Les macros SAS

Introduction

Cette partie très importante : il vous faudra utiliser le macrolangage pour réaliser votre projet de programmation.

Le macrolangage améliore les possibilités du langage de base. Il permet :

- de passer des paramètres entre les étapes DATA et PROC,
- d'augmenter la rapidité des écritures en permettant la répétition à volonté d'une étape,
- d'automatiser des programmes, sous forme de modules paramétrés. Ces outils sont facilement utilisables par des personnes ne connaissant pas SAS. Il leur suffit de saisir les paramètres du module.

Exemple 1

```
/* Cas 1 */
PROC MEANS DATA=base.clients MEAN ;
 VAR nbenf ;
RUN ;
/* Cas 2 */
PROC MEANS DATA=base.intranet MEAN SUM ;
 VAR DUREE NBPAGES ;
RUN ;
```

Entre les deux procédures MEANS présentées dans le précédent encadré, on peut trouver des éléments de syntaxe communes et d'autres qui peuvent varier; ces derniers peuvent être ms sous forme de paramètres, par exemple TABLE, STATS et LISTEVAR.

Pour retrouver les deux cas de l'exemple précédent, il suffit alors de données des valeurs adéquates aux trois paramètres : ainsi, TABLE vaut base.clients, STATS vaut MEAN et LISTEVAR vaut nbenf dans le premier cas.

Le fonctionnement du macro-langage

Avec le macro-langage, le paramétrage d'un programme SAS consiste à :

- écrire le code SAS du traitement à réaliser pour l'une des tables ;
- identifier les éléments dont les valeurs pourront varier, il s'agit en fait des *paramètres* du programme ;
- créer les paramètres en début de programme en attribuant à chacun d'eux un nom et éventuellement une valeur par défaut. Ces valeurs pourront être modifiées ultérieurement autant de fois que nécessaire ;
- retravailler le code SAS du traitement à réaliser précédemment écrit en remplaçant partout où cela s'avère nécessaire les valeurs des paramètres par des références à ces derniers à l'aide des noms des paramètres.

Le macrolangage est composé de :

- macro-variables
- macro-instructions
- macro-fonctions
- macros

Les **macro-variables** sont les *paramètres* du macrolangage. Pour utiliser ces paramètres dans un programme SAS, on fait précéder le nom de la macro-variable par le caractère &.

Exemple 2

proc print data=**&**table;
run;

On peut ensuite imprimer différentes tables en changeant la définition de &table. Si on définit &table=individus, les précédentes instructions permettent d'imprimer la table SAS individus.

Il y a deux types de macro-variables :

- les macro-variables créées par la programmation
- les macro-variables créés automatiquement par SAS.

Les **macro-instructions** sont des instructions spécifiques au macrolangage. Elles permettent de manipuler les macro-variables à l'aide d'expressions et de macro-fonctions. Les mots clés de ces instructions sont en général précédés du caractère %.

Exemple 3 : macro-instruction **%let**

%let table = donnees.individu ;

Le processeur macro affecte à la macro-variable table la valeur donnees.individu.

L'instruction précédente suivie des deux instructions de l'exemple 1 permet l'impression de la table individu (table permanente, stockée dans la librairie donnees).

Les **macro-fonctions** sont des fonctions spécifiques au macrolangage servant à manipuler les macrovariables. Les noms des macro-fonctions commencent toujours par le caractère %.

Exemple 4 : macro-fonction **%scan**

%let semaine = lundi - mardi - mercredi - jeudi - vendredi - samedi - dimanche ;
%let jour2 = **%scan**(&semaine,2,'-') ;

Dans cet exemple, la macro-variable jour2 correspond au 2ème mot de la chaîne de caractère semaine (mardi).

Les **macros** sont une suite de macro-instructions. Elles permettent de manipuler et de générer du texte. On peut par exemple générer de cette façon un programme SAS paramétré avec des instructions ou des étapes répétées.

Le bloc d'instructions définissant la macro est encadré des instructions **%macro** et **%mend**.

Pour utiliser la macro, on fait précéder le nom de la macro du caractère %.

Exemple 5 :

```
%macro imprim (table) ;
        proc print data = &table ;
        run ;
%mend ;
%imprim(donnees.individu) ;
```

La macro imprim permet d'imprimer une table SAS. Dans le précédent exemple, elle imprimera la table donnees.individu.

Certaines macro-instructions peuvent être utilisées en dehors d'un bloc macro, en code ouvert (open code). C'est le cas de l'instruction %let.

I. Les macro-variables

Les macro-variables servent à stocker du texte, et à le passer en paramètre d'une étape à l'autre. Elles n'ont pas de type (numérique ou caractère) comme les variables SAS.

Chaque macro-variable a une valeur unique à un instant donné. Cette valeur peut être modifiée par le programme (macro-instructions, macro-fonctions).

Une macro-variable n'est pas une variable SAS appartenant à une table SAS.

Principales différences entre macro-variables et variables SAS

Macro-variable	**Variable SAS**
Une macro-variable n'est pas rattachée à une table SAS.	Une variable SAS est rattachée à une table SAS : c'est une de ses colonnes.
Une macro-variable ne contient qu'une seule valeur.	Une variable SAS a une valeur par observation de la table SAS à laquelle elle est rattachée.
La valeur d'une macro-variable est systématiquement du texte.	Une variable SAS est de type numérique ou caractère.

Nom des macro-variables

Le nom d'une macro-variable doit :

- comporter de 1 à 8 caractères : lettres, chiffres et _ (blanc souligné) ;
- commencer par un lettre ou _.

La casse n'a pas d'importance pour le nom d'une macro-variable. Il est par exemple strictement équivalent d'écrire nomMV, Nommv, NOMMV, etc.

La longueur maximale autorisée pour le nom d'une macro-variable est de 32 caractères depuis la version 8 de SAS.

Appel d'une macro-variable

L'appel d'une macro-variable se fait par son nom précédé du signe &. ***&****table*.

Contenu d'une macro-variable

Le contenu ou valeur d'une macro-variable est une chaîne de caractères, même s'il s'agit d'un nombre, d'une opération, d'une date ou d'un morceau de code. La casse d'une macro-vraiable est importante car il s'agit d'un texte. Ainsi, il n'est pas équivalent d'écrire table, TABLE ou Table : il s'agit de valeurs différentes. Tous les caractères alphanumériques sont autorisés pour écrire la valeur d'une macro-variable :

- des lettres
- des chiffres
- des blancs
- des caractères spéciaux
- des délimiteurs.

Toutefois, certains caractères ou mots spécifiques du langage SAS dans la valeur d'une macro requièrent un traitement particulier à l'aide de macro-fonctions de quoting. C'est le cas, en particulier, de l'apostrophe, mais aussi, selon le contexte, du signe égal, de la virgule et du point-virgule.

La longueur maximale d'une macro-variable est de 65 534 caractères en version 9 de SAS, 32 767 caractères en version 8.

Exemple 6

%let semaine = lundi - mardi - mercredi - jeudi - vendredi - samedi - dimanche ;

Le nom de la macro-variable est *semaine*.

La valeur de la macro-variable est *lundi - mardi - mercredi - jeudi - vendredi - samedi - dimanche*.

Un macro-variable ne contient qu'une valeur. La valeur d'une macro-variable est initialisée et modifiable par les macro-instructions appropriées.

Création de macro-variables

Une macro-variable peut être créée de plusieurs manières. Une première méthode, avec une macro-instruction %LET, peut s'utiliser n'importe où dans un programme SAS. Une deuxième, avec la fonction CALL SYMPUT, s'utilise dans une étape Data pour créer des macro-variables à partir d'une ou de plusieurs variables SAS.

• L'instruction %let

La macro-instruction **%let** crée une macro-variable et lui affecte une valeur.

Une macro-intsruction %let peut s'utiliser n'importe où dans le code d'un programme SAS.

Syntaxe :

%let nom-de-la-macro-variable = valeur ;

Le champ *valeur* peut-être :
- vide
- une chaîne de caractères
- une expression avec des macro-variables
- une expression avec des macro-fonctions
- l'appel d'une macro

Exemple 7

%let semaine = lundi mardi mercredi jeudi vendredi samedi dimanche ;

%let list_var = &age &salaire ;

%let total1 = 31 + 28 ;

%let total2 = %eval(&i+1) ;

Nous reviendrons ultérieurement sur la macro-fonction %eval.

Les blancs à gauche et à droite de la chaîne de caractères sont automatiquement éliminés.

Tous les types de valeurs peuvent être cités sans guillemets, puisque systématiquement la valeur d'une macro-variable est considérée comme du texte. Quand des guillemets sont présents, comme par exemple :

%let LIBELLE= »Durée de connexion » ;

ils font partie intégrante de la valeur et seront recopiés lors de son utilisation.

La valeur de total1 reste le texte 31+28, et non pas le résultat de l'opération puisqu'il s'agit là encore d'un texte.

• Macro-variable indice

On peut générer une macro-variable indice notamment avec l'instruction **%do…%to**. Cette instruction est utilisable uniquement à l'intérieur d'une macro.

Exemple 8

```
%macro boucle ;
        %do i = 1 %to 10 ;
                proc print data = tab&i ;
                run ;
        %end ;
%mend ;
```

L'instruction %do crée la macro-variable i et lui affecte successivement les valeurs 1 à 10.

La macro boucle imprime les tables tab1 à tab10.

Sans macro-langage, il faut répéter 10 fois la proc print.

• Les instructions %global et %local

Ces instructions initialisent des macro-variables contenant une chaîne de caractère vide.

Syntaxe :

%global nom ;

%local prenom ;

L'utilisation des macro-instructions %global et %local sera abordée plus en détail lorsque nous aborderons l'environnement de référence.

• L'instruction CALL SYMPUT

L'instruction **call symput** permet de créer une ou plusieurs macro-variables en leur affectant les valeurs d'une variable d'une table SAS. Cette instruction s'utilise obligatoirement dans une étape data.

Syntaxe :

CALL SYMPUT (nom_macro-variable, valeur_macro_variable) ;

Le premier argument de l'instruction CALL SYMPUT définit le nom des macro-vraiables créées au cours de l'étape Data. Cette expression peur contenir :

- le nom de la macro-variable, entouré de guillemets,
- une formule de calcul de l'étape Data, utilisant des chaînes de caractères entre guillemets, des variables SAS et des fonctions.

Le deuxième argument de l'instruction 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 que seulement un nom de variable.

Exemple 9

On veut récupérer dans une macro-variable N le nombre d'individus du fichier tab. On peut le faire au cours d'une étape data _NULL_ :

```
data _NULL_ ;
  set tab ;
  call symput('N',_N_) ;
run;
```

Dans cet exemple, on copie de manière répétée, à toutes les observations lues, la valeur de la variable _N_ (numéro de l'observation lue). Il en résulte que, à la fin de l'étape Data, la macro-variable N a pris le numéro de la dernière observation lue.

Il est également possible de créer autant de macro-variables qu'il y a d'observations dans une table. Pour cela, il faut s'assurer que le premier argument de CALL SYMPUT changera à chaque observation. Il sera en général construit avec la formule suivante :

COMPRESS("macro-variable"||_N_)

Cette formule concatène la « racine » commune à tous les noms de macro-variables générés, un numéro qui s'incrémente à chaque observation. Ici, on obtient une séquence de macro-variables appelées macro-variable1, macro-variable2…

Exemple 10

```
data _NULL_;
 set base.clients;
 CALL SYMPUT(COMPRESS("nomCli"||_N_),nom);
run;
```

Cette exemple récupère, dans autant de macro-variables qu'il est nécessaire, les noms des clients. Les macro-variables s'appelleront NOMCLI1, NOMCLI2…

Les macro-variables automatiques

Les macro-variables automatiques sont créées lors de l'appel de SAS.

Elles ont des noms préfixés par SYS.

Elles donnent des informations sur le déroulement des étapes SAS.

Les principales macro-variables automatiques sont les suivantes :

SYSSCP	Système d'exploitation
SYSSITE	Numéro de site
SYSVER	Version de SAS
SYSCMD	La macro-variable SYSCMD reçoit la dernière commande issue de la ligne de commande d'une fenêtre macro.
SYSDATE	Date d'ouverture de la session en cours. Elle est donnée par le système au format DDMMYY7.
SYSDAY	Jour d'ouverture de la session, en anglais.
SYSTIME	Heure d'ouverture de la session
SYSJOBID	User (dépend du système d'exploitation)

SYSERR	Code retour de la dernière étape PROC ou DATA. Elle peut contenir cinq types de valeurs : 0 – L'éxécution s'est bien effectuée sans message d'avertissement 1 – L'exécution a été interrompue par l'utilisateur avec l'instruction CANCEL 2 – L'exécution a été interrompue par l'utilisateur avec les commandes ATTN ou BREAK 4 – L'exécution s'est entièrement effectuée mais avec des messages d'avertissement >4 – Une erreur s'est produite.
SYSRC	Code retour de la dernière instruction filename
SYSLIBRC	Code retour de la dernière instruction libname
SYSMSG	Message affichable dans la zone réservée aux messages d'une fenêtre macro. Cette macro-variable est remise à zéro (_blank_) après chaque exécution de l'instruction %DISPLAY. Ne fonctionne pas sous tous les les systèmes ni pour toutes les versions de SAS.

Suppression d'une macro-variable

Pour supprimer une ou plusieurs macro-variables, on utilise la macro-instruction %SYMDEL :

%SYMDEL macro-variable1 macro-variable2… ;

Pour accéder à la valeur d'une macro-variable

Macro-instruction %PUT

Il est possible de consulter la valeur d'une macro-variable dans la vue SASHELP.VMACRO. On peut aussi l'afficher dans la fenêtre *log* en utilisant une macro-instruction %PUT.

Syntaxe :

%put texte &nom_de_macro-variable ;

Exemple 11

```
%let semaine = lundi mardi mercredi jeudi vendredi samedi dimanche ;
%put &semaine ;
```

Les expressions suivantes permettent d'afficher les valeurs de toutes les macro-variables définies dans la session :

%put _ALL_	toutes les macro-variables
%put _AUTOMATIC_	toutes les macro-variables automatiques
%put _USER_	toutes les macro-variables créées par l'utilisateur

Cette macro-instruction sert en fait à afficher un texte quelconque dans la fenêtre *Log*. Ce texte peut contenir des références à des macro-variables.

Exemple 12

```
%let nbClients=15 ;
%let nomTab=ventes;
```

```
%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_ ;
```

La fenêtre Log affiche alors les informations suivantes :

```
   %PUT La table BASE.CLIENTS contient &nbClients observations;
La table BASE.CLIENTS contient 15 observations
   %PUT La table sur laquelle on travaille est "&nomTab";
La table sur laquelle on travaille est "ventes"
   %PUT La table sur laquelle on travaille est '&nomTab';
La table sur laquelle on travaille est '&nomTab'
```

Consultation de la vue SASHELP.VMACRO

Une seconde méthode va rechercher dans la vue SASHELP.VMACRO les macro-variables définies par l'utilisateur (par opposition à celles crées par SAS). Elles se repèrent à la valeur de leur variable SCOPE, qui vaut GLOBAL.

VIEWTABLE: Sashelp.Vmacro

	Macro Scope	Macro Variable Name	Offset into Macro Variable	Macro Variable Value
1	GLOBAL	NBCLIENTS	0	15
2	GLOBAL	SEMAINE	0	lundi
3	GLOBAL	NOMTAB	0	ventes

On peut aussi utiliser la procédure print suivante :

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

On obtient les sorties suivantes dans la fenêtre Output :

Macro Variable Name	**Macro Variable Value**
NBCLIENTS	15
NOMTAB	ventes

Référence et résolution

Pour faire référence à une macro-variable, on fait précéder son nom du caractère &. Par exemple, pour une macro-variable de nom NOMMV, on utilisera &NOMMV pour y faire référence.

La résolution des macro-variables est effectuée par le compilateur macro. Après remplacement des noms de macro-variables par leurs valeurs, le programme résultant est traité comme s'il était composé de langage SAS de base par le compilateur SAS.

Il y a 5 règles pour résoudre les références aux macro-variables en lisant l'expression de la gauche vers la droite :

1. **&nomMv _** remplacé par la valeur de la macro-variable appelée NOMMV ;
2. **&nomMv. _** remplacé par la valeur de la macro-variable appelée NOMMV ;

   & et **.** sont des délimiteurs de macro-variables.

   & marque le début. Le point marque la fin. Le point n'est pas obligatoire si la fin est implicite, c'est-à-dire si la nom de la macro-variable est suivie d'un blanc ou d'un & marquant le début d'une autre macro-variable.

   &AB représente la macro-variable AB
   &A&B est la concaténation des macro-variables A et B
   &A.B est la concaténation de la macro-variable A et de la chaîne de caractère B
3. **&& _** remplacé par un seul & pour la lecture suivante de l'expression ;
4. Le compilateur macro fait autant de lectures qu'il en faut pour éliminer tous les & de l'expression considérée ;
5. On ne prend pas en compte toute expression placée entre deux guillemets simples ou quotes : '&nomMV' est considéré comme un texte à renvoyer tel quel.

Exemple 13

La macro-variable *RE* contient la valeur *mer*

La macro-variable *B* contient la valeur *veille*

La macro-variable *REveille* contient la valeur *matin*

&&RE&B se résout d'abord en *&REveille*, puis en *matin*.

&RE&B se résout en *merveille*.

&RE&B.ux se résout en *merveilleux*.

NB : les & multiples sont intéressants à utiliser lorsque l'on manipule des macro-variables indicées.

Par exemple, pour référencer les macro-variables &group1 &group2 &group3… &group10, on pourra utiliser l'écriture &&group&i en faisant varier la macro-variable i de 1 à 10.

Exemple 14 : résolution de macro-variables : définition de quelques macro-variables

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

On crée dans cet exemple quatre macro-variables; on étudie ensuite comment elles sont résolues, seules ou combinés, en application des règles définies plus haut.

- &nomTab _ clients (règle 1)
- &nomTab**.** _ clients (règle 2)
- &nomTab1 _ ventes (règle 1)
- &nomTab**.**1 _ clients1 (règle 2)
- &nomBib&nomTab _ baseclients (règle 1)
- &nomBib**.**&nomTab _ baseclients (règle 2 jusqu'au point, puis règle 1)
- &nomBib**..**&nomTab _ base.clients (règle 2 jusqu'au point, puis règle 1)
- &nomTab&i _ clients1 (règle 1)
- &&nomTab&i _ &nomTab1 (règle 3 pour les deux &, puis règle 1 pour &i; règle 4 pour lire de nouveau l'expression générée (&nomTab1) _ ventes (règle 1).

II. Les macro-fonctions

Les macro-fonctions permettent de manipuler les macro-variables. Le principe est semblable à celui d'une fonction SAS agissant sur une variable SAS.

Le nom de certaines fonctions est analogue au nom des fonctions utilisées dans l'étape data. Il est précédé de %.

Fonctions manipulant des chaînes de caractère

%index(&mvar,chaine) recherche la chaîne de caractères chaîne dans la macro-variable mvar

Exemple 15

```
%let semaine = lundi mardi mercredi jeudi vendredi ;
%let position1 =  %index(&semaine,mar) ;
%let position2 =  %index(&semaine,dim) ;
%put &position1 &position2 ;
```

&position1 contient 7 (la chaîne de caractère *mar* commence à la 7ème position)
&position2 contient 0.

%length(&mvar) retourne la longueur de la macro-variable mvar

Exemple 16

```
%let longueur = %length(&semaine) ;
%put &longueur ;
```

&longueur contient 35.

%scan(&mvar,n,delim) retourne le nième mot de mvar. En 3ème argument, on peut préciser des délimiteurs.

N'utilisez pas la virgule, ni le point-virgule ; comme délimiteurs car il y aurait des confusions.

Par exemple, si on définit la macro-variable &semaine de la manière suivante :
%let semaine = lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche ;

La macro fonction %scan(&semaine,3,",") est interprétée comme si le délimiteur était ". Un message d'erreur est retourné, car le macro processeur identifie 4 arguments (séparés par les virgules) et seulement 3 arguments sont autorisés.

Si on définit la macro-variable &semaine de la manière suivante :
%let semaine = lundi ; mardi ; mercredi ; jeudi ; vendredi ; samedi ; dimanche ;

La macro-variable &semaine contient la valeur lundi uniquement. Chacun des autres jours sera considéré chacun comme une ligne d'instruction que le processeur SAS ne saura évidemment pas interpréter.

Exemple 17

```
%let semaine = lundi – mardi – mercredi – jeudi – vendredi – samedi – dimanche ;
```

Les 4 macro-fonctions suivantes retourneront toutes la valeur mercredi :

```
%let jour3=%scan(&semaine,3) ;
%let jour3=%scan(&semaine,3,-) ;
```

```
%let jour3=%scan(&semaine,3,_-_) ;
%let jour3=%scan(&semaine,3,_-_) ;
```

Certains délimiteurs sont automatiquement reconnus par SAS, mais il est plutôt recommandé de préciser le délimiteur en 3ème argument.

%substr(&mvar,i,n) extrait n caractères à partir du ième dans le contenu de mvar.

Exemple 18

```
%let semaine = lundi-mardi-mercredi-jeudi-vendredi-samedi-dimanche ;
%put %substr(&semaine,6,7) ;
```

La dernière instruction renvoie la chaîne de caractères –mardi-

%upcase(&mvar) retourne le contenu de mvar en majuscules.

Exemple 19

```
%let semaine = lundi-mardi-mercredi-jeudi-vndredi-samedi-dimanche ;
%put %upcase(&semaine) ;
```

La précédente instruction renvoie la chaîne de caractères LUNDI-MARDI-MERCREDI-JEUDI-VENDREDI-SAMEDI-DIMANCHE.

Il est possible de combiner plusieurs fonctions :

Exemple 20

```
%let longjour3 = %length(%scan(&semaine,3,-)) ;
%put &longjour3 ;
```

&longjour3 contient la valeur 8.

Fonctions d'évaluation

La valeur d'une macro-variable est un texte. Par exemple,la valeur d'une macro-variable à laquelle on affecte 31 + 28 est 31 + 28 et non 59. On peut forcer le compilateur macro à faire des opérations avec les macro-fonctions %EVAL et %SYSEVALF. Les opérations supportées se limitent aux additions, soustractions, multiplications et divisions.

%eval(expression) évalue des calculs sur des entiers à partir d'expressions contenant des macro-variables.

Si la valeur évaluée contient des décimales, la valeur est tronquée à la partie entière.

Exemple 21

```
%let i = 4 ;
%let j = &i+1;
%let k = %eval(&i+1);
```

La macro-variable j contient 4+1. La macro-variable k contient 5.

%sysevalf(expression,type de conversion) évalue des calculs sur des valeurs numériques décimales à partir d'expressions contenant des macrovariables.

Le deuxième argument, non obligatoire, permet éventuellement de convertir le résultat. Les valeurs possibles pour ce paramètre sont :

boolean	0 si le résultat est nul ou manquant, 1 sinon
ceil	le plus petit entier supérieur ou égal à l'évaluation de l'expression
floor	le plus grand entier inférieur ou égal à l'évaluation de l'expression
integer	partie entière

Exemple 22

```
%let i = 5 ;
%let j = %sysevalf(&i/2) ;
%let test1 = %sysevalf(&i/2 , boolean) ;
%let test2 = %sysevalf(&i/2 - . , boolean) ;
%let plafond=%sysevalf(&i/2 +3 , ceil);
%let plancher = %sysevalf(&i – 5.5 , floor) ;
&j contient la valeur 2.5
&test1 contient 1
&test2 contient 0
&plafond contient 6
&plancher contient –1
```

Fonctions SAS en macrolangage

La macro-fonction **%sysfunc()** permet d'utiliser les fonctions SAS en macrolangage.

Syntaxe :

%sysfunc(Fonction(argument(s)),<format.>)

On utilise le plus souvent des fonctions SAS sur les chaînes de caractères, comme COMPRESS, SUBSTR ou UPCASE. Il est également possible d'utiliser des fonctions pour les informations numériques (comme ROUND) : une conversion est automatiquement faite.

L'argument optionnel format permet de formater le résultat renvoyé par la fonction SAS.

Remarque : concernant les macro-fonctions %UPCASE , %SUBSTR, etc…, il est totalement équivalent d'utiliser %SYSFUNC(UPCASE(…)) ou %SYSFUNC(SUBSTR(…)), etc.

Exemple 23 : calcul d'une racine carrée

```
%let  nb = 1971 ;
%let racine2 = %sysfunc(sqrt(&nb),8.2) ;
&racine2 contient la valeur 44.40
```

Toutes ces macro-fonctions peuvent être utilisées en dehors d'un bloc macro, en code ouvert.

III. Les macros-programmes

Création d'un macro-programme (d'une macro)

Un macro-programme est un morceau de code SAS encapsulé, paramétré et stocké. Cette section présente la syntaxe d'un macro-programme, puis les différents modes de déclaration des paramètres en entrée, les macro-instructions qu'on ne peut utiliser que dans un macro-programme, et enfin la compilation d'un macro-programme.

La définition d'une macro commence par l'instruction **%macro** et se termine par l'instruction **%mend**.

%macro est suivie du nom de la macro, qui doit être un nom SAS et éventuellement des paramètres de la macro entre parenthèses. Les paramètres sont des macro-variables, créées et gérées par SAS avec les valeurs fournies lors de l'appel du macro-programme.

%mend peut être suivi du nom de la macro, mais ce n'est pas obligatoire.

Syntaxe :

%MACRO *nomMP* < *(parametre1,...,parametrep)*> </STORE> ;
corps du macro-programme nomMP
%MEND *nomMP*;

nomMP est le nom donné au macro-programme. *parametre1* à *parametrep* sont des macro-variables passées en paramètres. L'option STORE intervient lors de la compilation du macro-programme.

L'ensemble des instructions, de %MACRO à %MEND inclus, est appelé *code source* ou tout simplement *source* du macro-programme.

Exemple 24

```
%macro test ;
        %do i = 1 %to 3 ;
                proc print data = tab&i ;
                run ;
        %end ;
%mend;
```

Appel de la macro

Pour générer et exécuter le code SAS défini dans la macro, on appelle la macro par % suivi du nom de la macro et éventuellement des valeurs des paramètres entre parenthèses.

Syntaxe :

%NOMMP (*valparamètre1*,…,*valparamètrep*) ;

Exemple 25

%test ;

L'appel de la macro génère et exécute le programme suivant :

```
proc print data = tab1 ;
run ;
proc print data = tab2 ;
run ;
proc print data = tab3 ;
run ;
```

Paramétrage d'une macro

On peut définir les paramètres dans l'instruction %macro, après le nom de la macro et entre parenthèses. Les noms des paramètres doivent être des noms SAS.

Ces paramètres sont ensuite utilisés à l'intérieur de la macro, de la même façon que les macro-variables : **&** suivi de leur nom.

Il y a deux façons de paramétrer une macro : en utilisant des paramètres positionnels ou en utilisant des paramètres mots-clés. On ne peut pas utiliser les deux méthodes à la fois pour une même macro.

Paramètres positionnels

À la définition de la macro, les paramètres positionnels sont listés dans un ordre défini. Leurs noms sont simplement séparés par des virgules.

```
%macro nom_macro(F_va, F_vs, L_vaN, L_vaQ) ;
…
%mend;
```

A l'appel de la macro, on indique dans le même ordre les valeurs que doivent prendre les paramètres.

```
%nom_macro(tab1, tab2, vaN1 vaN2 vaN3, vaQ1 vaQ2 vaQ3 vaQ4);
```

SAS affecte la valeur tab1 à F_va,
tab2 à F_vs,
vaN1 vaN2 vaN3 à L_vaN,
vaQ1 vaQ2 vaQ3 vaQ4 à L_vaQ.

L'ordre dans lequel les paramètres ont été définis est donc essentiel.

Exemple 26

```
%macro printab(table,var1,var2) ;
        proc print data = &table ;
                var &var1 &var2 ;
        run ;
%mend ;
%printab(tab1,nom,prenom) ;
```

L'appel de la macro génère le programme :

```
proc print data = tab1 ;
        var nom prenom ;
run ;
```

A l'appel de la macro, tous les paramètres sont obligatoires et doivent être spécifiés dans l'ordre.

Paramètres mots-clés

À la définition de la macro, les paramètres mots-clés sont cités dans un ordre quelconque. Leurs noms sont suivis du signe égal et éventuellement d'une valeur par défaut.

La valeur par défaut est la valeur qui est utilisée à l'appel de la macro, le paramètre n'est pas cité.

```
%macro nom_macro(F_va=, F_vs=, L_vaN=, L_vaQ=) ;
…
%mend ;
```

A l'appel de la macro, on indique le nom des paramètres suivi du signe égal et des valeurs que doivent prendre les paramètres.

```
%nom_macro(F_vs= tab2, F_va=tab1, L_vaQ=vaQ1 vaQ2 vaQ3 vaQ4, L_vaN=vaN1 vaN2 vaN3) ;
```

Comme on le voit dans l'exemple précédent, l'ordre dans lequel les paramètres ont été définis n'a pas d'importance. À l'appel de la macro, on peut citer les paramètres dans un ordre différent de la définition de la macro.

Les paramètres sont optionnels, c'est-à-dire qu'on n'est pas obligé de les renseigner tous à l'appel de la macro. Dans le cas où un paramètre n'est pas renseigné, c'est la valeur par défaut spécifiée dans la définition de la macro qui est utilisée.

Cette méthode présente des avantages par rapport à la précédente :

- ordre des paramètres indifférents
- paramètres non obligatoires à l'appel de la macro
- possibilité de définir des paramètres par défaut.

Exemple 27

```
%macro printab(table=,opt=noobs,vars=) ;
        proc print data = &table &opt ;
                var &vars ;
        run ;
%mend printab ;
%printab(table=tab1, vars=age poids) ;
```

Dans cet exemple, le paramètre opt a une valeur par défaut (noobs). Si aucune valeur n'est précisée pour ce paramètre à l'appel de la macro, c'est sa valeur par défaut qui est utilisée.

L'appel de la macro précédente génère le code suivant :

```
proc print data=tab1 noobs ;
        var age poids ;
run ;
```

Appel de la macro

Les instructions décrites dans ce chapitre sont utilisables uniquement à l'intérieur d'une macro, et pas en code ouvert.

Les tests

La programmation conditionnelle en macrolangage permet de tester les valeurs des paramètres et des macro-variables générales et de générer des instructions différentes selon l'évaluation de ces valeurs.

%if expression_du_macrolangage **%then** texte ;
%else texte ;

Si le bloc des instructions comprend des points-virgules, il faut obligatoirement un bloc **%do %end**.

```
%if expression_du_macrolangage %then %do ;
    instructions
%end ;
%else %do ;
    instructions
%end ;
```

Exemple 28

```
%macro printab(table=, opt=noobs, vars= ) ;
        %if &table=  %then %put Paramètre TABLE manquant ;
        %if &vars=  %then %put Paramètre VARS manquant ;
        …
%mend printab ;
```

Exemple 29

```
%macro proced(table=, opt=noobs, vars=);
        %if %upcase(%scan(&table,1,'.'))=SASHELP %then %do ;
                proc contents data=&table ;
                run ;
        %end ;
        %else %do ;
                data temp ;
                        set &table ;
                        if age < 10 then classe = 1 ;
                        else if age < 20 then classe = 2 ;
                run;
                proc print data = temps &opt ;
                        var &vars classage ;
                run ;
        %end ;
%mend proced ;
```

Si la librairie de la table passée en paramètre est la librairie SASHELP, on génère une proc contents. On a utilisé la macro-fonction %upcase afin d'être sûr que le contenu de la chaîne de caractère à tester est en majuscules. On voit dans cet exemple l'utilité cette macro-fonction.

Sinon, on génère une étape data au cours de laquelle on créer une variable classe. On génère ensuite une proc print.
Cet exemple permet de bien faire la distinction entre la macro-instruction %if (macrolangage) et l'instruction de base if dans une étape data.

Le texte généré par le paramétrage sera donc le suivant :

```
%proced(table=sashelp.air) ;
⇒       proc contents data = sashelp.air ;
        run ;
%proced(table=base.individu, vars=age poids taille) ;
⇒       data temp ;
                set base.individu ;
                if age < 10 then classe = 1 ;
                else if age < 20 then classe = 2 ;
                else classe = 3 ;
        run ;
        proc print data = temp noobs ;
                var age poids taille classe ;
        run ;
```

Les boucles

Boucle générée par une macro-variable indice

%do indice = début **%to** fin **%by** increment ;
instructions
%end ;

Cette instruction crée une macro-variable indice qui prend successivement les valeurs de *début* à *fin* avec un pas de *incrément*.

début, *fin* et *incrément* sont obligatoirement des valeurs entières.

Exemple 30

```
%macro liste(nbitem=) ;
    %do i=1 %to &nbitem ;
        %put donnees.tab&I;
    %end;
%mend;
%liste(nbitem=8);
```

L'appel de la macro génère le texte :
donnees.tab1
donnees.tab2
…
donnees.tab8

Boucle gérée par une condition

Comme dans l'étape data, il y a deux façons d'exprimer la condition :

%do %while(expression) ;
instructions
%end ;

%do %until(expression) ;
instructions
%end ;

Exemple 31

```
%macro nbmot(chaine,delim);
  %let i=1;
  %do %until(&lm=0);
    %let lm=%lenght(%scan(&chaine,&i,&delim)) ;
    %let i=%eval(&i+1) ;
  %end ;
  %eval(&i-2) ;
%mend ;
```

La macro-fonction nbmot retourne le nombre de mots d'une chaîne de caractères.

```
%let plans = 1-2 3-4 ;    /* plans factoriels dans lequel sont représentés les individus et les
                             variables */
%let nb_plans = %nbmot(&plans,' ') ;
%put &nb_plans ;
```

&nb_plans contient la valeur 2.

On détaille ci-dessous le déroulement de la macro nbmot :

&i = 1
① &lm = %length(%scan(&plans,1, ' ')
= %length('1-2')
= 3
&i = %eval(&i+1)
= 2
② &lm <> 0 : on effectue un deuxième passage dans la boucle.
&lm = %length(%scan(&plans,2, ' ')
= %length('3-4')
= 3
&i = %eval(&i+1)
= 3
③ &lm <> 0 : on effectue un troisième passage dans la boucle.
&lm = %length(%scan(&plans,3, ' ')
= length(' ')
= 0
&i = eval(&i+1)
= 4
④ &lm = 0 : on sort de la boucle.
&nbmot = %eval(&i-2)
= %eval(4-2)
= 2

On a donc 2 plans factoriels.

Exercice 1

Programmez *Extcar*, macro-fonction ayant :
- argument : un mot,
- résultat : les deux derniers caractères du mot.

Indications : vous utiliserez les macro-fonctions substr, eval et length.

Exercice 2

Soient deux listes de variables :
L_vaN : liste de variables numériques indicée par une date (01, 02, …)
L_vaQ : liste de variables qualitatives indicée par une date (01, 02…)
(exemple : L_vaN= N1_01 N2_01 N3_01 N4_01 N5_01
N1_02 N2_02 N3_02
N1_03 N2_03 N3_03 N4_03 ;
L_vaQ= Q1_01 Q2_01 Q3_01
Q1_02 Q2_02 Q3_02 ;)

Les listes L_vaN ou L_vaQ peuvent être vides.

On veut constituer les listes :
L_indicN : liste des indices de la liste L_vaN
(dans l'exemple précédent, L_indicN= 01 02 03) ;
L_indicQ : liste des indices de la liste L_vaQ
(dans l'exemple précédent, L_indicQ= 01 02).

Indications sur l'algorithme :
1/ On crée une macro-variable N_vaN égale au nombre de variables dans L_vaN et une macro-variable N vaQ égale au nombre de variables dans L vaQ ;
2/ On crée une macro-variable N_G qui est égale aux nombres de listes non vides (1 si l'une des listes L_vaN ou L_vaQ est vide, 2 sinon).
Si L_vaN est non vide, on crée la macro-variable G1 égale à N.
Si L_vaQ est non vide, on crée une macro-variable G1 (si L_vaN vide) ou G2 (si L_vaN non vide) égale à Q ;
3/ On initialise les listes L indicN et L indicQ (à vide) ;
4/ Pour t allant de 1 à &N_G :
- *on initialise chacune des listes d'indices au premier indice de la liste (01 dans l'exemple précédent). On utilisera pour cela la fonction Extcar.*
- *on parcourt ensuite la liste de variables et si l'indice de 2 variables successives diffère (par exemple entre N5_01 et N1_02), on ajoute le nouvel indice à la liste d'indices.*

IV. Les environnements de référence

Environnements global et local

L'environnement de référence est l'étendue du champ d'action d'une macro-variable.

L'environnement global correspond à toute la session SAS y compris les macros. Toutes les macro-variables créées en dehors d'une macro font partie de cet environnement. Elles sont appelées macro-variables globales. Une macro-variable globale peut être appelée à tout moment de la session.

L'environnement local est créé par l'exécution d'une macro. Il commence au début de l'exécution de la macro et se termine à la fin de l'exécution de la macro. Les paramètres de la macro et les macro-variables créées à l'intérieur de la macro font partie de cet environnement. Elles sont dites macro-variables locales. Deux macro-variables locales appartenant à des macros différentes peuvent avoir le même nom.

Fonctionnement par défaut

Une macro-variable créée en dehors d'une macro est globale.

Une macro-variable créée à l'intérieur d'une macro est locale. Elle n'est connue qu'au moment de l'exécution de la macro.

Exemple 32

```
%macro pardeu(nbitem=) ;
  %let i=1;
  %do %until(&i>&nbitem) ;
      %let j=%eval(&i+1) ;
      data pardeu&i ;
         set &lib..tab&i
         &lib..tab&j ;
      run ;
      %let i=%eval(&i+2);
  %end ;
%mend ;
%let lib=donnees ;
%pardeu(nbitem=6);
```

nbitem, *i* et *j* sont des macro-variables locales.
lib est une macro-variable globale.

Instructions spécifiques

%global nom_de_macro-variable ;

Cette instruction permet de rendre une macro-variable globale, et en particulier d'utiliser une macro-variable créée à l'intérieur d'une macro en dehors de l'exécution de la macro.

Si l'instruction est à l'intérieur de la macro, elle doit précéder la définition de la macro-variable.

Cette instruction n'est pas applicable sur les paramètres d'une macro, qui ne peuvent être que des macro-variables locales.

L'instruction doit obligatoirement être à l'intérieur d'une macro avant la création de la macro-variable.

Exemple 33

```
%macro nbmot2(chaine) ;
    %global i ;
    %let i=1 ;
    %do %until(&lm=0) ;
        %let lm=%length(%scan(&chaine,&i)) ;
        %let i=%eval(&i+1) ;
    %end ;
    %let i=%eval(&i-2) ;
%mend ;
```

La macro-variable &i peut être utilisée après l'exécution de la macro.

%local nom_de_macro_variable ;

Cette instruction permet de rendre une macro-variable locale. La valeur de la macro-variable n'est pas mise à jour dans l'environnement global. On a ainsi deux macro-variables de même nom, une locale et une globale, pouvant avoir à un instant donné des valeurs différentes.

L'instruction doit obligatoirement être à l'intérieur d'une macro avant la création de la macro-variable.

Exemple 34

```
%let i = 50 ;

%macro liste(nbitem=) ;
    %local i ;
    %do i=1 %to &nbitem ;
        %put donnees.tab&i ;
    %end ;
%mend ;

%liste(nbitem=12) ;
%put &i ;
```

Au moment de l'exécution de la macro, il existe 2 macro-variables i. Une globale dont la valeur reste 50 et une locale qui prend successivement les valeurs 1 à 12.
Après l'exécution de la macro, il n'existe plus que la macro-variable i globale dont la valeur est toujours 50.
Sans l'instruction %local, il n'y aurait qu'une seule macro-variable i, globale. Après l'exécution de la macro, la valeur de i ne contiendrait pas 50 mais 13.

Affichage

Les expressions suivantes permettent d'afficher les macro-variables définies dans l'environnement local ou global.

%put _LOCAL_ ; toutes les macros-variables de l'environnement local
%put _GLOBAL_ ; toutes les macro-variables utilisateurs de l'environnement global.

L'interface avec l'étape DATA

Les fonctions décrites ci-dessous permettent de créer des macro-variables à partir de variables d'une table SAS et inversement de créer des variables d'une table SAS à partir de macro-variables.

La routine call symput

La routine CALL SYMPUT dans l'étape DATA permet de créer des macrovariables et de leur affecter des valeurs en relation avec les valeurs des variables d'une table SAS.

Syntaxe :

```
data nom_de_table ou _NULL_ ;
    call symput(nom, valeur) ;
run ;
```

nom, qui est le nom de la ou des macrovariables créées, peut être :

- une chaîne de caractères entre quotes : on crée une seule macro-variable
- le nom d'une variable SAS caractère : on crée autant de macro-variables que d'observations, le nom des macro-variables correspondent aux valeurs de la variable SAS caractère.
- une expression caractère qui peut être par exemple la concaténation d'une chaîne de caractère et d'une variable. Dans ce cas, on crée autant de macro-variables que d'observations.

valeur, qui est la valeur de la ou des macrovariables créées, peut être :

- une chaîne de caractères entre quotes : la valeur est alors la même pour toutes les macro-variables créées
- le nom d'une variable SAS :la valeur de la ou des macro-variables prend la valeur de la variable SAS
- une expression valide d'une étape DATA

Exemple 35

Contenu de table

var1	var2
AQU	12.5
AUV	147
BTG	25
LOI	13.8
MIP	14

```
data _NULL_ ;
    set table ;
    call symput('new','absolument') ;①
    call symput(var1,var2) ;②
    call symput('mac'||left(put(_N_,2.)),var1) ;③
run;
```

① &new contient la valeur *absolument*.

② &AQU contient la valeur 12.5
&AUV 147
&BTG 25
&LOI 13.8
&MIP 14

③ Créé 5 macro-variables. Les noms des macro-variables sont formés du radical mac concaténé au numéro d'observation (_N_) : mac1, mac2, mac3, mac4, mac5. Les valeurs de ces macro-variables sont les valeurs de la variable var1.
&mac1 contient la valeur AQU
&mac2 AUV
…
&mac5 MIP

Par défaut, la macro-variable créée via un call symput dans une étape DATA ne le sera réellement qu'une fois l'étape DATA exécutée.

Exercice 3

On a effectué une analyse factorielle des variables de la liste L_va. Les valeurs propres v1, v2…. de cette analyse factorielle sont contenues dans le fichier VAL_PR.

On veut déterminer le nombre N_F d'axes de valeur propre non nulle de l'analyse factorielle. Ce nombre N_F devra pouvoir être récupéré en dehors de la macro.

Indications :

N_F sera égal au nombre N_va de variables de la liste L_va si toutes ces variables sont indépendantes : on initialise donc la macro-variable N_F à N_va. S'il existe des collinéarités entre les variables (ce qui est en général le cas lorsque l'on traite beaucoup de variable), N_F est égal au nombre de valeurs propres non nulles : on teste la valeur des valeurs propres successives et si on rencontre une valeur propre non nulle, alors N_F est égal au rang de la valeur propre précédente.

On utilisera la routine call symput.

La fonction resolve

La fonction **resolve** permet d'utiliser les macro-variables dans l'étape data qui les a créées.

Syntaxe :

Resolve('&nom_de_macro_variable')

Exemple 36

```
data temp ;
   set table ;
   call symput ('mac'||left(put(_N_,2.)),var1) ;
   premval=resolve('&mac1');
run;
```

La variable *premval* contient la valeur de la macro-variable mac1, c'est à dire *AQU* (pour toutes les observations).

La fonction symget

La fonction SYMGET dans l'étape DATA permet de créer des variables dans une table SAS et/ou de leur affecter des valeurs en relation avec les valeurs de macrovariables.

Syntaxe :

var = **symget**(argument) ;

argument peut être :

- le nom d'une macro-variable entre quotes
- le nom d'une variable SAS caractère : les valeurs de la nouvelle variable seront les valeurs des macro-variables ayant pour nom les valeurs de la variable passée en argument
- une expression caractère pouvant être la concaténation d'une chaîne de caractère et d'une variable.

Exemple 37

```
data table;
   set table;
   length   res1 $ 12
            res2 $ 30
            res3 $ 3;
   res1=symget('new') ;
   res2=symget(var1) ;
   res3=symget('mac'||left(put(_N_,2.))) ;
run ;
```

res1 prend la valeur absolument pour toutes les observations.

var1 prend les valeurs AQU, AUV, BTG, LOI et MIP. Donc, res2 prend les valeurs des macro-variables &AQU (=12.5) pour la 1ère observation, &AUV (=147) pour la 2ème observation, &BTG (=25) pour la 3ème observation, &LOI (=13.8) pour la 4ème observation et &MIP (=14) pour la 5ème observation.

res3 prend les valeurs de la macro-variable &mac1 (=AQU) pour la 1ère observation, …&mac5 (=MIP) pour la 5ème observation.

La fonction SYMGET crée par défaut une variable de 200 caractères de long. Il faut donc penser à utiliser l'instruction LENGTH.

V. Macros compilées

Soumettre le code source a pour effet de la compiler et de stocker le macro-programme correspondant dans un catalogue appelé SASMACR. Le macro-programme compilé est alors disponible pour une utilisation ultérieure.

Par défaut, le catalogue SASMACR est stocké dans la bibliothèque WORK.

Par exemple, la macro compilée générée par l'exécution du code :

```
%macro contnum ;
…
%mend ;
```

est enregistrée dans l'entrée WORK.SASMACR.CONTNUM.MACRO

Par défaut, le compilé d'une macro est donc perdu en fin de session.

Stockage des macros

Il est possible de stocker les macros compilées dans le catalogue *sasmacr* d'une librairie permanente. On utilise pour cela :

- les options générales *sasmstore=* et *mstored*
- l'option */store* dans l'instruction *%macro*

Par exemple, pour stocker le macro-programme MP dans le catalogue SASMACR d'une bibliothèque appelée MALIB, deux instructions sont nécessaires :

```
LIBNAME malib 'nom_physique_de_librairie';

OPTIONS MSTORED SASMSTORE = malib ;
%MACRO MP (…) / STORE DES='description_de_la_macro';
  corps du macro-programme
%MEND MP;
```

L'option générale **sasmstore=** permet de préciser dans quelle librairie on souhaite enregistrer les macros compilées et dans quelles librairies se trouvent les macros compilées que l'on souhaite utiliser.

L'option générale **mstored** permet de préciser que l'on souhaite utiliser des macros qui sont enregistrées dans la librairie précisée dans l'option *sasmstore=*.

L'option **store** de l'instruction %macro indique que l'on va stocker le compilé de la macro dans le catalogue SASMACR de la librairie précisée dans l'option sasmstore=. L'option **des=** de l'instruction %macro permet d'ajouter une description qui apparaît dans la consultation du catalogue.

Exemple 38

```
libname ade 'c:\developpement\SAS_macros\ade' ;
options sasmstore=ade mstored ;

%macro AFM(parametre1= ;parametre2=….) / store des='Analyse Factorielle Multiple';
   …
%mend ;
```

La macro AFM est stockée en compilé dans le catalogue ade.sasmacr, dans l'entrée ade.sasmacr.afm.macro avec sa description.

Pour l'exécuter ultérieurement (dans un autre programme), il suffit d'indiquer les instructions suivantes :

```
libname ade 'c:\developpement\SAS_macros\ade' ;
options sasmstore=ade mstored ;
%AFM(parametre1=valeur1,parametre2=valeur2…) ;
```

Macros en autocall

Cette fonctionnalité permet de compiler une macro automatiquement au premier appel de la macro dans la session SAS.

Les étapes sont les suivantes :

- on compile une première fois la macro avec les options précédentes (options générales sasmstore et mstored, option /store dans l'instruction %macro)
- pour l'exécuter ultérieurement (dans un autre programme), on utilise les instructions suivantes :

```
filename fileref 'nom_physique_de_la_bibliothèque_de_macros' ;
options   sasautos =  fileref
                      mautosource
                      ;
```

On peut ensuite appeler la macro, qui aura été préalablement sauvegardée dans un fichier SAS, dans la bibliothèque de macros fileref. Si entre temps, on a modifié la macro, elle est automatiquement recompilée (il n'est pas nécessaire de la compiler à nouveau).

Exemple 39

On a sauvegardé la macro %Nbmot dans le fichier Nbmot.sas, dans le répertoire d:/sasmacr.

```
filename macrosas 'd:/sasmacr ';
options sasautos=    macrosas
                     mautosource
                     ;
%let semaine=lundi - mardi ;
%put %nbmot(&semaine,'-') ;
```

C'est la dernière version de la macro %Nbmot qui est exécutée (i.e. elle est de nouveau compilée automatiquement avant d'être exécutée).

Recherche du macro-programme à l'appel

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

1. sous forme compilée dans le catalogue WORK.SASMACR ;

2. puis sous forme compilée dans un catalogue SASMACR d'une bibliothèque spécifiée par l'option SASMSTORE ;

3. puis sous forme de code source portant le nom du macro-programme dans un répertoire précisé dans l'option SASAUTOS.

Le passage à la phase 2 est conditionné par l'activation de l'option MSTORED. Le passage à la phase 3 est conditionné par l'activation de l'option MAUTOSOURCE. La recherche s'arrête à la première phase où SAS trouve le code du macro-programme. S'il n'est pas compilé (phase 3 uniquement), SAS le compile, stocke la forme compilée dans le catalogue WORK .SASMACR et exécute le macro-programme. S'il n'est pas déjà compilé, SAS se contente de l'exécuter.

Pour accéder à la phase 2 et chercher dans le catalogue MALIB.SASMACR, on exécute donc avant l'appel du macro-programme l'instruction suivante.

```
OPTIONS MSTORED SASMSTORE=malib ;
```

Pour accéder à la phase 3 et chercher dans le répertoire _c:\temp_, on exécute donc avant l'appel du macro-programme l'instruction suivante :

```
OPTIONS SASAUTOS=   malib
                    MAUTOSOURCE ;
```

VI. Fonctions de compilation

Fonction %str

La macro-fonction %str empêche l'interprétation des éléments suivants : + - * / , < > = ‘ ~ ; blanc lt eq gt and or not le ge ne

Cette fonction permet donc d'insérer des caractère spéciaux dans des macrovariables, notamment :

- insérer des blancs en début et fin de macro-variables,
- insérer des points-virgules

Syntaxe :

%str(chaine_de_caractères)

Exemple 40

```
%let nom1 =            Durand ;
%let nom2 = %str(          Durand) ;

%let pgm1 = proc print ; run ;
%let pgm2 = %str(proc print ; run ;) ;
```

&nom 1 contient la valeur ‘Durand’.
&nom2 contient la valeur ‘   Durand’.
&pgm1 contient la valeur ‘proc print’ ;
&pgm2 contient la valeur ‘proc print ; run ;’.

Il est possible d'insérer une quote (simple ou double) ou une parenthèse, mais il faut la faire précéder de %.

Exemple 41

```
%let titre = %str(Feuille d%’émargement) ;
```

titre est résolu en Feuille d'émargement.

Fonction %nrstr

La macro-fonction **%nrstr** va encore plus loin que la macro-fonction %str. Elle empêche le macro-langage d'interpréter les précédents caractères (+ - * / , < > = ‘ ~ ; blanc lt eq gt and or not le ge ne) , ainsi que des **%** et **&**. Ainsi toute référence à une macro-variable (&) ou macro-programme ou fonction (%) est ignorée.

Syntaxe :

%**nrstr**(chaîne_de_caractères)

Exemple 41

```
%let var1 = A ;
%let var2 = %str(&var1*B) ;
%let var3 = %nrstr(&var1*B);
```

&var1 contient la valeur A
&var2 contient la valeur A*B
&var3 contient la valeur &var1*B.

VII. Fonctions d'exécution

Ces fonctions permettent de traiter les éléments issus de la résolution d'une macro-expression comme du texte.

Les éléments ne sont résolus qu'à l'exécution de la macro. Cela permet de traiter des éléments de syntaxe comme du texte et donc d'éviter que ces éléments soient interprétés et exécutés.

La fonction %quote

La fonction **%quote** empêche l'interprétation des éléments suivants :
+ - * / , < > = ^ ~ ; blanc
lt eq gt and or not le ge ne

Syntaxe :

%quote(chaîne_de_caractères)

Exemple 42

```
%macro test(metal=) ;
    %if %quote(&metal)= 'cuivre' %then %do ;
    …
    %end ;
%mend;
%test(metal=or) ;
```

%quote permet que 'or' (resolution de &metal) ne soit pas interprété en tant qu'opérateur, ce qui génèrerait une erreur de syntaxe.

La fonction %nrquote

La fonction **%nrquote** est équivalente à %quote mais empêche également l'interprétation des **%** et **&**.

Syntaxe :

%nrquote(chaîne_de_caractères)

La fonction %bquote

La fonction **%bquote** est équivalente à %quote mais empêche également l'interprétation des quotes (simples et doubles) et des parenthèses non précédées de %.

Syntaxe :

%bquote(chaîne_de_caractères)

La fonction %nrbquote

La fonction **%nrbquote** est équivalente à %nrquote mais empêche également l'interprétation des quotes et parenthèses non précédées de %.

Syntaxe :

%nrbquote(chaîne_de_caractères)

VIII. Programmation de fenêtres de saisie

Les macro-instructions qui vont être présentées dans cette dernière partie ne permettent pas de créer de véritables applications comme avec le module SAS/AF en langage SCL. Mais les ressources utilisées par le macro-langage sont moins difficiles à mettre en œuvre.

Syntaxe de définition d'une fenêtre

La syntaxe de définition d'une fenêtre simple est la suivante :

%WINDOW nom_fenetre <attributs> champ1 <..champn>;
① ② ③ ④

① nom de la fenêtre. Paramètre obligatoire.
Les autres paramètres sont optionnels :
② attributs de la fenêtre.
③ définition du 1er champ de la fenêtre
④ définition du nième champ de la fenêtre

Les attributs de la fenêtre

Les attributs permettent de définir l'aspect général de la fenêtre.

%WINDOW nom_fenetre
COLOR=*color* ①
COLUMNS=*columns* ②
ROWS=*rows* ③
ICOLUMN=*column* ④
IROW=*row* ⑤
KEYS=<<*libref.*>*catalog.*>*keys-entry* ⑥
MENU=<<*libref.*>*catalog.*>*pmenu-entry* ⑦
champ1 <…champn>;

① COLOR = *color*
Spécifie la couleur de fond. Les couleurs possibles sont les suivantes :
BLACK, BLUE, BROWN, CYAN, GRAY (ou GREY), GREEN, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

② COLUMNS = *columns*
Détermine la largeur de la fenêtre (par défaut la largeur est maximale)

③ ROW= *rows*
Détermine la hauteur de la fenêtre (valeur maximale par défaut)

④ ICOLUMN = *column*
Abscisse du point supérieur gauche de la fenêtre (1 par défaut)

⑤ IROW = *row*
Ordonnée du point supérieur gauche de la fenêtre (1 par défaut)

⑥ KEYS = <<*libref.*>*catalog.*>*keys-entry*
Définition des touches de fonctions à utiliser (correspond à une entrée de catalogue de type KEYS).
Si on ne précise pas *libref* et *catalog*, SAS utilise SASUSER.PROFILE.*keys-entry*.
Si on ne précise pas l'option KEYS=, SAS utilise les touches de fonctions courantes.

Pour définir les touches de fonction, on peut utiliser la fenêtre KEYS (en tapant KEYS sur la ligne de commande), ou la commande KEYDEF ou encore la macro %KEYDEF.

⑦ MENU = <<*libref.*>*catalog.*>*pmenu-entry*
Définition des barres de menu (correspond à une entrée de catalogue de type PMENU).
Si on ne précise pas *libref* et *catalog*, SAS utilise SASUSER.PROFILE.*pmenu -entry*.

La définition d'un PMENU se fait avec la procédure PMENU.

La définition des touches de fonction et des barres de menu n'est pas propre au langage macro. Ces éléments se retrouvent dans les applications développées en langage SCL.

Les différents champs de la fenêtre

Il existe deux sortes de champs dans les fenêtres macros :
- les champs constants contenant du texte et/ou une macro-expression (la valeur est connue lors de la définition de la fenêtre).
- Les champs modifiables (dont la valeur n'est pas connue au moment de la définition de la fenêtre : par exemple, des champs de saisie modifiables par l'utilisateur).

La syntaxe de déclaration d'un champ est la même dans les deux cas et comprend les informations suivantes :
- définition de position ;
- définition de contenu (ou contenant du champ)
- options du champ.

Exemple 43 : champ constant de type texte

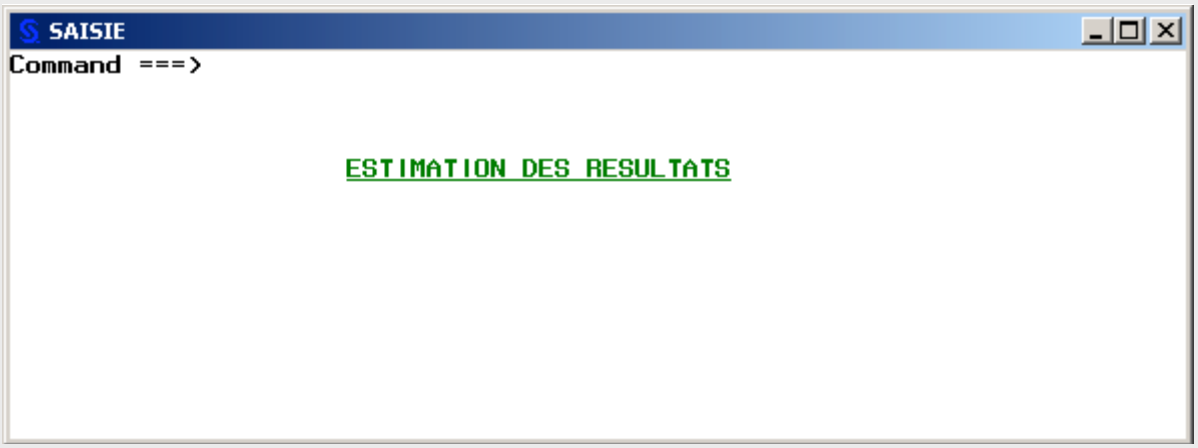
```
%WINDOW saisie
  #4  @22    "ESTIMATION DES RESULTATS" ATTR=UNDERLINE COLOR=GREEN;
  ①  ②       ③                                ④                    ⑤
```

① # positionne le champ à la ligne indiquée (ici, la 4ème ligne)
② @ positionne le champ à la colonne indiquée (ici, la colonne 22)
③ Définition du champ (constant de type texte)
④ Attribut d'affichage du texte (ici en souligné). Les valeurs possibles sont :
BLINK (le champ clignote), HIGHLIGHT (le champ est affiché avec une intensité élevée), REV_VIDEO (le champ est affiché en vidéo inversée. Par exemple, dans l'exemple précédent, les caractères seraient affichés en blanc sur fond vert), UNDERLINE (le champ est souligné)
⑤ Couleur d'affichage du champ
Les couleurs peuvent être : BLACK, BLUE, BROWN, CYAN, GRAY (ou GREY), GREEN, MAGENTA, ORANGE, PINK, WHITE, YELLOW

Si on exécute la commande :

```
%DISPLAY saisie ;
```

La fenêtre ci-dessous apparaît.



Exemple 44 : champ constant comportant une macro-expression

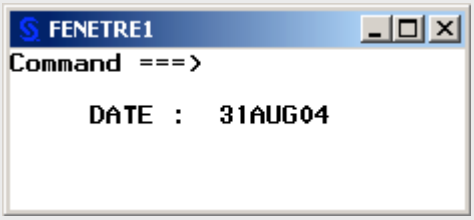
```
%WINDOW fenetre1
  /    +5       "DATE :  &SYSDATE";
  ①  ②        ③
```

① / positionne le champ É à la ligne suivante colonne 1
② + décale le champ É de 5 colonnes
③ définition du champ (texte avec macro-expression). La définition du champ se fait entre deux " ou ‘. La résolution de la macro-expression impose l’utilisation de ".

Si on exécute la commande :

%DISPLAY fenetre1 ;

La fenêtre ci-dessous apparaît.



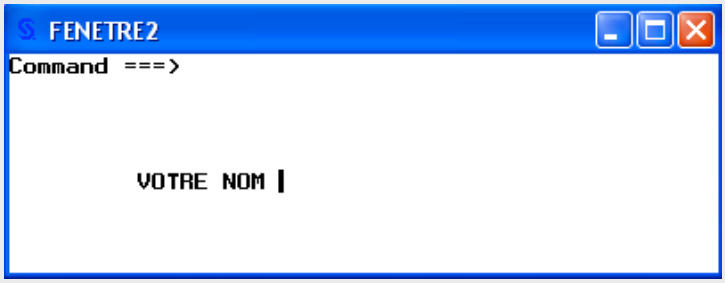
Exemple 45 : champ modifiable de saisie

```
%WINDOW fenetre2
#5 @10 'VOTRE NOM'  @20    NOM 10   REQUIRED=YES     DISPLAY=YES;
                    ①      ②        ③                ④
```

%DISPLAY fenetre2;

① Positionne le champ ② à la colonne 20
② Définition d’un champ de saisie. La longueur du champ de saisie est fixée à 10 caractères. La valeur saisie est enregistrée dans la macro-variable NOM.
③ Détermine si le champ de saisie doit être ou non rempli avant la fermeture de la fenêtre.
④ Indique si le champ saisi s’affiche ou non à l’écran (Oui par défaut ? On utilisera non pour la saisie d’un mot de passe par exemple).

La fenêtre suivante s'affiche :



Exemple 46 : champ protégé

```
%WINDOW fenetre3
 #7 @10 'BONJOUR' @20 NOM 10 PROTECT=YES DISPLAY=YES;
%DISPLAY fenetre3;
```

L'attribut PROTECT indique que le champ n'est pas modifiable par l'utilisateur. Il faut donc, préalablement à l'instruction %DISPLAY, affecter une valeur à la macro-variable nom.

Fenêtre composée

Une fenêtre composée autorise la définition de plusieurs groupes de champs à l'intérieur d'une même fenêtre. Chaque groupe est alors affiché individuellement suivant la valeur d'un paramètre.

Syntaxe générale :

```
  %WINDOW fenetre2 <attributs> ①
      GROUP=<groupe1>          ②
          champ1 <…champn> ;   ③
  …
      GROUP=<groupen>          ④
          champ1 <…champn> ;   ⑤
```

① nom et attribut de la fenêtre
② définition du premier groupe de champs
③ définition des champs du premier groupe
④ définition du n^{ième} groupe de champs
⑤ définition des champs du n^{ième} groupe

Exemple 47

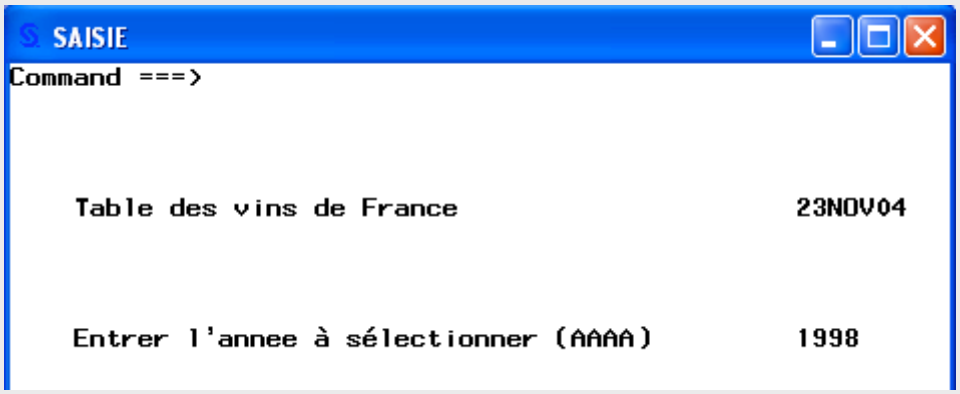
```
%WINDOW saisie
  GROUP=commun
  #5  @5  "Table des vins de France"  @50  "&SYSDATE"

  GROUP=annee
  #10  @5  "Entrer l'annee à sélectionner (AAAA)"
       @50  annee  4 required=YES

  GROUP=vin
  #10  @5  "Entrer le vin désiré"
       @50 vin 15 REQUIRED=YES
;
```

```
%MACRO affiche(choix) ;
  %DISPLAY saisie.commun NOINPUT ;
  %IF &choix=annee %THEN %DISPLAY saisie.annee ;
  %ELSE %DISPLAY saisie.vin ;
%MEND affiche ;
```

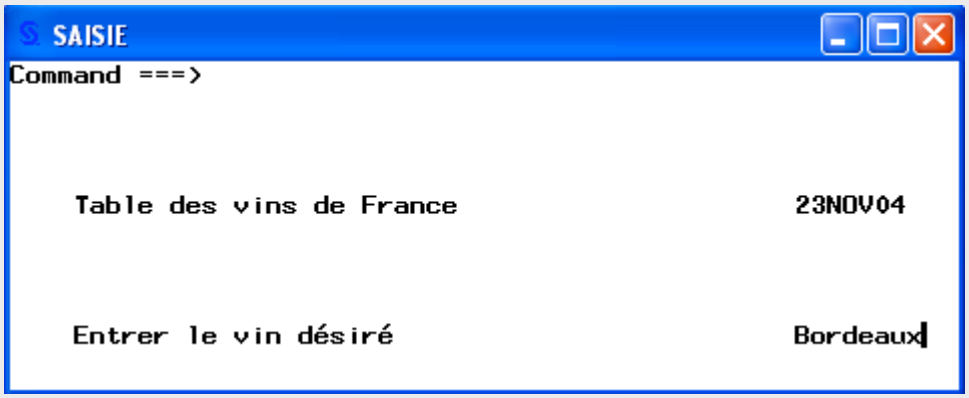
Les commandes suivantes permettent de saisir successivement l'année et le vin choisi :

```
%affiche(annee);
%affiche(vin);
```

La fenêtre suivante s'affiche en premier :



Dans l'exemple précédent, on a saisi la valeur 1998. Après avoir appuyé sur la touche Entrée, la deuxième fenêtre s'affiche :



On tape de nouveau sur la touche Entrée.
La macro-variable &annee contient la valeur 1998.
La macro-variable &vin contient la valeur Bordeaux.

Remarque : le premier %DISPLAY employé avec l'option NOINPUT interdit toute saisie de champs (s'il y en avait) et provoque l'affichage immédiat du second %DISPLAY (groupe annee ou vin suivant la valeur de sélection).

Si on omet l'option INPUT, la fenêtre suivante s'affiche :

On peut saisir des données sur la première ligne de commande. Une fois qu'on a tapé une première fois sur la touche Enter, les fenêtres précédentes s'affichent.

Exemple 48 : la sélection du groupe peut être interne à la fenêtre.

```
%WINDOW saisie
       GROUP=commun
       #5  @5  "Table des vins de France"  @50  "&SYSDATE"
       #7  @5  "Selection sur l'année (choix : 1)"
       #8  @5  "ou le type de vin (choix : 2)"
       @55 choix 1

       GROUP=annee
       #10  @5  "Entrer l'annee à sélectionner (AAAA)"
       @50  annee 4 required=YES

       GROUP=vin
       #10  @5  "Entrer le vin désiré"
       @50 vin 15 REQUIRED=YES
;

%MACRO affiche ;
       %DISPLAY saisie.commun ;
       %IF &choix=1 %THEN %DISPLAY saisie.annee ;
       %ELSE %DISPLAY saisie.vin ;
%MEND affiche ;
%affiche;
```

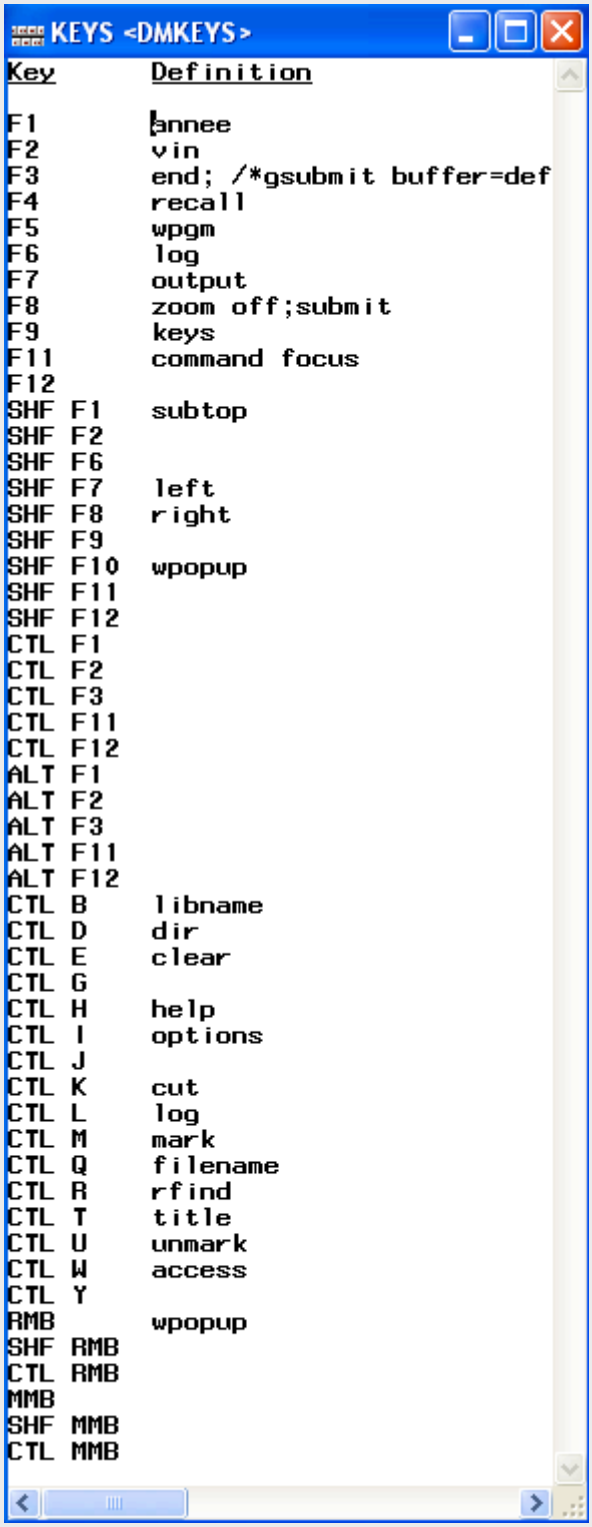
Exemple 49 : la sélection du groupe peut se faire par le biais de la ligne de commande.

```
%WINDOW saisie
    GROUP=commun
    #5  @5  "Table des vins de France"  @50  "&SYSDATE"
    #7  @5  "Entrer vin ou annee sur la ligne de commande pour faire votre sélection"

    GROUP=annee
    #10  @5  "Entrer l'annee à sélectionner (AAAA)"
        @50  annee 4
```

```
    GROUP=vin
    #10  @5  "Entrer le vin désiré"
        @50 vin 15
;

%MACRO affiche ;
    %DISPLAY saisie.commun ;
    %IF %UPCASE(&SYSCMD)=ANNEE %THEN %DISPLAY saisie.annee ;
    %ELSE %DISPLAY saisie.vin ;
%MEND affiche ;
%affiche;
```

L'affichage du groupe *vin* ou *annee* se détermine au niveau de la ligne de commande (en rentrant *vin* ou *annee*). Le contenu de la ligne de commande est retourné dans la macro variable SYSCMD qui est testé dans le macro-programme.

Exemple 50 : La sélection du groupe peut se faire par des touches de fonctions. La définition des touches de fonction peut se faire en modifiant le fichier SASUSER.PROFILE.DMKEYS.KEYS, auquel on peut accéder à partir de l'explorer.



Ici, on a redéfinit F1 sélection par année et F2 sélection par vin.

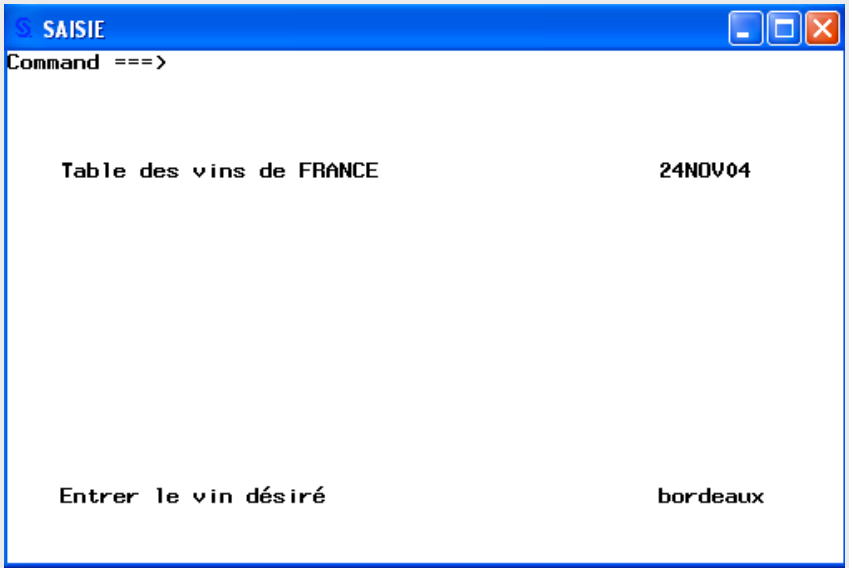
Cette manière de procéder présente un inconvénient majeur : la définition des touches de fonction est permanente. Pour revenir à la définition standard des touches de fonction, l'utilisateur doit lui-même redéfinir les touches de fonction dans la fenêtre DMKEYS.

On peut ensuite utiliser le programme suivant :

```
%WINDOW saisie
 GROUP=commun
 #5 @5 "Table des vins de FRANCE" @50 "&SYSDATE"

 GROUP=annee
 #10 @5 "Entrer l'année à sélectionner (AAAA)"
    @50 annee 4

 GROUP=vin
 #20 @5 "Entrer le vin désiré"
    @50 vin 15
;
%MACRO affiche;
 %DISPLAY saisie.commun;
 %IF %UPCASE(&SYSCMD)=ANNEE %THEN %DISPLAY saisie.annee;
 %ELSE %IF %UPCASE(&SYSCMD)=VIN %THEN %DISPLAY saisie.vin;
%MEND affiche;
%affiche;
```

Si on tape F1 sur la ligne de commande, la fenêtre suivante apparaît :

Si on tape F2, on a la fenêtre suivante :



IX. Le « débogage » des erreurs de compilation

Les commentaires fournis par défaut

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

Quelques options de « débogage »

Pour faciliter d'un programme écrit avec du macro-langage, on dispose de trois options qui enrichissent les messages de la fenêtre Journal : SYMBOLOGEN, MPRINT et MLOGIC.

L'option **SYMBOLOGEN** affiche un message à chaque application d'une des règles de résolution d'une macro-variable.

```
OPTIONS SYMBOLGEN ;
```

Elle donne lieu à l'affichage dans la fenêtre Log de messages de ce type :

```
%MACRO NBMOT(CHAINE);

 %LET I=1;
 %DO %UNTIL(&LM=0);
 %LET LM=%LENGTH(%SCAN(&CHAINE,&I));
 %LET I=%EVAL(&I+1);
 %END;
 %EVAL(&I-2)

%MEND NBMOT;

%let semaine=lundi mardi mercredi jeudi vendredi samedi dimanche;
%put %Nbmot(&semaine);

SYMBOLGEN:  Macro variable SEMAINE resolves to lundi mardi mercredi jeudi vendredi
samedi dimanche
SYMBOLGEN:  Macro variable CHAINE resolves to lundi mardi mercredi jeudi vendredi
samedi dimanche
SYMBOLGEN:  Macro variable I resolves to 1
SYMBOLGEN:  Macro variable I resolves to 1
SYMBOLGEN:  Macro variable LM resolves to 5
```

L'option de débogage **MPRINT** n'agit que dans un macro-programme.

```
OPTIONS MPRINT ;
```

L'option MPRINT donnera lieu à l'affichage dans la fenêtre Log du code SAS, détaillé instruction par instruction, généré par le compilateur macro.

```
%MACRO ARC;
/* FICHIER DES VARIABLES */
%LET F=linnerud;

/* LISTE DU PREMIER GROUPE DE VARIABLES ACTIVES NUMERIQUES */
%LET X=poids taille pouls;
/* LISTE DU SECOND GROUPE DE VARIABLES ACTIVES NUMERIQUES */
%LET Y=traction flexion saut;

/* LISTE DES VARIABLES SUPPLEMENTAIRES NUMERIQUES A PROJETER AVEC
X */
%LET XILL=;
/* LISTE DES VARIABLES SUPPLEMENTAIRES NUMERIQUES A PROJETER AVEC
X */
%LET YILL=;

[...]

DATA F;
 SET &F(KEEP=&ID &X &Y &XILL &YILL);
RUN;

[...]
%MEND;

MPRINT(ARC):   DATA F;
MPRINT(ARC):   SET linnerud(KEEP=ID poids taille pouls traction flexion saut );
MPRINT(ARC):  RUN;
```

Le nom du macro-programme ARC est répété, suivi du code SAS qui est transmis au compilateur SAS.

Tout comme l'option MPRINT, l'option **MLOGIC** n'agit que dans le cadre d'un macro-programme. Avec cette option, SAS va indiquer :

- les valeurs des paramètres du macro-programme ;
- les expressions évaluées dans les instructions de test (%IF) et si elles sont vraies ou fausses ;
- le déroulement des boucles (%DO).

OPTIONS MLOGIC ;

```
MLOGIC(NBMOT):  Beginning execution.
MLOGIC(NBMOT):  Parameter CHAINE has value
MLOGIC(NBMOT):  %LET (variable name is I)
MLOGIC(NBMOT):  %DO %UNTIL(&LM=0) loop beginning.
MLOGIC(NBMOT):  %LET (variable name is LM)
MLOGIC(NBMOT):  %LET (variable name is I)
MLOGIC(NBMOT):  %DO %UNTIL() condition is TRUE; loop will not iterate again.
MLOGIC(NBMOT):  Ending execution.
```

Il est déconseillé d'utiliser simultanément ces trois options, sous peine de noyer la fenêtre log de messages et d'en rendre le dépouillement fastidieux. Chacune de ces options se désactive en faisant précéder son nom de NO :

OPTIONS NOMLOGIC NOMPRINT NOSYMBOLGEN ;

Résumé des notions les plus importantes

Le macro-langage fournit des outils essentiels pour paramétrer un programme SAS. La présence du macro-langage dans le code SAS est repérée grâce aux caractères % et &. Le macro-langage utilise plusieurs concepts spécifiques : macro-instructions, macro-variables, macro-fonctions et macro-programmes.

Le macro-langage est analysé et traité par un compilateur spécifique qui ne connaît pas le langage SAS et ne traite que du texte.

Les macro-variables sont des paramètres créés et utilisés dans un programme SAS. Un paramètre est caractérisé par son nom et sa valeur. Pour faire référence à la valeur d'un paramètre dans un programme SAS, il faut faire précéder son nom du caractère &. La valeur d'une macro-variable est toujours considérée comme du texte.

Les macro-fonctions sont des fonctions agissant sur la valeur des macro-variables.

Les macro-programmes sont des programmes SAS paramétrés, encapsulés entre une instruction de début et une autre de fin de programme. Ils sont compilés et stockés dans des catalogues en attendant un appel de macro.

Instruction	Action
%LET *nomMV* = *valeur* ;	Macro-instruction qui crée une macro-variable nomMV en lui affectant une valeur.
CALL SYMPUT(*nomMV, variable SAS*);	Instruction de l'étape Data qui crée une ou plusieurs macro-variables en récupérant des valeurs de variables SAS.
%MACRO *NOMMP*(*nomMV1,...,nomMVp*) ;	Macro-instruction qui indique le début du macro-programme NOMMP et crée ses p paramètres en entrée.
%MEND *NOMMP* ;	Macro-instruction qui indique la fin du macro-programme NOMMP.
%IF (*expression*) %THEN %DO ; instructions %END ; < %ELSE %DO ; … %END ; >	Macro-instructions permettant de réaliser des tests sur la valeur de macro-variables.
%DO *indice* = *début* %TO *fin* ;	La macro-instruction %DO crée une macro-variable INDICE dont la valeur s'incrémentera de début à fin.