Programmation & Logiciels Statistiques

Cours 5

Salim Lardjane - Université de Bretagne-Sud

Le macro-langage
Généralités

- Le macro-langage permet de **paramétrer** les programmes SAS et donc de simplifier certains programmes
- Il est particulièrement utile si l'on souhaite effectuer un même traitement sur **un grand nombre de tables SAS**
- Sans recours au macro-langage, la démarche est la suivante :

Le macro-langage

Généralités

1. Ecrire le programme SAS du traitement souhaité pour l'une des tables
2. Copier le programme SAS obtenu autant de fois qu'il reste de tables
3. Identifier les éléments qui diffèrent d'une table à l'autre (noms des tables, listes des variables, titres, options, instructions…)
4. Modifier les valeurs de ces éléments dans les portions de code correspondant aux différentes tables

Le macro-langage

Généralités

- Le macro-langage permet d'alléger ces manipulations et de réduire le risque d'erreur
- Avec le macro-langage, les traitements précédents peuvent être effectués de la façon suivante :

1. Ecrire le code SAS du traitement à effectuer pour l'une des tables
2. Identifier les éléments du code dont les valeurs pourront varier : il s'agira des **paramètres** du programme

Le macro-langage
Généralités

3. Déclarer les paramètres en début de programme en attribuant un **nom** à chacun d'eux et éventuellement une valeur par défaut
4. Retravailler le code SAS du traitement à effectuer en remplaçant les valeurs des paramètres par des **références** à ces derniers à l'aide de leurs noms

Le macro-langage
Généralités

- De cette façon, le programme SAS du traitement à effectuer est écrit une fois pour toute
- Chaque fois que l'utilisateur souhaitera exécuter ce programme, il lui suffira d'attribuer de nouvelles valeurs aux paramètres **sans toucher au code** du traitement à réaliser

Le macro-langage
Généralités

- Le macro-langage est également utile pour :

1. Lancer certains traitements et pas d'autres **en fonction** des valeurs affectées à un ou plusieurs paramètres
2. Récupérer certains éléments intervenant dans une étape **data** (nombre d'observations, valeurs de variables, cumuls,…) pour les utiliser dans une autre étape **data** ou une autre étape **proc**

Le macro-langage
Fondamentaux

- Dans le jargon SAS, les paramètres d'un programme sont appelés **macro-variables**
- Dans un programme paramétré, on fait référence à un paramètre de nom NOMV sous la forme **&**NOMV
- En plus des instructions du langage SAS de base, les programmes paramétrés peuvent faire intervenir des **macro-instructions**

Le macro-langage
Fondamentaux

- Les macro-instructions ont la particularité de commencer par le caractère **%**
- Pour gérer et manipuler les macro-variables, on utilise des fonctions spécifiques, appelées **macro-fonctions**
- Celles-ci commencent également par le caractère **%**

Le macro-langage
Fondamentaux

- Il existe **deux approches** pour réaliser le paramétrage d'un programme SAS à l'aide du macro-langage

1. **Paramétrage sans macro-programme** : on fait précéder le code SAS de la définition et de l'affectation de valeurs aux paramètres. Le traitement ne fait intervenir que des macro-variables

Le macro-langage
Fondamentaux

2. **Paramétrage avec macro-programme** : le code SAS est encapsulé, paramétré et stocké. Il est ensuite **appelé** après l'affectation de valeurs aux paramètres

- La **compilation** d'un programme SAS contenant du macro-langage se déroule en deux étapes :

Le macro-langage
Fondamentaux

1. Traitement du programme par le compilateur macro
2. Compilation usuelle

- **Aucun des deux compilateurs ne sait traiter les éléments de langage gérés par l'autre**
- Le compilateur macro génère un programme en langage SAS de base mais ne le compile pas

Le macro-langage
Macro-variables

- Une macro-variable est un objet du macro-langage doté d'un nom et d'une valeur
- Sa valeur peut être utilisée n'importe quand dans un programme
- On peut créer une macro-variable de plusieurs façons :

1. A l'aide d'une macro-instruction %let n'importe où dans un programme SAS

Le macro-langage
Macro-variables

2. L'instruction call symput permet, **dans une étape data**, de créer des macro-variables à partir d'une ou plusieurs variables SAS

- Le **nom** d'une macro-variable commence toujours par une lettre ou _
- La casse n'a pas d'importance : nomMV, NOMMV, nommv sont équivalents

Le macro-langage
Macro-variables

- La **longueur maximale autorisée** pour le **nom** d'une macro-variable est de 32 caractères depuis SAS 8
- Les **caractères autorisés** pour les **noms** de macro-variables sont : toutes les lettres, les chiffres et _

Le macro-langage
Macro-variables

- La **valeur** d'une macro-variable est toujours une **chaîne de caractères**
- Dans ce cas, la casse (majuscules, minuscules) est importante
- La longueur maximale autorisée pour le contenu (la valeur) d'une macro-variable est de 65 534 caractères (version 9 de SAS)
- Tous les caractères alphanumériques sont autorisés

Le macro-langage
Macro-variables

- L'utilisation de **certains** caractères (apostrophe, virgule, point-virgule, =) dans la valeur d'une macro-variable requiert toutefois un traitement à l'aide de **macro-fonctions de « quoting »**

Le macro-langage
Création de macro-variables

- L'utilisation de la macro-instruction %let est la façon la plus fréquente de créer une macro-variable. Par exemple,

```
%let nomprenom = Lardjane Salim;
%let annee = 2003;
%let libelle = "Duree de connexion";
%let date_achat = 10/11/2011;
%let total = 31 + 56;
```

Le macro-langage
Création de macro-variables

- Les espaces au début ou à la fin de la valeur ne sont pas pris en compte
- Quand les guillemets sont présents, ils font partie intégrante de la valeur et sont recopiés lors de son utilisation
- La valeur de total est **le texte** 31 + 56 et non pas le résultat de l'opération

Le macro-langage
Création de macro-variables

- L'instruction call symput permet de créer une ou plusieurs macro-variables à partir de variables SAS
- Elle s'utilise **uniquement** dans une étape **data**
- Par exemple,

```
libname base 'd:/data/sasdata/banque';
data _NULL_;
     set base.clients;
     call symput("nbclients",_n_);
run;
```

Le macro-langage
Création de macro-variables

- Le **premier argument** de call symput définit le **nom** de la macro-variable créée au cours de l'étape data
- Cette expression peut contenir :

1. Le nom de la macro-variable entre guillemets
2. Une formule de calcul de l'étape data, utilisant des chaînes de caractères entre guillemets, des variables SAS et des fonctions

Le macro-langage
Création de macro-variables

- Le **deuxième argument** donne la variable SAS dont on souhaite récupérer les valeurs
- A **chaque observation lue**, la valeur de cette variable est copiée dans la macro-variable voulue
- Il est possible de donner dans cet argument un calcul plutôt qu'un nom de variable

Le macro-langage
Création de macro-variables

- Dans l'exemple, on copie de manière répétée le numéro de l'observation lue dans la macro-variable
- Par conséquent, à la fin de l'étape data, la macro-variable nbclients a pris le numéro de la dernière observation lue

Le macro-langage
Création de macro-variables

- Mise dans une macro-variable, cette information peut être utilisée à un endroit ou la lecture de la table n'est pas autorisée
- Par exemple, dans un titre du type « Statistiques calculées sur *n* clients »

Le macro-langage
Création de macro-variables

- Un autre exemple :

```
proc sort data = base.clients;
     by nbenf;
run;
data _NULL_;
     set base.clients;
     call symput("nb_enfants_max",nbenf);
     call symput("nom_client",nom);
run;
```

Le macro-langage
Création de macro-variables

- Il est également possible de créer autant de macro-variables qu'il y a d'observations dans une table
- A cet effet, on s'assure que le premier argument du call symput change à chaque observation
- C'est fait en général à l'aide de l'instruction compress

Le macro-langage
Création de macro-variables

- Exemple :

```
data _NULL_;
      set base.clients;
      call symput(compress("nomCli"||_n_),
      nom);
run;
```

- Les macro-variables obtenues s'appelleront NOMCLI1, NOMCLI2, …, NOMCLI15, la table clients comptant 15 observations

Le macro-langage
Création de macro-variables

- L'instruction call symput peut également être utilisée d'un autre manière
- Par exemple,

Le macro-langage

Création de macro-variables

```
data test;
        input x1 $ x2;
cards;
a 1
b 2
c 1
d 2
f 1
;
run;
data test;
        set test;
        call symput(x1,x2);
run;
```

Le macro-langage
Création de macro-variables

- Dans ce cas, **on crée autant de macro-variables qu'il y a de modalités à la variable x1**
- Cette dernière variable doit être de type caractère et ses modalités doivent commencer par un caractère ou par _
- Dans l'exemple précédent, cinq macro-variables sont créées : &a, &b, &c, &d, &f

Le macro-langage
Création de macro-variables

- Les valeurs récupérées via un call symput sont dépourvues de format
- Pour appliquer un format à la valeur copiée dans la macro-variable, on utilise put dans le second argument. Par exemple,

```
data _NULL_;
     set base.clients;
     call symput(compress("annee"||_n_),
          put(dtnais,year4.));
run;
```

Le macro-langage
Création de macro-variables

- Les macro-variables crées avec l'instruction call symput ne sont disponibles et utilisables qu'après l'instruction run de l'étape **data**
- Si l'étape data s'arrête prématurément en raison d'erreurs, **aucune des macro-variables n'est créée**
- Pour **supprimer** une ou plusieurs macro-variables, on utilise l'instruction %symdel

Le macro-langage
Résolution des macro-variables

- Une macro-variable est utilisable **partout** dans un programme SAS
- La valeur affectée à une macro-variable est générée par le compilateur macro
- En jargon SAS, on dit que celui-ci effectue une **résolution** des macro-variables

Le macro-langage
Résolution des macro-variables

- Il est possible de **consulter** la valeur d'une macro-variable dans la vue SASHELP.VMACRO ou de **l'afficher** dans la fenêtre Log en utilsant l'instruction %put
- Il y a **six règles** de résolution des références aux macro-variables en lisant l'expression **de la gauche vers la droite**

Le macro-langage
Résolution des macro-variables

1. &nomMV est remplacé par la valeur de la macro-variable appelée *nommv*
2. &nomMV. est remplacé par la valeur de la macro-variable appelée *nommv*
3. Si un texte ne contient ni & ni % collé à un autre caractère, il est restitué tel quel par le compilateur macro

Le macro-langage
Résolution des macro-variables

4. && est remplacé par un seul & pour la lecture suivante de l'expression
5. Le compilateur macro fait autant de lectures que nécessaire pour éliminer tous les & de l'expression considérée
6. Toute expression placée entre deux guillemets simples (quotes) est considérée comme un texte à renvoyer tel quel

Le macro-langage
Résolution des macro-variables

- En cas de **référence erronée** à un nom de macro-variable *nommv*, on obtient dans la fenêtre Log le warning « Apparent symbolic reference *NOMMV* not resolved »
- Exemples de résolution :

```
%let nomBib = base;
%let nomTab = clients;
%let nomTab1 = ventes;
%let i = 1;
```

Le macro-langage
Résolution des macro-variables

- &nomTab → clients (règle 1)
- &nomTab. → clients (règle 2)
- &nomTab1 → ventes (règle 1)
- &nomTab.1 → clients1 (règle 2)
- &nomBib&nomTab → baseclients (2 fois de suite le règle 1 depuis la gauche)
- &nomBib.&nomTab → baseclients (règle 2 jusqu'au point puis règle 1)

Le macro-langage
Résolution des macro-variables

- &nomBib..&nomTab → base.clients (règle 2, règle 3 puis règle 1, depuis la gauche)
- &nomTab&i → clients1 (deux fois de suite la règle 1, depuis la gauche)
- &&nomTab&i → &nomTab1 (règle 4 puis règle 3, depuis la gauche, règle 1 pour &i) → ventes (règle 5, règle 1)

Le macro-langage
Affichage des macro-variables

- Pour afficher le **contenu** d'une macro-variable dans la fenêtre Log, on utilise %put
- Cette instruction permet également d'afficher un texte quelconque dans la fenêtre Log
- Exemple :

```
%put La table base.clients contient
     &nbclients observations;
%put La table sur laquelle on travaille est
     "&nomTab";
%put La table sur laquelle on travaille est
     '&nomTab';
```

Le macro-langage
Affichage des macro-variables

- La règle 1 est utilisée pour les deux premiers affichages
- Conformément à la règle 6, une référence à une macro-variable entre guillemets **simples** (quotes) n'est pas résolue

Le macro-langage
Affichage des macro-variables

- Pour afficher la **liste** de toutes les macro-variables, on peut utiliser %put _all_
- Pour afficher la **liste** de toutes les macro-variables définies par l'utilisateur, on peut utiliser %put _user_
- Pour afficher la **liste** de toutes les macro-variables définies automatiquement par SAS, on peut utiliser %put _automatic_

Le macro-langage
Affichage des macro-variables

- Une seconde méthode consiste à aller rechercher les macro-variables dans SASHELP.VMACRO
- Les macro-variables définies par l'utilisateur se repèrent par les valeurs de la variable scope, qui vaut GLOBAL
- Exemple :

Le macro-langage

Affichage des macro-variables

```
proc print data = sashelp.vmacro label
                  noobs;
     var name value;
     where scope = "GLOBAL";
run;
```

Le macro-langage
Macro-variables automatiques

- Les macro-variables **automatiques** sont créées et gérées par SAS
- L'utilisateur ne peut ni les modifier ni les détruire
- Les plus utiles sont les suivantes :
- **Sysdate** : date de démarrage de la session (ex. de valeur : 10NOV11)
- **Sysday** : jour de démarrage de la session (ex. de valeur : Saturday)

Le macro-langage
Macro-variables automatiques

- **Systime** : heure de démarrage de la session (ex. de valeur : 08:20)
- **Syslast** : nom de la dernière table créée (ex. de valeur : base.clients)
- **Sysuserid** : nom de l'utilisateur du système (ex. de valeur : propriétaire)
- **Sysscp** : nom de l'environnement utilisé (ex. de valeur : WIN)

Le macro-langage
Macro-variables automatiques

- **Sysscpl** : version de l'environnement utilisé (ex. de valeur : X64_VSHOME)
- **Sysver** : version du système SAS en cours d'utilisation (ex. de valeur : 9.2)
- **Syserr** : code retour des erreurs des procédures SAS (ex. de valeur : 0)
- Syserr est pratique pour arrêter l'exécution d'un programme à la première erreur grave rencontrée

Le macro-langage
Macro-variables automatiques

- Syserr prend la valeur 0 si l'étape s'est déroulée sans erreur
- Syserr prend la valeur 4 si l'étape a connu des erreurs fatales (messages warning dans la fenêtre Log)
- Syserr prend la valeur 5 ou supérieures selon le type d'erreurs fatales rencontrées (messages error dans la fenêtre Log)

Le macro-langage
Macro-variables automatiques

- Les codes d'erreur n'étant pas documentés au-delà de 5, in teste en général si la valeur de Syserr est supérieure à 4 ou non
- Exemple d'utilisation des variables automatiques :

Le macro-langage

Macro-variables automatiques

```
title1 "&sysday &sysdate";
title2 "Vous utilisez la version de SAS :
          &sysver";
title3 "Statistiques des ventes";
proc means data = base.ventes sum mean
          maxdec=2;
     var mt_moy nb_achats;
run;
```

Le macro-langage
Macro-fonctions

- Les **macro-fonctions** permettent d'agir sur les valeurs des macro-variables
- Le **nom** d'une macro-fonction commence toujours par **%**
- On distingue **trois** catégories de macro-fonctions :

Le macro-langage
Macro-fonctions

1. Celles qui prolongent les fonctions SAS
2. Celles qui permettent d'évaluer des expressions arithmétiques faisant intervenir des macro-variables
3. Celles qui servent au « quoting »

Le macro-langage
Macro-fonctions

- %sysfunc permet d'utiliser **une fonction SAS** sur la valeur d'une macro-variable
- Le plus souvent, il s'agit de fonctions SAS agissant sur les chaînes de caractères : compress, substr, upcase, notamment
- L'argument optionnel format. Permet de formater le résultat renvoyé par la fonction SAS

Le macro-langage
Macro-fonctions

- Exemple :

```
%let dateSyst = %sysfunc(today(),fradfwkx.);
%put Nous sommes le &dateSyst;
```

Le macro-langage
Macro-fonctions

- Pour effectuer des **opérations arithmétiques** (+,-,*,/) avec les valeurs des macro-variables, on utilise %sysevalf
- Exemple :

```
%put &total;
%put %sysevalf(&total);
%put %sysevalf((&total+1)/120);
```

Le macro-langage
Macro-fonctions

- Pour **masquer** des caractères pouvant poser problème dans la valeur d'une macro-variable (apostrophe, point-virgule, virgule) (« **quoting** »), on peut utiliser la macro-fonction %str
- Si le texte contient une apostrophe, on la fait précéder du signe %

Le macro-langage
Macro-fonctions

- Exemple :

```
%let resetTitres = %str(title; footnote;);
%put &resetTitres;
%put %str(Nombre d%'observations :
     &nbclients);
```

Le macro-langage
Paramétrage sans macro-programme

- On a vu qu'il existait deux approches pour paramétrer un programme SAS : avec ou sans macro-programme
- La première approche consiste à :

1. Créer un ou plusieurs paramètres en début de programme
2. Paramétrer le traitement à effectuer

Le macro-langage

Paramétrage sans macro-programme

- Exemple 1 :

```
%let table = base.clients;
%let stats = mean;
%let listeVar = nbenf;

proc means data = &table &stats;
     var &listeVar;
run;
```

Le macro-langage

Paramétrage sans macro-programme

- Exemple 2 :

```
%let table = base.intranet;
%let stats = mean sum;
%let listeVar = duree nbpages;

proc means data = &table &stats;
     var &listeVar;
run;
```

Le macro-langage
Macro-programmes

- Un macro-programme est une portion de code SAS encapsulée, paramétrée et stockées
- Son utilisation se fait en deux phases

1. **Définition** ou **compilation** : celle-ci s'accompagne d'un stockage
2. **Appel** : instruction permettant d'exécuter le code SAS stocké dans le macro-programme

Le macro-langage
Macro-programmes

- Un macro-programme possède en général des paramètres en **entrée**
- Ceux-ci sont des macro-variables, créées et gérées par SAS, dont les **valeurs** sont fournies lors de l'appel du macro-programme
- La macro-instruction %macro sert à **créer** un macro-programme
- La dernière line, %mend, sert à marquer la fin du macro-programme

Le macro-langage
Macro-programmes

- L'ensemble des instructions, de %macro à %mend, est appelé en jargon SAS, **code source** ou **source** du macro-programme
- Un macro-programme est appelé par **%** suivi de son **nom** puis, entre parenthèses, des valeurs à attribuer aux paramètres

Le macro-langage
Macro-programmes

- Exemple : soumettre le code source

```
%macro calcul(table, stats, listeVar);
      proc means data = &table &stats;
            var &listeVar;
      run;
%mend calcul;
```

- Appel :

```
%calcul(base.clients, mean, nbenf);
%calcul(base.intranet, mean sum, duree
nbpages);
```

Le macro-langage
Macro-programmes

- Le macro-programme créé s'appelle *calcul*. Les paramètres *table*, *stats* et *listeVar* sont créés au moment de leur déclaration mais ne prennent leurs valeurs que lors d'une instruction d'appel du macro-programme

Le macro-langage
Macro-programmes

- Les paramètres d'entrée d'un macro-programme peuvent être de deux types :

1. Paramètres **positionnels**
2. Paramètres **mots-clefs**

- Dans une instruction d'appel d'un macro-programme, les paramètres **positionnels** doivent apparaître **dans le même ordre** que lors de leur déclaration

Le macro-langage
Macro-programmes

- Exemple :

```
%macro periodeAchats(premierJ, dernierJ,
      stats);
      proc means data = base.ventes &stats;
            where "&premierJ"d <= date_der <=
                  "&dernierJ"d;
            var mt_der;
      run;
%mend periodeAchats;

%periodeAchats(01dec1998, 15jun1999, mean);
%periodeAchats(01jan2001, 31dec2001,);
```

Le macro-langage
Macro-programmes

- Les paramètres **mots-clefs** ont la particularité de pouvoir se voir assigner une valeur par défaut
- L'ordre de spécification des valeurs des paramètres mots-clefs lors de l'appel n'a pas d'importance
- Lors de l'appel, tout paramètre non cité prend sa valeur par défaut
- Il est nécessaire de connaître non plus l'ordre mais le **nom** des paramètres

Le macro-langage
Macro-programmes

- Exemple :

```
%macro periodeAchats2(premierJ= , dernierJ =
&sysdate, stats = mean);
     proc means data = base.ventes &stats;
          where "&premierJ"d <= date_der <=
               "&dernierJ"d;
          var mt_der;
     run;
%mend periodeAchats2;
```

Le macro-langage

Macro-programmes

```
%periodeAchats2(premierJ = 01dec1998,
     dernierJ = 15jun1999, stats = sum);
%periodeAchats2(stats = sum, premierJ =
     31dec2001);
```

Le macro-langage
Macro-programmes

- Il est possible d'utiliser **à la fois** des paramètres positionnels et des paramètres mots-clefs
- On parle alors de paramétrage **mixte**
- Dans ce cas, les paramètres positionnels doivent **tous** apparaître avant les paramètres mots-clefs lors de leur déclaration et lors de l'appel

Le macro-langage
Macro-programmes

- On a vu que le code-source d'un macro-programme était composé d'instructions et de macro-instructions
- Certaines macro-instructions ne sont valides que dans un macro-programme
- C'est le cas par exemple de %if et du %do itératif

Le macro-langage
Macro-programmes

- Exemple 1 :

```
%macro reportstat;
      %if &sysday = sunday %then %do;
            title "Montant des ventes de la
semaine                    ";
            proc means data = base.ventes mean;
                  where today()-6 <= date_der <=
                        today();
                  var mt_der;
            run;
      %end;
%mend reportstat;
```

Le macro-langage

Macro-programmes

- Exemple 2 :

```
%macro repartitionNbEnf(nbMax = 5);
      %do i = 0 %to &nbMax;
            title1 "Clients ayant &i enfant(s)";
            proc freq data = base.clients;
                  where nbenf = &i;
                  table sitfam;
            run;
            proc gchart data = base.clients;
                  where nbenf = &i;
                  pie3d sexe;
            run; quit;
            title;
      %end;
%mend repartitionNbEnf;
```

Le macro-langage

Macro-programmes : compilation

- Soumettre le code source a pour effet de le **compiler** et de **stocker** le macro-programme dans un catalogue appelé SASMACR
- Le macro-programme est alors disponible pour utilisation
- Le code compilé peut être **exécuté** mais pas « décompilé »
- Pour **modifier** un macro-programme, il est nécessaire d'en conserver le code

Le macro-langage
Macro-programmes : compilation

- Par défaut, le catalogue SASMACR est stocké dans la bibliothèque Work
- Il est possible de stocker les macro-programmes compilés dans une autre bibliothèque en spécifiant / store dans la première ligne du code source
- Pour cela, il faut activer l'option globale mstored et spécifier la bibliothèque à l'aide de l'option sasmstore

Le macro-langage

Macro-programmes : compilation

- Il est également possible de se passer de la forme compilée
- Pour cela, il faut :

1. Sauvegarder le code source dans un programme SAS portant **le même nom** que le macro-programme
2. Ne pas y inclure autre chose que le code source du macro-programme

Le macro-langage

Macro-programmes : appel

- Lors de l'appel du macro-programme, SAS recherche le code correspondant en trois phases **successives** :

1. Sous **forme compilée** dans Work.SASMACR
2. Sous **forme compilées** dans un catalogue SASMACR d'une bibliothèque spécifiée par l'option sasmstore
3. Sous forme de **code source** portant le nom du macro-programme, dans un répertoire précisé dans l'option sasautos (l'option mautosource doit être activée)

Le macro-langage
Macro-programmes : appel

- Exemples :

```
options mstored sasmstore = mabib;

options sasautos = (sasautos "d:/data/macro"
"d:/temp") mautosource;
```

- Ces instructions doivent être exécutées **avant** l'appel du macro-programme

Le macro-langage
Macro-programmes : débogage

- Les **commentaires** fournis à l'exécution d'un programme contenant du macro-langage **se limitent** aux messages concernant le code SAS exécuté par le compilateur SAS
- Les instructions de macro-langage sont simplement **recopiées** dans la fenêtre Log

Le macro-langage

Macro-programmes : débogage

- Pour faciliter le débogage d'un programme faisant intervenir du macro-langage, on dispose de trois **options** qui **enrichissent les messages** de la fenêtre Log :

1. symbolgen
2. mprint
3. mlogic

Le macro-langage
Macro-programmes : débogage

- symbolgen affiche un message à chaque application d'une des règles de **résolution** d'une macro-variable
- Elle est activée par

```
options symbolgen;
```

Le macro-langage
Macro-programmes : débogage

- mprint permet d'afficher **le code SAS** généré par le compilateur macro, instruction par instruction
- Elle n'agit que dans le cadre d'un **macro-programme**
- Elle est activée de la même façon que symbolgen

Le macro-langage
Macro-programmes : débogage

- mlogic permet d'afficher :

1. Les valeurs des paramètres du macro-programme
2. Les expressions évaluées dans les instructions de test (%if) et si elles sont vraies ou fausses
3. Le déroulement des boucles (%do)

Le macro-langage
Macro-programmes : débogage

- mlogic n'agit que dans le cadre d'un **macro-programme**
- Elle est activée de la même façon que les deux options précédentes
- Chacune des options **se désactive** en faisant précéder son nom de no :

```
options nomlogic nomprint nosymbolgen;
```

Références

- *SAS, Maîtriser SAS Base et SAS Macro*, H. Kontchou-Kouomegni, O. Decourt, Dunod, Chapitre 6

Pour aller plus loin…

- *SAS, Introduction au décisionnel*, S. Ringuedé, Pearson, Chapitre 9
- *SAS 9.1 Macro Language Reference*, Documentation SAS (disponible sur Internet)

Compléments
symget

- La fonction symget permet de **créer des variables dans une table SAS à partir de macro-variables** définies au préalable
- Elle s'utilise dans une étape **data** et a, dans une certaines mesure, la fonction inverse de call symput
- Par exemple,

Compléments

symget

```
data _NULL_;
      set test;
      call symput('mv1',x2);
      call symput(compress('mvv'||_N_),x2);
      call symput(x1,x2);
run;
data test2;
      set test;
      z1 = symget('mv1');
      z2 = symget('mvv'||left(put(_N_,2.)));
      z3 = symget(x1);
run;
proc print;
run;
```

Compléments
symget

- La commande symget crée par défaut une variable de type caractère de longueur égale à 200
- Afin que la taille de la table SAS manipulée ne devienne pas trop importante, il convient de définir avant toute commande symget la **longueur** de la variable à créer
- Par exemple,

Compléments

symget

```
data test2;
      set test;
      length       z1 $ 8
                   z2 $ 6
                   z3 $ 4;
      z1 = left(symget('mv1'));
      z2 = left(symget('mvv'||left(put(_N_,2.))));
      z3 = left(symget(x1));
run;
```

- L'instruction left est importante car symget aligne par défaut le texte à droite

Compléments
call execute

- L'instruction (ou **routine**) call execute permet d'utiliser le macro-langage **à l'intérieur** d'une étape **data**
- Si l'on souhaite par exemple effacer l'ensemble des macro-variables définies par l'utilisateur, on peut utiliser le macro-programme suivant :

Compléments
call execute

```
%macro delmacros;
      data macros;
            set sashelp.vmacro;
      run;
      data _NULL_;
            set macros;
            if scope = 'GLOBAL' then
                  call execute('%symdel '||
                        trim(left(name))||';');
            run;
%mend delmacros;

%delmacros;
```

Compléments
macro-variables locales et globales

- Les macro-variables créées **à l'intérieur d'un macro-programme** sont par défaut de type LOCAL
- On n'y a pas accès en dehors du macro-programme où elles interviennent
- Les macro-variables créées **en dehors des macro-programmes** (en *code ouvert*) sont par défaut de type GLOBAL et toujours accessibles

Compléments

macro-variables locales et globales

- Exemple :

```
%let mv = glo;
%macro test;
      %let mv = loc;
      %let mvv = loc;
      %put &mv &mvv;
%mend test;

%put &mv &mvv;

%test;

%put &mv &mvv;
```

Compléments
macro-variables locales et globales

- *mv* étant une macro-variable **globale**, les transformations effectuées sur celle-ci par le macro-programme **restent valables en dehors du macro-programme**
- Autre exemple :

Compléments

macro-variables locales et globales

```
%symdel mv mvv;

%let mv = glo;
%macro test;
        %local mv;
        %local mvv;
        %let mv = loc;
        %let mvv = loc;
        %put &mv &mvv;
%mend test;

%put &mv &mvv;

%test;

%put &mv &mvv;
```

Compléments
macro-variables locales et globales

- La modification effectuée sur *mv* reste locale
- Après l'exécution du macro-programme, *mv* conserve sa valeur donnée au niveau global
- L'instruction %local mv permet en fait **la coexistence** de deux macro-variables *mv* simultanément : l'une au niveau local et l'autre au niveau global
- Autre exemple :

Compléments

macro-variables locales et globales

```
%symdel mv mvv;

%let mv = glo;
%macro test;
       %global mvv;
       %let mv = loc;
       %let mvv = loc;
       %put &mv &mvv;
%mend test;

%put &mv &mvv;

%test;

%put &mv &mvv;
```

Compléments
macro-variables locales et globales

- La macro-variable *mvv* ayant été déclarée comme **globale**, elle prend sa valeur suite à l'exécution du macro-programme et celle-ci reste accessible par la suite
- Ces notions sont particulièrement utiles lorsqu'au sein d'un même programme, on souhaite que des macro-programmes puissent utiliser des macro-variables définies par les macro-programmes précédents

Compléments
%scan

- La macro-fonction %scan permet de récupérer le *nième* mot d'une chaîne de caractères
- Par exemple,

```
%let mv = x1 x2 x3;
%put %scan(&mv,2);
```

- Elle est particulièrement utile en conjonction avec une boucle %do pour récupérer des intitulés successifs de variables donnés à la suite dans une liste

Compléments
by vs. class

- Pour mener une même analyse statistique sur plusieurs sous-ensembles d'observations on dispose des instructions by et class
- Les **différences** entres ces deux instructions sont les suivantes :
- L'instruction by est valable dans **toutes** les procédures d'analyse statistique, alors que class n'est valable que dans certaines

Compléments

by vs. class

- Dans certaines procédures, class a un sens différent : elle sert à désigner les variables qualitatives
- Les valeurs manquantes de chaque variable du by définissent un sous-ensemble des données qui est pris en compte dans l'analyse
- Par défaut, les observations correspondant à des valeurs manquantes des variables de class ne sont pas prises en compte dans l'analyse

Compléments
by vs. class

- L'utilisation de by nécessite un **tri** préalable de la table SAS utilisée par la même liste de variables que celle de l'instruction by et dans le même ordre
- L'utilisation de class ne nécessite pas de tri préalable de la table SAS utilisée
- Avec l'instruction by, pendant la phase de calcul, SAS ne garde en mémoire que les observations d'une sous-groupe à la fois

Compléments
by vs. class

- Avec l'instruction class, **toutes** les observations de la table sont gardées en mémoire pendant la phase de calcul
- On utilisera donc plutôt by pour des données volumineuses

Compléments
weight vs. freq

- Dans certains cas, il est nécessaire de **pondérer** les observations de la table SAS en en entrée par les valeurs d'une variable numérique
- Pour cela, on dispose des instructions : weight et freq
- Ces deux instructions **ne peuvent être utilisées simultanément**

Compléments
weight vs. freq

- Les différences et similarités entre ses deux instructions sont les suivantes :
- La variable de l'instruction weight doit être numérique, **avec des valeurs entières ou non**. Les observations ayant un poids négatif ou nul sont ignorées
- La variable de l'instruction freq doit être numérique également. **Si les valeurs ne sont**

Compléments
weight vs. freq

pas entières, elles sont tronquées au nombre entier inférieur. Les observations ayant un poids négatif ou nul sont ignorées dans les calculs

- L'instruction weight est valable dans la plupart des procédures d'analyse statistique
- L'instruction freq n'est valable que dans certaines

Compléments
weight vs. freq

- Chaque valeur de l'instruction weight est considéré comme le poids de l'observation correspondante : **le nombre d'observations renseignées ne change pas**, il reste le même que celui de la table SAS utilisée
- Chaque valeur de l'instruction freq est considéré comme **le nombre d'occurrences** de l'observation correspondante : le nombre d'observations renseignées est alors **égal à la somme des pondérations** données dans l'instruction freq

Compléments
attrib

- Lors de la définition d'une table SAS, toutes les variables présentent un certain nombre de caractéristiques : nom, label, type, format, informat, longueur
- L'instruction attrib a pour but de centraliser la spécification de ces caractéristiques
- Exemples (à insérer dans un étape data) :

Compléments
attrib

```
attrib codePostal
       label = "Code postal de résidence du client"
       format = $5.
       length = $ 5
;
attrib dFinEff
       label = "Date de fin d'effet du contrat"
       format = ddmmyy10.
;
attrib mtAchats
       label = "Montant d'achats cumulés en euros"
       format = numx8.
;
```

Compléments
attrib

- Le **label** est un **texte libre** de 256 caractères maximum
- Le **format** est un masque **d'affichage** des données. Il permet d'assurer une transition entre les valeurs telles qu'elles sont stockées et telles qu'elles sont affichées
- **La longueur** est le nombre d'octets affectés au **stockage** de chaque valeur

Compléments
attrib

- Pour une variable de type **numérique**, la longueur peut généralement être laissée à la gestion de SAS (8 octets/valeur par défaut)
- Pour une variable de type **caractère**, on spécifie après un $ la longueur **maximale** que peuvent atteindre les valeurs
- **La gestion des attributs est la première chose à considérer lors de la création de nouvelles variables**

Compléments

Principales fonctions numériques

- round(varNum,precision)
- log(varNum)
- exp(varNum)
- sqrt(varNum)
- abs(varNum)

- varNum désigne une variable de type **numérique**

Compléments

Principales fonctions sur les dates

- today()
- day(varDate)
- month(varDate)
- year(varDate)
- weekday(varDate) : jour de la semaine (1,2,…)
- mdy(mois,jour,année)

- varDate désigne une variable de date

Compléments

Principales fonctions sur les dates

- intck(‘période’,début,fin) : nombre de débuts de période (1er du mois, dimanche, 1er janvier) écoulés entre deux dates. La période est un mot-clef à choisir parmi ‘week’, ‘month’, ‘year’.
- intnx(‘période’,début,nbPériodes) : décalage d’une date de nbPériodes dans le temps. La date en sortie correspond au **début** de la période de la date obtenue

Compléments

Principales fonctions sur chaînes de caractères

- varCar1 || varCar2 : concaténation
- substr(varCar,debut,longueur)
- upcase(varCar)
- lowcase(varCar)
- left(varCar) : supprime les blancs à gauche d'une chaîne de caractères
- trim(varCar) : supprime les blans à droite d'une chaîne de caractères

Compléments

Principales fonctions sur chaînes de caractères

- strip(varCar) : suppression des blancs à droite et à gauche d'une chaîne de caractères
- compress(varCar) : suppression des blancs dans une chaîne de caractères
- compbl(varCar) : suppression des blancs consécutifs dans une chaîne de caractères

- varCar, varCar1, varCar2 désignent des variables de type caractère

Compléments

Fonctions spéciales

- lag(var) : valeur lue à l'observation précédente
- put(varNum,format) : conversion d'une variable numérique en variable caractère
- input(varCar,format) : conversion d'une variable caractère en variable numériques

Compléments

Fonctions statistiques de l'étape data

- sum(varNum1,varNum2,varNum3,…) : somme **de plusieurs variables**
- mean(varNum1,varNum2,…) : moyenne
- min(varNum1,varNum2,…) : minimum
- max(varNum1,varNum2,…) : maximum
- median(varNum1,varNum2,…) : médiane

Compléments

Les générateurs de nombres aléatoires

- rannor(seed) : loi normale standard
- ranexp(seed) : loi exponentielle standard
- ranbin(seed,n,p) : loi binomiale de paramètres n et p
- rancau(seed) : loi de Cauchy standard
- rangam(seed, alpha) : loi gamma de paramètre de forme $\alpha > 0$

Compléments

Les générateurs de nombres aléatoires

- ranuni(seed) : loi uniforme sur [0,1]
- ranpoi(seed, lambda) : loi de Poisson de paramètre $\lambda > 0$
- rantri(seed, h) : loi triangulaire de paramètre h, $0 < h < 1$
- rantbl(seed, p1, . . . , pn) : loi discrète sur {1,…,n} de masses p1, . . . , pn

Références

- *SAS, Maîtriser SAS Base et SAS Macro*, H. Kontchou-Kouomegni, O. Decourt, éditions Dunod
- *SAS, Introduction au décisionnel*, S. Ringuedé, éditions Pearson