..............................
>> table();
          ..............................
>> table(1=22,2=42);
          ..............................

>> F := table(sin=cos,cos=-sin):
>> op(op(F));
          ..............................
>> F[cos](PI/2);
                        ....
>> couleur;
            ......................
>> type(couleur);
                  .............
>> T[x] := a;
                  .............
>> T;
                  .............
```

▸ **Exercice 45.** *Un tableau est un type particulier de table. Quel est donc la définition exacte d'un tableau en MuPAD et à quel type MuPAD ces objets correspondent-ils ? Ordonnez en utilisant un tri à bulle un tableau quelconque.*

5 Exercices de révision

▸ **Exercice 46.** *Les expressions sous MuPAD sont complètement évaluées. Pour bien s'en rendre compte, complétez l'exécution suivante :*

```
>> c := b: b := a: a := 1: b;
                   ...
>> c;
                   ...
>> b := NIL: c;
                   ...
>> b;
                   ...
>> a;
                   ...
```

▸ **Exercice 47.** *Complétez l'exécution suivante :*

```
>> c := b: b := a: a := 1:
   b := NIL: c;
                   ...
>> b;
                   ...
>> a;
                   ...
```

▸ **Exercice 48.** *Il est possible de savoir, non pas la valeur d'une variable, mais la valeur qui lui a été assignée. Par exemple, après avoir entré les instructions* `a:=b: b:=1: a;` *nous aimerions savoir que* `a` *a reçu la valeur* `b` *et non* `1` *comme indiqué par le résultat de ces instructions. Comment devons-nous faire ?*

▸ **Exercice 49.** *Il existe une fonction en MuPAD qui retourne la liste de toutes les variables assignées (d'une valeur). Quelle est cette fonction ?*

▸ **Exercice 50.** *Voici quelques questions pour voir si nous avons les bons réflexes en ce qui concerne les expressions MuPAD.*

1. *Soit le polynôme $P = (x - y + 5)^5$. Quel est le troisième terme de l'expression développée de P ?*

2. *Combien P a-t-il de termes ?*

3. *Quels sont les types de P,* `factor(`P`)`, $(x+1)(x+2)$, x^2+3x+2 *et* $(x+1)(x+1)$ *?*

▸ **Exercice 51.**

1. *Construisez la séquence des carrés des entiers de 10 à 15 ;*

2. *Construisez la séquence des plus grands communs diviseurs de 12 et des entiers de 1 à 30 ;*

3. *Construisez la liste et l'ensemble de ces plus grands communs diviseurs.*

▸ **Exercice 52.** *Complétez et expliquez l'exemple suivant :*

```
>> x.i$i=1..5;
                 . . . . . . . . . . . . . . . . . .
>> i:=3: x.i$i=1..5;
                 . . . . . . . . . . . . . . . . . .
>> i:=3: x.i$hold(i)=1..5;
                 . . . . . . . . . . . . . . . . . .
```

▸ **Exercice 53.**

1. *Construisez la liste* $l = [x_1, x_2, \ldots, x_{30}]$ *;*

2. *À partir de la liste l, construisez la liste* $[x_4, x_5, \ldots, x_{30}]$ *; À partir de cette dernière liste, construisez la liste* $[x_{30}, x_{29}, \ldots, x_4]$ *;*

3. *Construisez de toute pièce cette même liste.*

▸ **Exercice 54.** *En utilisant la fonction* `time`*, étudiez la différence de temps entre la création de la séquence* $1, \ldots, 2^{10}$ *par une boucle et par un appel à l'opérateur* `$`(2)*.*

(2) On relancera *MuPAD* à chaque fois, afin de se retrouver dans les mêmes conditions d'exécution.

6 Programmation MuPAD

▸ **Exercice 55.** *Trouvez et commentez les résultats des instructions suivantes :*

```
> proc(x, y) begin x^2 + y^2 end_proc;
     . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
> sommecarres := %;
     . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
> sommecarres(3, 4);
                . . . . . . . .
> sommecarres(3, h);
                . . . . . . . .
```

▸ **Exercice 56.** *Donnez une procédure* `MEMBER` *prenant en argument un élément* `x` *et une liste* `L`*, testant l'appartenance de* `x` *à* `L`*.*

▸ **Exercice 57.** *Refaites l'exercice précédent en améliorant votre fonction* `MEMBER` *afin qu'elle détecte d'éventuelles erreurs de types d'arguments.*

▸ **Exercice 58.** *Définissez un alias pour votre fonction* `MEMBER`*.*

▸ **Exercice 59.** *Donnez une fonction prenant en argument une liste, et retournant la liste de tous les carrés des éléments de la liste d'entrée. La fonction utilisera une boucle* `for` *dans cette première version. Que pensez-vous de la complexité en temps de votre procédure ?*

▸ **Exercice 60.** *Refaites l'exercice précédent afin de donner une version s'exécutant en un temps linéaire en la taille de la liste d'entrée.*

▸ **Exercice 61.** *Comment utiliser la fonction* `map` *pour* `map`*er une fonction à plusieurs arguments ?*

▸ **Exercice 62.** *Donnez une fonction* `maximum` *calculant le maximum de ces arguments. La fonctions devra accepter un nombre variable d'arguments.*

▸ **Exercice 63.** *Donnez une version récursive efficace d'une fonction calculant le n-ème nombre de Fibonacci. Dans ce cas précis, le mot efficace signifie s'exécutant en temps linéaire par rapport à n.*

▸ **Exercice 64.** *Il est possible de donner un ordre explicite de sauvegarde du résultat d'un appel de fonction. Les lignes suivantes illustrent cette situation. Complétez l'exécution suivante :*

```
>> F := proc(n)
     begin F(n) := F(n-1) + F(n-2)
   end_proc:
>> F(0) := 0: F(1) := 1: op(op(F), 5);
              ..........................

>> F(4): op(op(F), 5);

              ..........................
```

▸ **Exercice 65.** *Donnez une fonction* `PGCD` *calculant le plus grand commun diviseur de deux entiers* `a` *et* `b` *passés en argument. Positionnez habilement la variable* `PRINTLEVEL` *pour suivre le déroulement du calcul* `PGCD(21, 15)`.

7 Exercices de programmation

▸ **Exercice 66.** *Écrivez une procédure* `NextPrime` *prenant en argument un entier* `N` *et retournant le plus petit nombre premier supérieur strict à* `N`. *Prévoyez un contrôle du type de l'argument.*

▸ **Exercice 67.** *Les nombres de Fermat sont les nombres de la forme $2^{a(n)}+1$ avec $a(n) = 2^n$. Pierre de Fermat les croyait à tord, premiers pour tout n. Mais quel est donc le plus petit $n \geq 0$, tel que $2^{a(n)} + 1$ ne soit pas premier ? On fera tout d'abord une procédure* `Fermat` *prenant en argument un entier* `n` *et retournant le nombre de Fermat correspondant. Puis, en utilisant cette procédure, on écrira la procédure de calcul de ce plus petit entier recherché.*

▸ **Exercice 68.** *Les nombres de Mersenne sont les nombres entiers de la forme $M_p = 2^p - 1$. Si M_p est premier, p est alors nécessairement premier. On ne connaît à ce jour que 33 nombres de Mersenne premiers dont le plus grand (pour l'instant) a été découvert en 1994 et correspond à $p = 859433$.*

1. *Mais en fait, jusqu'à quel p, pouvez-vous calculer en un temps "raisonnable" le nombre M_p sur votre machine ? Etes-vous loin de $p = 859433$?*
2. *On ne sait pas s'il existe ou non une infinité de nombres de Mersenne premiers. Dans ce cadre, regardez pour le p évalué à la question précédente le temps d'exécution de la commande* `isprime(p)`.

▸ **Exercice 69.** *Un entier est dit parfait s'il est égal à la somme de ses diviseurs (à l'exception de lui-même mais en comptant 1).*

1. *Écrivez une procédure* `Parfait` *prenant en argument un entier* `n`, *testant si cet entier est parfait. On retournera utilement la valeur* `TRUE` *ou* `FALSE`.
2. *Utilisez cette fonction pour écrire une fonction* `kParfait` *prenant en argument un entier* `k` *et retournant la liste des* `k` *premiers nombres parfaits.*
3. *Les nombres parfaits auraient-ils tendance à n'être que pairs ?*

▸ **Exercice 70.** *L'objet de cet exercice est de comprendre ce que nous apporte la phrase suivante : "MuPAD réalise une arithmétique exacte sur entier". Proposez une procédure calculant le i^{eme} terme d'une suite définie par récurrence. La suite devra converger vers une certaine valeur en calcul entier, et vers une toute autre valeur en calcul flottant.*