Chapitre M IV

Les procédures ou les fonctions (Version 5)

J.M. Janod

Définition d'une procédure ou fonction

Sous Maple les procédures sont implicitement des fonctions, c'est-à-dire qu'elle retourne automatiquement une valeur. Maple est donc un langage fonctionnel. Maple permet de définir ses propres procédures.

La syntaxe est la suivante:

<nom de la procédure>**:=proc(**<séquence des paramètres>**)**
global <séquence des variables globales>:
local <séquence des variables locales>:
options <séquence des options>:
description <chaine de caractères décrivant la fonction>:
<les instructions>:
end;

La valeur retournée par la procédure est par défaut l'évaluation de la dernière expression rencontrée. C'est la méthode recommandée pour donner une valeur à la fonction. On peut aussi fixer la valeur retournée en utilisant l'instruction **RETURN(**<valeur>**)** qui fixe la valeur de la fonction et permet de sortir de la fonction sans exécuter les instructions suivantes. Elle s'utilise donc avec un branchement conditionnel (If <condition de sortie> then RETURN(<valeur>) pour traiter des cas particuliers.

Signalons simplement l'option **remember** qui permet de mémoriser dans une table les valeurs de la fonction, ce qui évite le processus de calcul pour des valeurs déjà utilisées. Elle s'utilise lorsque le nombre d'instructions d' un algorithmique (récursif) croît rapidement en fonction d'un paramètre. (voir aussi fonction récursive).

La **description** est simplement une chaîne de caractères correspondant à un commentaire.

Avant de préciser les différents point, examinons les exemples suivants:

Définissons la procédure fonctionnelle max2 à deux paramètres (attention max existe) qui calcule le maximum de 2 nombres, hello une procédure sans paramètre, et logfac qui calcule le logarithme de factoriel n. On remarquera l'utilisation de

eval(<nom de la procédure>**);**
qui affiche sa définition

```
> max2:=proc(x,y)
    local t;
    description "calcule le max de deux numériques";
    if x>y then t:=x; else t:=y; fi;
    end;
```

```
> eval(max2);
```

```
> max2(3,5);
```

```
> max2(10,2)/3;# retourne un entier
```

```
> h:=evalf(rand(1..99)/100):x:=h();y:=h();max2(x,y);#un réel
```

```
> hello:=proc()
      print("bonjour");# la procédure retourne NULL car print
     end;
```

```
> z;hello();%;
```

```
> hello2:=proc()
      "BONJOUR";# elle retourne "BONJOUR"
      end;
```

```
> hello2();%;
```

```
> hellobis:=hello2;hellobis();
```

Dans la fonction logfac ci-dessous, l'option remember a été introduite dans un but purement pédagogique, hors de son contexte d'utilisation. On notera que la variable globale initialisée à 0 permet de compter le nombre de fois où les instructions de la fonction est exécutée.

```
> nbr:=0;logfac:=proc(n)
        local i,s;
        global nbr;
        options remember;
        description "calcule ln(n!)";
        if n=0 or n=1 then RETURN(0) fi;
        s:=0;
        nbr:=nbr+1;
```

```
            for i from 1 to n do s:=s+ln(i); od;
  end;
```

```
> logfac(0);logfac(1);
```

```
> for i from 2 to 5 do logfac(i);od;# on peut rajouter un evalf
```

L'affichage et les procédures.

Le résultat retourné par la procédure s'affiche si l'appel est suivi du point virgule et que printlevel est non négatif.

Les évaluations des instructions sont affichées selon la valeur de printlevel. Le niveau d'affichage 5=0+5 affiche l'appel et le retour de la procédure, le niveau 6=1+5 affiche les évaluations des instructions imbriquées au niveau 1 du corps des instructions de la procédure, 7=2+5 les niveaux des instructions imbriquées de niveau 2 etc. Plus généralement le niveau 5n=0+5n affiche les appels des procédures imbriquées de niveau n, 1+5n les évaluations du niveau 1 des procédures imbriquées de niveau n et ainsi de suite. L'utilisation du printlevel permet la recherche d'éventuelles erreurs.

```
> printlevel:=-1:max2(10,8);
```

```
> printlevel:=5:max2(12,15);
{--> enter max2, args = 12, 15
<-- exit max2 (now at top level) = 15}
```

```
> printlevel:=6:max2(8,3);printlevel:=1;
{--> enter max2, args = 8, 3

<-- exit max2 (now at top level) = 8}
```

Pour l'affichage des procédures prédéfinies, il faut utiliser la variable d'interface verboseproc en sachant que:

interface(verboseproc=1)

valeur par défaut print et eval affichent le squelette des procédures prédéfinies

interface(verboseproc=2)

print et eval affichent la définition compléte de la procédure prédéfinie.

```
> print(rand);
```

```
> interface(verboseproc=2);print(rand);# à essayer la réponse
  est très longue et a été supprimée
```

```
> interface(verboseproc=1);
```

Déclaration des variables locales et globales.

Les variables locales correspondent exactement au principe des variables locales introduites en algorithmique classique. Elles sont créées lors de l'appel et détruites à la sortie de la procédure.

Les variables globales ne doivent pas se substituer aux paramètres conformément aux principes algorithmiques; Leur emploi doit donc être réservé à des cas bien spéciaux.

Les déclarations des variables locales ou globales sont malheureusement optionnelles dans Maple. Il n'en demeure pas moins que la déclaration des variables locales doit être effectuée car l'évaluation dépend du statut des variables (cf paragraphe ci-dessous). En cas d'oubli, Maple choisira en fonction de ses propres critères, par exemple il considère automatiquement que toute variable non déclarée qui reçoit une affectation est locale, de même pour un indice de boucle.

```
> variableLG:=proc()
            u:=0;
            for i from 1 to fin do u:=u+i*v;od;
            end;
Warning, `u` is implicitly declared local
Warning, `i` is implicitly declared local
```

```
> i:=5;fin:=2;v:=2;variableLG();
```

```
> 'i'=i,'fin'=fin,'v'=v;#valeurs après l'appel
```

L'évaluation des variables locales

Les variables globales sont évaluées selon la règle de l'évaluation finale (si elle s'applique, nous verrons des variables qui sont régies autrement).Les paramètres effectifs suivent aussi l'évaluation finale (sauf si on met les quotes ').

Le principe de l'évaluation finale ne s'applique pas aux variables locales. Chaque variable locale est évaluée par l'expression de sa dernière affectation, les variables locales pouvant apparaître dans cette expression ne seront pas évaluées. on dit que c'est une évaluation au premier niveau (one level); Notons que les variables globales de l'expression ont une évaluation finale. Les paramètres formels sont évalués au premier niveau

```
> x:='x':y:='y':z:='z':x:=y;y:=z;z:=1;x,y,z;
```

```
> evaln1:=proc()
      global x,y,z;
      local x1,y1,z1,t1:
      y1:='x'+1;t1:=x+y;x1:=y1;y1:=z1;z1:=t1+1;
      z:=2;
print('t1'=t1,'x1'=x1,'y1'=y1,'z1'=z1,"y1 niveau2"=eval(y1,2)
);
      print('x1'=eval(x1),'y1'=eval(y1));
      assigned(x1),assigned(y1);
 end;
```

```
> evaln1();x,y,z;
```

La fonction op et les procédures

Maple considère qu'une procédure a un seul opérande, cet opérande étant sa définition. Le nombre d'opérande est donc égale à 1. op(<nom de la procédure>) affiche le même résultat que eval (<nom de la procédure>).

```
> nops(logfac);
```

```
> op(logfac);
```

```
> eval(logfac);
```

On peut aussi chercher les opérandes de l'opérande d'une procédure (les opérandes de op(<nom de la procédure>). Ils sont au nombre de six et correspondent aux éléments suivants:

- **op(1,op(**<nom de la procédure>**))** donne la liste des paramètres.
- **op(2,op(**<nom de la procédure>**))** donne la liste des variables locales.
- **op(3,op(**<nom de la procédure>**))** donne la liste des options.
- **op(4,op(**<nom de la procédure>**))** donne la table remember.

- **op(5,op(**<nom de la procédure>**))** donne la chaîne de description.
- **op(6,op(**<nom de la procédure>**))** donne la liste des variables globales.

```
> nops(op(logfac));
```

```
> for i from 1 to 6 do op(i,op(logfac));od;
```

Le passage des paramètres

Le passage des paramètres peut s'effectuer soit par valeur soit par adresse (i.e. par variable ou référence) comme dans de nombreux langages ou logiciel. Le choix est précisé dans la définition de la procédure, conformément à l'algorithmique.

Passage par valeur

le passage par valeur est l'option par défaut. En Maple, l'implantation de ce type de passage interdit toute nouvelle affectation d'une variable paramètre passée par valeur. On ne peut donc pas modifier la valeur d'un paramètre passé par valeur, si l'algorithme l'exige il faut introduire des variables locales. L'évaluation des paramètres effectifs suit la règle de l'évaluation finale.

```
> somchiffre:=proc(n)# erreur à l'exécution
        local s;
        s:=0;
        while n>0 do
        s:=s+irem(n,10);
        n:=iquo(n,10);
        od;
        s;
  end;
```

```
> somchiffre(11);
Error, (in somchiffre) illegal use of a formal parameter
```

```
> somchiffre:=proc(n) #utilisation d'une variable locale
        description "calcul la somme des chiffre d'un
  nombre";
        local s,n1;
```

```
          n1:=n;
          s:=0;
          while n1>0 do
          s:=s+irem(n1,10);
          n1:=iquo(n1,10);
          od;
          s;
end;
```

```
> ni:=nj;nj:=11;somchiffre(ni);#evaluation finale de ni
```

Vérification du type des paramètres

Maple peut vérifier lui-même le type des paramètres effectifs passés par valeur si les types des paramètres formels (passés par valeur) ont été déclarés. D'autre part on peut aussi inclure une vérification du type dans la partie instruction en utilisant les fonctions type et ERROR.

Vérification du type par programmation

Cette vérification permet de personnaliser le message d'erreur et s'applique à tous les paramètres. On ne précise plus le type dans les paramètres formels, mais on introduit une instruction de la forme suivante dans les instructions.

if not(type(<paramètre>**,**<son type>**) then ERROR(**<"message d'erreur">**): fi;**

La fonction ERROR affiche le message d'erreur et interrompt l'exécution.

```
> somchiffre2:=proc(n)
      local s,n1;
        if not(type(n,integer)) then
          ERROR("le paramètre doit être un entier");fi;
        n1:=n;
        s:=0;
        while n1>0 do
        s:=s+irem(n1,10);
        n1:=iquo(n1,10);
        od;
        s;
end;
```

```
> somchiffre2(3.12);
Error, (in somchiffre2) le paramètre doit être un entier
```

Déclaration des types pour les paramètres passés par valeur

On peut préciser explicitement le type des paramètres dans la liste des paramètres formels passés par valeur a l'aide de :: ainsi.

proc(n::integer,l::list)

Lors de l'appel si le type du paramètre effectif ne correspond pas, une erreur est signalée.

```
> somchiffre1:=proc(n::nonnegint)#déclaration du type
      local s,n1;
        n1:=n;
        s:=0;
        while n1>0 do
        s:=s+irem(n1,10);
        n1:=iquo(n1,10);
        od;
        s;
  end;
```

```
> somchiffre1(3.1);
Error, somchiffre1 expects its 1st argument, n, to be of type
nonnegint, but received 3.1
```

Passage par variable

Le passage par variable dit encore par adresse ou référence permet de modifier les paramètres effectifs de l'appel. Ce passage permet l'implantation de la notion algorithmique de procédure même si la dernière évaluation est retournée comme valeur.

En Maple, un paramètre formel est supposé être passé par variable si on précise le type name (entrée ou entrée-sortie).

proc(<parametre formel>**::name,..)**

Lors de l'appel, les paramètres effectifs seront passés par variable si on utilise les quotes ' qui évitent l'évaluation donc le passage par valeur, ou bien si la variable est non assignée.

```
> sompro:=proc(x,y,s::name,p::name)
      description "calcule la somme et le produit de 2
  nombres";
      p:=x*y;
      s:=x+y;
      NULL;
      end;
```

```
> sompro(2,5,s1,p1);s1,p1;
Error, sompro expects its 3rd argument, s, to be of type name, but
received 7
```

```
> sompro(2,5,s1,p1);
Error, sompro expects its 3rd argument, s, to be of type name, but
received 7
> sompro(2,5,'s1','p1');s1,p1;
```

Procédure avec un nombre variable de paramètres

Dans les procédures, les paramètres sont dans une séquence de nom args. Le premier paramètre est donné par args[1], le i-ème par args[i]. On dispose de la variable nargs qui contient le nombre de paramètres de la procédure. Pour définir une procédure avec un nombre variable d'arguments on utilise la forme suivante:

<nom>:=proc()
....
<instructions utilisant **args[i]** et **nargs**>
end;

Donnons un exemple d'une procédure calculant le maximum d'un nombre quelconque de données numériques.

```
> max3:=proc()
    local i,m:
    m:=args[1]; #initialisation du maximum
    for i from 2 to nargs do
        if args[i]>m then m:=args[i]:fi;
     od;
     m;# pour retourner la valeur
    end;
> max3(1,7,9,8,6);
```

Définition d'un opérateur

Sous Maple on peut définir un opérateur binaire à partir d'une procédure, qui s'utilisera comme les opérateurs prédéfinis du langage. Pour définir un opérateur il suffit de donner comme nom de la procédure le caractère, souhaité pour l'opérateur, précédé du signe spécial & dans une chaîne de caractères.

`&**<symboles>**`:=proc(..)
...
end;

```
> `&//`:=proc(a,b::integer)
    iquo(a,b);
    end;
```

```
> 30 &// 3;
```

Les fonctions récursives

Maple accepte les fonctions récursives. Elles sont conformes aux principes algorithmiques de la récursivité, tout en respectant les principes généraux des procédures en Maple.

Pour éviter que les appels récursifs calculent plusieurs fois les mêmes valeurs, l'utilisation de l'option remember est très utile.

Donnons deux exemples, la somme des chiffres d'un nombre par récursivité et la suite de Fibonacci (mathématicien italien plus connu sous le nom de Léonardo de Pise évêque de Pise, qui a modélisé la reproduction des lapins par cette suite, lors du siège de Pise au 13ième siècle).

Fibo(0)=Fibo(1)=1

avec

Fibo(n)=Fibo(n-1)+Fibo(n-2).

```
> somchifR:=proc(n::integer)
        description "version récursive de somchiffre";
        if n<10 then RETURN(n);
             else
       (somchifR(n &// 10)+irem(n,10));
       fi;
       end;
```

```
> somchifR(123);
```

Pour la suite de Fibonacci, le calcul de Fibo(n-2) étant nécessaire pour Fibo(n-1), on utilisera l'option remember.

```
> Fibo:=proc(n::integer)
     option remember;
     description "version récursive de Fibonacci";
     if n<=1 then RETURN(1)
            else RETURN(Fibo(n-1)+Fibo(n-2));
     fi;
     end;
```

```
> Fibo(1);Fibo(25);Fibo(50);
```