Quelques mots sur les variables globales

Rémi Peyre

4 mars 2015

1 Qu'est-ce qu'une variable globale ?

1.1 Introduction par l'exemple

Comparons les résultats des deux programmes suivants, en langage MATLAB :

Programme `A.m`

```
function A
x = 0;
fprintf ('Initialisation de ''x'' dans la fonction principale, ');
fprintf ('à la valeur %d.\n', x);
auxiliaire;
fprintf ('Retour dans la fonction principale : ''x'' vaut %d.\n', x);
return
end

function auxiliaire
x = 1;
fprintf ('Passage dans la fonction auxiliaire : ''x'' vaut %d.\n', x);
return
end
```

Programme `B.m`

```
function B
global x
x = 0;
fprintf ('Initialisation de ''x'' dans la fonction principale, ');
fprintf ('à la valeur %d.\n', x);
auxiliaire;
fprintf ('Retour dans la fonction principale : ''x'' vaut %d.\n', x);
return
end

function auxiliaire
global x
x = 1;
fprintf ('Passage dans la fonction auxiliaire : ''x'' vaut %d.\n', x);
return
end
```

À l'exécution :

```
>> A
Initialisation de 'x' dans la fonction principale, à la valeur 0.
Passage dans la fonction auxiliaire : 'x' vaut 1.
Retour dans la fonction principale : 'x' vaut 0.

>> B
Initialisation de 'x' dans la fonction principale, à la valeur 0.
Passage dans la fonction auxiliaire : 'x' vaut 1.
Retour dans la fonction principale : 'x' vaut 1.
```

Que constatons-nous ? Dans le programme A, le fait de modifier la valeur de `x` dans la fonction auxiliaire n'a eu aucun effet sur la valeur de `x` dans la fonction principale ; alors que dans le programme B, la valeur de `x` dans la fonction principale a été modifiée par la fonction auxiliaire ! Ce sont bien sûr les instructions « `global x` » qui sont responsables de ce phénomène... En fait, comme nous allons le voir, dans le programme A la notation « `x` » renvoie à deux entités *différentes* selon qu'on est dans la fonction principale ou auxiliaire, alors que dans le programme B, c'est le *même* objet... !

1.2 Concepts fondamentaux

Variables et identifiants En informatique, quand on parle de « la variable `x` », on commet en réalité un léger abus de langage : en fait, on parle d'une certaine variable, stockée dans la mémoire de l'ordinateur, qui est *désignée* par « `x` ». On dit que « `x` » est l'*identifiant* de la variable en question.

Portée Ce qui est intéressant, c'est que deux variables *différentes* peuvent, dans certains cas, recevoir le *même* identifiant ! On appelle *portée* d'une variable `tartempion` la partie du code dans laquelle l'identifiant « `tartempion` » sera utilisé pour désigner la même variable. Bien entendu, les portées de deux variables différentes ayant le même identifiant ne peuvent donc pas se recouvrir, puisqu'un identifiant « `tartempion` » ne saurait désigner deux variables *en même temps*...

Localité, globalité On dit, demanière générale, qu'une variable est *locale* quand sa portée est "petite", et *globale* quand sa portée est "grande". Selon les langages de programmation, il peut y avoir divers degrés de localité et de globalité plus ou moins fins [∗] ; toutefois en première approximation, on retiendra la dichotomie suivante :

- Une variable *locale* est une variable dont la portée est limitée à la fonction dans laquelle elle est définie ;
- Une variable *globale* est une variable dont la portée est susceptible de s'étendre dans tous les fichiers du programme.

Remarque. La notion de variable globale (ou locale) est partagée par la quasi-totalité des langages de programmation. Si les détails de leur utilisation (statut par défaut, nom des instructions, portée) peuvent dépendre du langage, les concepts en revanche sont tout-à-faits transversaux, et indispensables à assimiler pour quiconque sera amené à écrire des programmes informatiques... !

[∗]. Dans certains langages de programmation, on peut par exemple rencontrer des variables « locales à un bloc d'instructions » (encore plus locales qu'une variable locale "normale"), ou « globales à un fichier » (partagées par toutes les fonctions d'un même fichier, mais ne s'étendant pas en-dehors du fichier), etc.

2 Variables locales et globales dans le langage MATLAB

2.1 Principes généraux

Dans le langage MATLAB, par défaut, une variable est considérée comme *locale*. Pour traiter un identifiant « `tartempion` » comme une variable globale, il faut l'indiquer explicitement par l'instruction :

```
global tartempion
```

Remarque. On peut définir en une seule instruction les variables `tartempion1`, `tartempion2` et `tartempion3` comme globales, par « `global tartempion1 tartempion2 tartempion3` ».

Remarque. À noter que pour MATLAB, la console [†] se comporte comme un fichier de fonction : par défaut, les variables de la console sont ainsi locales à la console, mais on peut les choisir globales grâce à l'instruction `global`.

Remarque. Dans le langage MATLAB, lorsqu'on veut utiliser une variable globale, il est *obligatoire* que l'instruction « `global tartempion` » vienne *avant* l'utilisation de la variable `tartempion` ! Par conséquent, en général on déclare les variables globales au début du code des fonctions.

Remarque. Attention ! Si la variable `tartempion` a été définie comme globale dans une *autre* fonction, et qu'on code une fonction où on ne précise *pas* « `global tartempion` », alors dans notre fonction l'identifiant `tartempion` se réfèrera à une variable *différente* de la variable globale : il s'agira là d'une variable *locale* à la fonction ! Par exemple, dans le programme `B.m` ci-dessus, si nous avions omis *une seule* des deux déclarations « `global x` », l'exécution aurait donné le même résultat que pour le programme `A.m`.

Remarque. Les variables globales permettent ainsi de partager une même variable `tartempion` entre plusieurs fonctions. Par contre, il est impossible d'utiliser l'identifiant « `tartempion` » pour désigner, d'une part une variable qui serait commune aux fonctions `A` et `B` [‡], et d'autre part une *autre* variable qui serait commune à d'autres fonctions `C` et `D`... Le même identifiant peut désigner une variable globale et une ou plusieurs plusieurs variables locales, mais jamais plusieurs variables globales !

Remarque. Dans les fichiers `*.m`, l'éditeur de MATLAB colore les variables globales en couleur bleu-vert, afin qu'on les repère immédiatement [§].

2.2 Cas d'une fonction définie à l'intérieur d'une autre fonction

☛ *Cette sous-section est un peu plus technique ; mais vous pouvez la sauter sans dommage, dès lors que vous ne comptez pas définir de fonctions à l'intérieur d'autres fonctions.*

La portée d'une variable locale, comme nous l'avons dit, est l'ensemble de la fonction dans laquelle elle a été définie. Il arrive cependant qu'à l'intérieur de notre

[†]. Que MATLAB appelle « workspace »

[‡]. Sauf dans le cas bien particulier où l'une de ces deux fonctions serait définie à l'intérieur de l'autre, comme nous le verrons plus loin.

[§]. Comme nous le verrons plus loin, cette couleur est en fait utilisée dès que la variable est partagée entre plusieurs fonctions, qu'elle soit globale ou locale.

fonction (càd. avant que le « `end` » final de la fonction soit atteint), on définisse une *autre* fonction... *Dans ce cas, la portée de la variable de la fonction "externe" s'étend,* même pour une variable locale, *à la fonction "interne".*

Ce phénomène est illustré par la comparaison des deux programmes suivants :

Programme `C.m`

```
function C
x = 0;
fprintf ('''x'' est initialisée à %d.\n', x);
auxiliaire;
fprintf ('''x'' vaut maintenant %d.\n', x);
return
    function auxiliaire
        x = 1;
        fprintf ('Dans la fonction auxiliaire, ''x'' vaut %d.\n', x);
        return
    end
end
```

Programme `D.m`

```
function D
x = 0;
fprintf ('''x'' est initialisée à %d.\n', x);
auxiliaire;
fprintf ('''x'' vaut maintenant %d.\n', x);
return
end

function auxiliaire
x = 1;
fprintf ('Dans la fonction auxiliaire, ''x'' vaut %d.\n', x);
return
end
```

À l'exécution :

```
>> C
'x' est initialisée à 0.
Dans la fonction auxiliaire, 'x' vaut 1.
'x' vaut maintenant 1.

>> D
'x' est initialisée à 0.
Dans la fonction auxiliaire, 'x' vaut 1.
'x' vaut maintenant 0.
```

Remarque. Dans le fichier `C.m`, la variable `x`, bien que locale, a été colorée en bleu-vert par l'éditeur : c'est parce que MATLAB emploie cette couleur dès lors qu'une variable est partagée entre plusieurs fonctions, que ce soit parce qu'une fonction a été définie à l'intérieur d'une autre ou parce que la variable est globale.

2.3 Cas des identifiants de fonctions

☛ *À nouveau, il s'agit d'une sous-section technique, que vous pouvez sauter sans dommage dès lors que vous ne définissez qu'une seule fonction par fichier* `*.m`.

Les fonctions ont *aussi* des identifiants... ! Par conséquent, de même que pour les variables, ces identifiants de fonctions ont chacun une certaine portée... Dans le langage MATLAB, cette portée s'étend comme suit :

— Lorsqu'une fonction est associée au nom d'un fichier `*.m` — auquel cas elle doit être la première fonction définie dans ce fichier —, sa portée est *globale*.
— Lorsqu'une fonction est définie hors de toute fonction, mais ne correspond pas au nom du fichier `*.m` où elle est définie (ce qui arrive lorsqu'un fichier `*.m` définit plusieurs fonctions), sa portée est *globale au fichier* `*.m`.
— Lorsqu'une fonction est définie à l'intérieur d'une autre fonction, sa portée est *locale à la fonction la plus interne dans laquelle elle a été définie*.

En fait, lorsque MATLAB rencontre un identifiant [¶], celui-ci va être interprété dans l'ordre de priorité suivant :

— On regarde d'abord si l'identifiant correspond au nom d'une variable ou d'une sous-fonction définie dans la fonction courante [|||]. Si on a trouvé, on utilise alors cette signification ;
— Si on n'a pas trouvé, on regarde si l'identifiant correspond au nom d'une fonction définie dans le fichier `*.m` courant (en-dehors de toute autre fonction). Si on a trouvé, on utilise alors cette signification ;
— Si on n'a toujours pas trouvé, on regarde si cela correspond au nom d'une fonction associée à un autre fichier `*.m` (au sein de notre répertoire de travail, s'entend) ;
— Si on n'a toujours pas trouvé, c'est une erreur... !

On voit ainsi que MATLAB peut utiliser le même identifiant à la fois pour des variables et pour des fonctions, tantôt localement et tantôt globalement, avec de nombreuses significations différentes !

3 Du bon usage des variables globales

3.1 Les variables globales, une arme à double tranchant

Les variables globales sont un outil à la fois puissant et dangereux :

— *Puissant*, car elles permettent de partager une valeur entre plusieurs fonctions sans avoir à la passer en argument [**]. En outre, elles correspondent à notre conception naïve de ce qu'est une variable informatique.
— *Dangereux*, parce qu'utiliser des variables globales va à l'encontre du caractère *modulaire* de la programmation fonctionnelle. Ainsi, si la variable globale `tartempion` utilisée par une fonction `truc` est susceptible d'être modifiée par d'autres fonctions, on ne pourra pas bien comprendre le fonctionnement de `truc` juste en regardant son seul code-source, et on risque de faire des fautes

[¶]. Notez qu'il n'y a pas de moyen de savoir à priori si un identifiant est associé à une variable ou à une fonction...

[|||]. Ces deux cas sont mutuellement exclusifs, MATLAB interdisant de définir une variable et une fonction ayant le même identifiant et dont les portées se recouvriraient.

[**]. D'où gain de temps aussi, car tout passage en argument implique une copie dans la mémoire.

de programmation si on oublie comment la variable `tartempion` est susceptible d'être modifiée par d'autres fonctions... Et si par malheur on a l'idée d'utiliser le même identifiant de variable globale pour deux usages différents, on court à la catastrophe ! Il faut donc veiller à ne pas abuser des variables globales.

Le bon usage des variables globales consiste par conséquent à ne les utiliser que quand il s'agit d'une valeur ayant une signification homogène dans tout le programme, réellement partagée par toutes les fonctions, et qui ne change qu'en des occasions bien précises.

Remarque. Ce conseil ne vous paraitra peut-être pas très important à notre niveau, mais il devient absolument crucial quand on utilise des programmes sophistiqués contenant des centaines de fonctions et de variables : si toutes les variables étaient globales, on ne s'y retrouverait plus... !

3.2 Usage des variables globales dans mon cours

Dans les exemple de codes que j'utilise dans mon cours, l'usage des variables globales que je fais correspond en fait à ce qu'on appelle des *macros* : à savoir, je donne un nom à une expression numérique pour manipuler une expression littérale plutôt que la valeur numérique elle-même, ce nom correspondant à une variable globale ; la variable globale en question est alors initialisée une unique fois au début du code, et plus jamais modifiée ensuite. *Cette technique est vivement encouragée,* car elle présente deux avantages considérables :

— Le code devient plus lisible : si quatorze paramètres numériques de notre programme valent tous 3, un identifiant comme « `NOMBRE_DE_DES` » sera bien plus parlant que « `3` » pour comprendre le sens du code !

— Le code devient aussi plus facile à modifier : car si vous souhaitez changer la valeur d'un paramètre, vous n'aurez qu'à remplacer l'*unique* endroit où ce paramètre aura été initialisé ! Alors qu'utiliser la valeur numérique du paramètre à plusieurs endroits différents présente un risque d'erreur terrible quand on souhaite changer celle-ci...

Remarque. Dans d'autres langages que MATLAB, il existe le concept de « macro » qui remplace avantageusement l'usage des variables globales exposé ci-dessus.

Remarque. De manière générale, on peut distinguer au moins deux autres usages judicieux des variables globales :

— Pour une quantité utilisée à travers tout le programme, et susceptible d'être modifiée en de nombreux endroits dans des fonctions différentes : par exemple, si on souhaite compter le nombre total de fois qu'on a dû faire appel à un tirage au sort dans l'ensemble de notre programme, une variable globale `nombre_total_de_tirages` sera tout indiquée...

— Pour un paramètre général du programme qui ne sera pas modifié, mais qu'on ne connait pas à l'avance, par exemple parce qu'il sera demandé à l'utilisateur en cours d'exécution...

4 Allocation statique et dynamique des variables

Parler de variables locales et globales est aussi l'occasion de parler du concept d'allocation *statique* et *dynamique* des variables informatiques.

Dans MATLAB (et dans la quasi-totalité des langages de programmation), à la fin de l'exécution d'un programme, MATLAB garde en mémoire l'existence des variables globales ainsi que leurs valeurs, de sorte que ces variables globales seront toujours accessibles, sans être réinitialisées, lors d'une nouvelle exécution : on dit que l'allocation des variables globales est *statique*. Par contre, par défaut sous MATLAB, lorsqu'une fonction se termine, les variables *locales* qui y étaient définies sont "oubliées" définitivement par le logiciel, comme si elles n'existaient plus, et devront être réinitialisées au prochain appel de la fonction : on dit que l'allocation des variables locales est *dynamique*[††].

Il peut être bon de savoir qu'on peut néanmoins imposer que certaines variables *locales* soient pourtant d'allocation *statique* : sous MATLAB, cela se fait par l'instruction `persistent`. Cette instruction relève toutefois d'un usage de programmation assez avancé, et je ne l'utiliserai pas dans mon cours.

[††]. Et plus précisément *automatique*, car MATLAB oublie ces variables *de lui-même*, sans qu'on ait à lui en donner l'instruction explicite.