Chapitre M III

La Programmation(version5)

J.M. Janod

Les entrées et les sorties

Les commandes d'entrée sortie sont des instructions permettant de communiquer des données entre l'Unité centrale et les périphériques tels que le clavier pour les entrées, ou l'écran pour les sorties, et plus généralement avec les supports de masse (disque dur,disquette,..). Les informaticiens ont développé les instructions d'entrée sortie à partir d'un modèle de base qui s'adapte selon le périphérique utilisé.

Pour les entrées, l'information issue du clavier, ou d'un fichier de données, est dit un flux d'entrée. Ce flux peut être structuré en lignes de texte. A l'aide du clavier les lignes de texte sont délimitées par la frappe de <enter> ou <Maj enter>.

Pour les sorties, l'information envoyée par l'UC vers l'écran ou un fichier est dit un flux de sortie. Il peut être structuré là encore en lignes de texte. Pour l'écran les lignes de texte sont délimitées par un saut de ligne.

Dans ce chapitre nous n'aborderons pas les flux dirigés ou issus des fichiers implantés sur les supports de masse. Par défaut Maple considère que le clavier génère les flux d'entrée, et que l'écran reçoit les flux de sortie.

Les fonctions d'entrée

Les fonctions d'entrée correspondent à l'instruction algorithmique lire(x). Sous Maple, on peut utiliser deux commandes, readstat() pour lire une commande Maple au clavier, readline() pour lire une ligne entière au clavier.

La fonction readstat()

Elle est définie comme une fonction lisant une commande à partir du flux d'entrée, par défaut le clavier. La syntaxe la plus simple est

readstat();

ou

readstat("message d'invite");

La traduction de l'instruction algorithmique lire (x) en Maple est:

x:=readstat();

ou

x:=readstat("message d'invite");

Lorsque cette fonction est exécutée, Maple affiche le message d'invite et attend que l'utilisateur tape une expression Maple correcte suivie d'un point virgule et <enter>, l'expression est retournée par la fonction dans x.

Si l'utilisateur tape <enter> et que l'expression est manifestement incomplète Maple renouvelle le message d'invite permettant la frappe de la suite sans autre commentaire.

Si l'expression est incorrecte, Maple signale l'erreur et réexécute readstat() avec le même message d'invite.

Si le message d'invite reste affiché, retenons que readstat est toujours actif et a diagnostiqué soit une erreur soit une expression incomplète par exemple l'absence du point virgule.

```
>    x:=readstat("taper une expression ");
taper une expression    a+
taper une expression    2;


>   x:=readstat("taper une expression ");
taper une expression    a++2
syntax error, `+` unexpected:
a++2
  ^
taper une expression    a+2;
```

Si l'utilisateur clique en désespoir de cause sur le bouton stop lors d'un readstat ("message"), la fonction readstat est interrompue, mais le message d'invite reste dans toutes les commandes suivantes. Pour revenir au signe `>` on peut soit quitter Maple soit utiliser la commande suivante

interface(prompt=">")

remarquer l'égalité.

Contrairement à l'aide en ligne, readstat n'accepte que la première ligne et considère la commande incomplète.

```
>   interface(prompt=">");
>   readstat("multiligne ");
multiligne    a+
           2;
multiligne    5;
```

La fonction readline

Elle lit une ligne entière dans le flux d'entrée actif et retourne une chaîne de caractères. Si aucun fichier n'a été précisé, le flux d'entrée est par défaut le clavier, si un doute existe on peut donner le paramètre terminal dans la fonction.

Sa syntaxe est donc

readline()

ou

readline(terminal)

On l'utilise sous la forme

nom_variable:=readline()

ou

nom_variable:=readline(terminal)

Si l'utilisateur tape plusieurs lignes avec <Maj enter> après le premier appel de cette fonction la première ligne est retournée, La deuxième ligne sera lue automatiquement par le prochain appel de la fonction et ainsi de suite jusqu'à la dernière ligne, sans que l'utilisateur intervienne. Si on tape un point virgule il est retourné dans la chaîne.

```
> x:=readline();
> a+1
  b;

> y:=readline();# y contient le point virgule
```

L'affichage des réponses de Maple: mode et niveau

Une instruction suivie d'un point virgule entraîne une écriture à l'écran des résultats, cette écriture dépend du niveau d'affichage choisi, et du mode d'affichage. Nous verrons qu'il n'y a aucun affichage dans les procédures.

Mode d'affichage: prettyprint

La variable d'environnement prettyprint est fixée par défaut à 2 et correspond à un affichage en deux dimensions en mode graphique (**Typeset Notation),** c'est le mode le plus agréable et conforme à l'écriture des mathématiciens. On peut lui donner la valeur zéro, le mode d'affichage est alors sur une ligne (**Lineprint Notation**), la valeur 1 entraîne l'affichage en deux dimensions en mode caractère (**Character Notation**). On utilise la fonction interface pour modifier cette variable ou bien le menu **option/output display.**

La syntaxe de la fonction interface est:

interface(name=valeur)

Remarquer l'égalité

```
> Int(2^x,x=a..b);#par défaut prettyprint=2

> interface(prettyprint=0);Int(2^x,x=a..b);
Int(2^x,x = a .. b)
> interface(prettyprint=1);Int(2^x,x=a..b);
                                          b
                                         /
                                        |    x
                                        |   2  dx
                                        |
                                       /
                                          a
> interface(prettyprint=2);
```

Le Niveau d'affichage: printlevel

La variable printlevel permet de fixer le niveau d'information donnée dans les réponses de Maple. Par défaut la valeur est 1, la valeur zéro ne donne aucune information pour certaines commandes par exemple les boucles, des valeurs supérieures fournissent d'autres informations dans certaines commandes ou dans une procédure.

Une valeur négative annule tous les affichages résultant d'un point virgule.

La syntaxe est

printlevel:=<entier>

Les exemples suivants utilisent la boucle "pour" qu'on introduira dans un autre paragraphe.

```
> printlevel:=0;for i from 1 to 6 do i! od;x:=1;# les
  valeurs de i! ne s'affichent pas
```

```
> printlevel:=-1;Int(x^2,x=a..b);x:=1;# pas d'affichage
```

```
> printlevel:=1;for i from 1 to 6 do i! od;# les valeurs
  de i! s'affichent
```

Les fonctions de sortie

Les fonctions print(), lprint()

La fonction print(exp1,exp2,..) permet d'afficher la séquence des valeurs en respectant le mode courant, elle force l'affichage si le niveau d'affichage est négatif.

Sa syntaxe est

print(exp1,exp2,...expn)

cette fonction retourne la valeur NULL, donc évite un double affichage avec le point virgule et n'affecte pas les commandes %, %%, %%%.

La fonction lprint(exp1,exp2,..), avec l pour lineprint, est équivalente à print(exp1,exp2,..) mais le mode d'affichage est alors en ligne. Elle retourne la valeur NULL.

```
> x:=1;u:='x'+v;
```

```
> print(x,u);'%';%%%;
```

```
> printlevel:=-1;'x';print(x,3*v);printlevel:=1;'x';
```

```
>   interface(prettyprint=1);print(Int('x^2','x'=a..b));
```

```
>   interface(prettyprint=2);lprint(Int('x^2','x'=a..b));
  print(Int('x^2','x'=a..b));
Int(x^2,x = a .. b)
```

Sortie avec format:printf()

La fonction printf(), avec f pour format, permet un affichage selon un format précis pour les valeurs numériques, les chaînes de caractères et les autres expressions Maple.

La syntaxe est

printf(formats,exp1,exp2,..expn)

où

formats est une chaîne de caractères contenant pour chaque expression la spécification de son écriture qui est de la forme:

"message%[option] [largeur][.précision][code]"

où :

- **%** indique la place du résultat.
- **option** peut être
  - + pour que les nombres positifs soient signés,
  - - pour une justification à gauche
  - 0 (zéro) pour que le nombre soit précédé de zéros.
- **largeur** fixe le nombre minimum de caractères utilisés.
- **precision** donne le nombre de chiffres après la virgule pour les numériques, ou le nombre maximum pour des chaînes de caractères.
- **code** précise le type de donnée par exemple
  - d pour décimal et entier.
  - e, E pour la notation scientifique des valeurs numériques.
  - f pour réel (Float).
  - c pour un caractère.
  - s pour chaîne de caractères.
  - a pour autre.

  On peut utiliser
  - \n pour saut d'une ligne
  - \\ pour écrire \
  - \' pour écrire '

```
>   x:=2:printf("x=%+04d \n \' chaine\'=%8.3s",12,"tatatat")
  ;%;
x=+012
```

```
' chaine'=     tat
```

```
>    printf("%a",Int('x^2','x'));
Int(x^2,x)
```

La fonction printf est semblable à la fonction du même nom en langage C.

L'affectation

Nous renvoyons le lecteur au chapitre 1 sur les variables Maple

Les branchements conditionnels

Le branchement conditionnel simple

L'instruction algorithmique **Si** <condition> **alors** <instructions> **finsi** a, en Maple, la syntaxe suivante:

if <condition> **then** <instruction> **fi**

Une des instructions peut, elle-même, être un if qui est dit imbriqué.

Remarque

Cette instruction avec un point virgule affiche les évaluations en fonction du niveau d'affichage:

- niveau 0 pas d'affichage
- niveau 1 affichage du premier if
- niveau 2 affichage des évaluations jusqu'au deuxième if imbriqué

etc.

Cette technique permet de rechercher des erreurs. Pour afficher les résultats il est plus clair d'utiliser la fonction print().

```
>    x:=2:
   if x>0 then "positif":
                  if x>1 then ">1" fi:
   fi;#printlevel est 1
```

```
>    printlevel:=2;
   if x>0 then "positif":
                  if x>1 then ">1" fi:
   fi;# affichage des deux niveaux
```

```
> printlevel:=0;
  if x>0 then "positif":
              if x>1 then ">1" fi:
  fi;#pas d'affichage
```

```
> printlevel:=1;
  if x>0 then print("positif"):
              if x>1 then print(">1") fi:
  fi;#utilisation de print()
```

Le branchement conditionnel double

L'instruction algorithmique **Si** <condition> **alors** <instructions> **sinon** <instructions> **finsi** a, en Maple, la syntaxe suivante:

if <conditions> **then** <instructions> **else** <instructions> **fi**

On peut imbriquer des branchements conditionnels. L'affichage fonctionne comme pour un branchement simple.

```
> x;if x>0 then print("valeur positive")
   else print("valeur négative") fi;
```

Le Si immédiat

Le Si immédiat est une fonction renvoyant une des deux valeurs possibles en fonction d'une condition. Elle est équivalente à un branchement double.

La syntaxe est

`if `(<condition>,<valeur si vrai>,<valeur si faux>)

```
> x:=-2;5+`if`(x>0,2*x,-2*x);
```

Le branchement conditionnel multiple

L'instruction algorithmique **selon cas faire**

<cas1> **:**<instructions1>

...
<cas n>:<instructions n>
autrecas: <instructions autre cas>
fincas

a pour syntaxe:

if <cas1> **then** <instructions1>
elif <cas2> **then** <instructions 2>
...
elif <casn> **then** <instructions n>
else <instructions autre cas>
fi

elif est la contraction de else if, un seul fi suffit.

Cette commande fonctionne du point de vue de l'affichage comme les précédentes.

```
>    printlevel:=-1:x:=readstat("taper un nombre positif");
   if x<10 then print(" un chiffre")
     elif x<100 then print("nombre à deux chiffres")
    elif x<1000 then print("nombre à 3 chiffres")
    else print(">999")
   fi:printlevel:=1:;
taper un nombre positif   4;
```

Les boucles

La boucle itérer

La traduction de la boucle algorithmique

itérer
<instructions>
sortir si <condition>
<instructions>
fin itérer

est en Maple:

do
<instructions>**;**
if <condition> **then break fi;**
<instructions>**;**
od;

Les points-virgules peuvent être remplacés par deux points. Si od est suivi par ; alors les évaluations, dans la boucle, seront affichées sous réserve que printlevel>=1, l'affichage est annulé si on a od:; Le niveau d'affichage fixe les sorties à écran des instructions imbriquées d'un niveau inférieur ou égal.

Dans l'exemple suivant qui calcule le premier entier i tel que la somme des carrés des premiers entiers soit supérieure à 15, le i entre else et fi s'affichera si printlevel>=2.

Remarque

Pour éviter des affichages intempestifs, on utilisera od:; et la fonction print() pour les affichages désirés (ou printlevel:= -1 et od;).

```
>    i:=1:s:=0:
   do
    s:=s+i^2:
    if s>15 then break else i fi;
    i:=i+1:
   od;# avec affichage de niveau 1
   'i'=i;
```

```
>    printlevel:=0:i:=1:s:=0:# sans affichage
   do
    s:=s+i^2:
    if s>15 then break fi;
    i:=i+1:
   od;
   'i'=i;
```

```
>   printlevel:=1:i:=1:s:=0:
  do
   s:=s+i^2:
   if s>15 then break fi;
   i:=i+1:
  od: #sans affichage
  `i=`,i;
```

La boucle tant que

La traduction de la boucle algorithmique

tant que <condition> **faire**
  <instructions>
fin faire

est en Maple

while <condition> **do**
<instructions>**;**
od;

Remarque
Les affichages fonctionnent comme pour la boucle itérer.

```
>   s:=0:i:=1:
  while (s<15) do i:=i+1;s:=s+i^2: od:
  'i'=i;
```

La boucle pour

Il existe sous maple deux boucles pour

La boucle pour classique

la traduction de la boucle algorithmique

pour <i> **allant de** <début> **à** <fin> **avec** <pas> **faire**
<instructions>
fin faire

est en Maple

for < i> **from** <début> **to** <fin> **[by** <pas>**] do**
<instructions>
od;

Le by est optionnel. Le compteur <i>, à la sortie, est supèrieur ou égale à <fin>+<pas>.Le compteur <i> doit être numérique et les valeurs <début>, <Fin>, <Pas> cohérentes, sinon la boucle n'est pas exécutée. Les valeurs de <début>, <fin>, <pas> ne doivent pas être modifiées par les instructions de la boucle. Toute modification de ces valeurs dans le corps de la boucle n'affecte pas le déroulement de la boucle. Si début n'est pas précisé, la valeur 1 est prise par défaut.

L'affichage fonctionne comme pour la boucle itérer.

```
>   i:=7;for i from 1 to 4 do print(i!) od:
  'i'=i;
```

```
>   i:=7;for i from 4 to 1 by -1 do print(i!) od:
  'i'=i;
```

```
>   i:=7;for i from 1 to -2 do print(i!) od:
  'i'=i;
```

```
>   i:=7;fin:=4;
  for i  to fin do print(i):fin:=fin-1: od:
  'i'=i;'fin'=fin;
```

La boucle pour avec liste ou ensemble

Dans cette boucle le compteur prend toutes les valeurs d'une liste ou d'un ensemble. Sa syntaxe est:

for <i> **in** <liste|ensemble> **do** <instructions> **od;**

Le compteur <i>, <liste|ensemble> ne doivent pas être modifiés par les instructions de la boucle, car c'est sans effet et on perd la compréhension de la boucle.

L'affichage fonctionne comme pour la boucle itérer.

```
>   l:=[seq(i^2,i=1..4)];
```

```
>    printlevel:=-1:for i in l do print(i):i:=i+3:l:=subsop
   (nops(l)=NULL,l):od;printlevel:=1;l;'i'=i;
```

La boucle répéter jusqu'a

La boucle répéter jusqu'a n'existe pas en Maple mais peut s'écrire avec la boucle itérer ainsi

do
<instructions>**:**
if <condition> **then break fi:**
od;

Itération suivant

Dans toutes les boucles on peut introduire la commande next sous la forme

if <condition> **then next :if;**

Si la condition est vraie alors on passe à l'itération suivante, le compteur s'il existe est incrémenté.

L'exemple suivant calcule la somme des notes comprises entre 0 et 20 et pour sortir il suffit de taper -1

```
>    s:=0:i:=0:
   do
   x:=readstat("note suivante; (ou -1)?"):
   if (x=-1) then break fi:
   if (x<0) or (x>20) then next fi:
   s:=s+x:
   i:=i+1:
   od:
   's'=s;
note suivante; (ou -1)?   12;
note suivante; (ou -1)?   21;
note suivante; (ou -1)?   8;
note suivante; (ou -1)?   -1;
```

La syntaxe des boucles en Maple

La programmation des boucles doit toujours s'effectuer selon les principes des boucles algorithmiques, l'implantation de l'algorithmique dans un langage donné tient compte de ses particularités ou facilités de sa syntaxe, sans violation des principes élémentaires d'algorithmique.

La syntaxe des boucles Maple est très souple et est définie dans l'aide en ligne par:

```
|for <name>| |from <début>| |by <pas>| |to <fin>| |while <condition>|
                    do <statement sequence> od;
```

ou

```
   |for <name>| |in <expr>| |while <expr>|
                    do <statement sequence> od;
```

Les barres verticales indiquent des options facultatives. La seule partie obligatoire est

do <statement sequence> od;

Il faut alors utiliser **un break** dans la séquence des instructions.

On utilise for <name> si un compteur est nécessaire, qui est initialisé à 1, par défaut, ou par from <début> (optionnel). to <fin> est optionnel par suite il faut utiliser une condition de sortie soit avec un break soit avec le while <condition>.

On utilise le while si une condition est nécessaire. Lorsqu'elle cohabite avec un compteur la boucle s'effectue tant que le compteur est inférieur ou égal à <fin> et que la condition est vraie.

Remarque: Pour les problèmes d'affichage dans les boucles soit on utilise od: et le print dans la boucle pour l'affichage que l'on souhaite, soit od; et on a l'affichage automatique au niveau 1. On peut aussi mettre un point virgule systématiquement à la fin de chaque instruction du corps de la boucle.