临床研究SAS高级编程

SAS Macro

Outlines

- Introduction
- Macro Variables
- Macro Function
- Macro Quoting
- Macro Programs
- DATA Step and SQL Interfaces
- How to use SAS Macros

Examples (1/5)

- **Example 1: Using a Macro Variable to Select Observations to Process**

```
%let car_maker=Audi;

proc print data=sashelp.cars(where=(Make="&car_maker"));
   title "Cars information for &car_maker";
   var Model Type Origin DriveTrain EngineSize;
run;
```

Examples (2/5)

- **Example 2: Displaying System Information**

```
%let car_maker=Audi;

title1 "Cars information for &car_maker";
title2 "As of &systime &sysday &sysdate";
title3 "Using SAS Version: &sysver";

proc print data=sashelp.cars(where=(Make="&car_maker"));
   var Model Type Origin DriveTrain EngineSize;
run;
```

Cars information for Audi
As of 10:18 Sunday 23SEP12
Using SAS Version: 9.2

Obs	Model	Type	Origin	DriveTrain	EngineSize
8	A4 1.8T 4dr	Sedan	Europe	Front	1.8
9	A41.8T convertible 2dr	Sedan	Europe	Front	1.8

Examples (3/5)

- **Example 3: Iterative Processing of SAS Steps**

```
%macro classcat;
   data %do i=12 %to 16;
             age&i
        %end;
        ;
      set sashelp.class;
         if age=12 then output age12;
            %do i=13 %to 16;
            else if age=&i then output age&i;
            %end;
   run;
%mend classcat;

/*After interpretation by the macro processor, the program becomes:*/

data age12 age13 age14 age15 age16;
   set sashelp.class;
      if age=12 then output age12;
         else if age=13 then output age13;
         else if age=14 then output age14;
         else if age=15 then output age15;
         else if age=16 then output age16;
run;
```

Examples (4/5)

- **Example 4: Using a Macro Program to Execute the Same PROC Step on Multiple Data Sets**

```
%macro Classinfo;

   %do age=12 %to 16;

      proc print data=work.age&age;
         title "Students demographics of age &age in class";
         var Name Sex Height Weight;
      run;
   %end;

%mend Classinfo;
```

Examples (5/5)

- **Example 5: Conditionally Generating SAS Code**

```
%macro whatstep(  info=
                 ,mydata=);
   %if &info=print %then
   %do;
      proc print data=&mydata;
      run;
   %end;
   %else %if &info=report %then
   %do;
      options nodate nonumber ps=18 ls=70 fmtsearch=(sasuser);
         proc report data=&mydata nowd;
         column manager dept sales;
         where sector="se";
         format manager $mgrfmt. dept $deptfmt. sales dollar11.2;
         title "Sales for the Southeast Sector";
      run;
   %end;
%mend whatstep;
```

What Is the SAS Macro Facility*

- The *macro facility* is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do common tasks.
- The macro facility enables you to assign a name to character strings or groups of SAS programming statements. You can work with the names rather than with the text itself.

* From the SAS document—SAS Macro Language

What Is the SAS Macro Facility*

- The macro facility has two components:
  - *The macro processor*
    - is the portion of SAS that does the work
  - *The macro language*
    - is the syntax that you use to communicate with the macro processor
- When SAS compiles program text, two delimiters trigger macro processor activity:
  - **&name** a macro variable
  - **%name** a macro

    (%INCLUDE, %LIST, and %RUN statements are not part of the macro facility)

* From the SAS document—SAS Macro Language

What Is the SAS Macro Facility

- SAS Macro language has two components:
  - *SAS macro variables*
    - Typically, Store text; is used to insert text throughout a SAS program repeatedly
  - *SAS macro programs*
    - Use macro variables and macro programming statements to build SAS programs

How to Access SAS Macro Facility

- The SAS macro facility is a component of Base SAS.
- If you have access to SAS, you have access to the macro facility, and you can include macro facility features in your programs.

Purpose of the SAS Macro Facility

- The macro facility is a text processing facility for automating and customizing flexible SAS code.
- The macro facility supports
  - Symbolic substitution within SAS code
  - Automated production of SAS code
  - Dynamic generation of SAS code
  - Conditional construction of SAS code.

Purpose of the SAS Macro Facility

- The macro facility is a text processing facility for automating and customizing flexible SAS code.
- The macro facility enables you to
  - Create and resolve macro variables anywhere within a SAS program
  - Write and call macro programs (macros) that generate custom SAS code.

Where Can the SAS Macro Facility Be Used

- The macro facility can be used with all SAS products.
  - Customizes data set processing, PROC steps and reports
  - Passes data between steps in a program
  - Conditionally executes DATA steps and PROC steps
  - Iteratively processes DATA steps and PROC steps
  - Contains libraries of macro program routines
  - Contains libraries of macro routines for annotating SAS/GRAPH output
  - Creates functions that can be used with the macro facility
  - ……

Macro Variables (1/6)

- **How to define a Macro variable**
  - %let macrovar = macrovalue
  - Macrovar: regular SAS naming conventions Macrovalue: any string or macro expression

stored as character type length 0-32K characters

math expressions are not evaluated

quotations not needed

leading/trailing blanks are removed

Macro Variable (2/6)

- How to resolve a Macro variable
- &macrovar

NOT resolved in single quotes--you must use double quotes.

Examples :

%let var1= SAS Macro;

%let var2 = 'SAS Macro';

%let var3 = "SAS' Macro";

%let var4 = 3+4;

%put &macrovar

%put '&var1'

%put "&var1"

Macro Variable	Value
var1	SAS Macro
Var2	'SAS Macro'
Var3	"SAS' Macro"
var4	3+4
	'&var1'
	"SAS Macro"

Macro Variable (3/6)

Type of Macro variables

Automatic/User-defined macro variable

Global/Local Macro Variable

It depends on where the macro variable is defined If

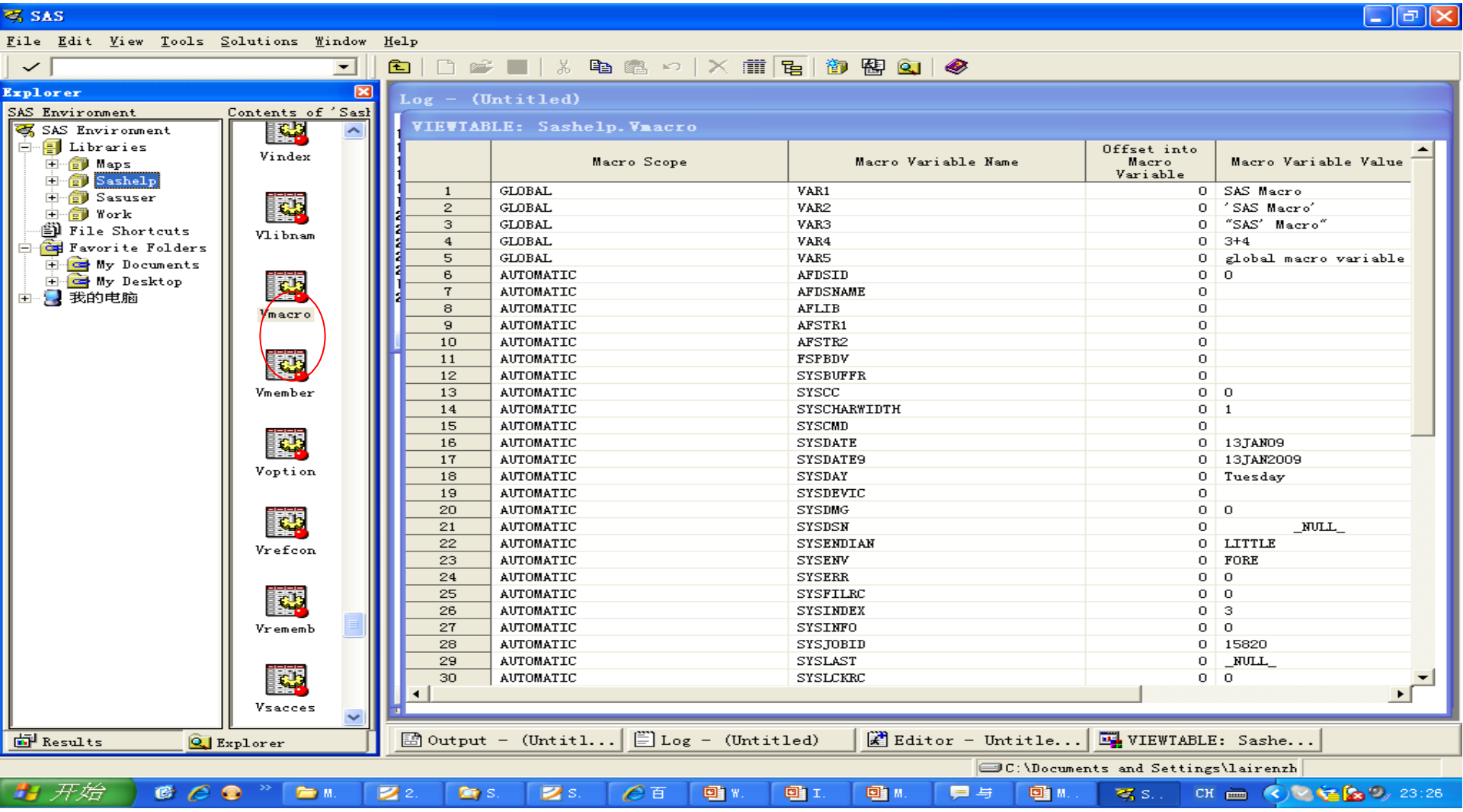
in open code then Global Else Local

%global or %local;

```
%let var5= global macro variable;
%macro test;
   %let var6= local macro variable;
%mend;
%test;
```

Macro Variable (4/6)

Where Macro variables store



	Macro Scope	Macro Variable Name	Offset into Macro Variable	Macro Variable Value
1	GLOBAL	VAR1	0	SAS Macro
2	GLOBAL	VAR2	0	'SAS Macro'
3	GLOBAL	VAR3	0	"SAS' Macro"
4	GLOBAL	VAR4	0	3+4
5	GLOBAL	VAR5	0	global macro variable
6	AUTOMATIC	AFDSID	0	0
7	AUTOMATIC	AFDSNAME	0	
8	AUTOMATIC	AFLIB	0	
9	AUTOMATIC	AFSTR1	0	
10	AUTOMATIC	AFSTR2	0	
11	AUTOMATIC	FSPBDV	0	
12	AUTOMATIC	SYSBUFFR	0	
13	AUTOMATIC	SYSCC	0	0
14	AUTOMATIC	SYSCHARWIDTH	0	1
15	AUTOMATIC	SYSCMD	0	
16	AUTOMATIC	SYSDATE	0	13JAN09
17	AUTOMATIC	SYSDATE9	0	13JAN2009
18	AUTOMATIC	SYSDAY	0	Tuesday
19	AUTOMATIC	SYSDEVIC	0	
20	AUTOMATIC	SYSDMG	0	0
21	AUTOMATIC	SYSDSN	0	_NULL_
22	AUTOMATIC	SYSENDIAN	0	LITTLE
23	AUTOMATIC	SYSENV	0	FORE
24	AUTOMATIC	SYSERR	0	0
25	AUTOMATIC	SYSFILRC	0	0
26	AUTOMATIC	SYSINDEX	0	3
27	AUTOMATIC	SYSINFO	0	0
28	AUTOMATIC	SYSJOBID	0	15820
29	AUTOMATIC	SYSLAST	0	_NULL_
30	AUTOMATIC	SYSLCKRC	0	0

Macro Variable (5/6)

How to display Macro variables

%put _all_: to display all macro variables;

%put _automatic_: to display all automatic macro variables;

%put _user_: to display all user-defined macro variables;

%put _global_ : to display all global variables (user-defined);

%put _local_: to display all local variables (user-defined);

Macro Variable (6/6)

Combine Macro variable with Text

Text&macvar

&macvar.Text

&macvar1&macvar2

```
%put &var1.Text;
%put Text&var1;
%put &var1&var2;
%put &var1..Text;
```

Results

```
SAS MacroText
TextSAS Macro
SAS Macro'SAS Macro‘
SAS Macro.Text
```

Macro Function (1/4)

- **Purpose**
- Used to manipulate character strings
- Used to perform arithmetic
- Used to execute SAS functions
- Have similar syntax to corresponding data step functions and yield similar results

Macro Function (2/4)

- Manipulate character strings
- %UPCASE translates letters to uppercase
- %SUBSTR produces a substring of a character string
- %SCAN extracts a word from a character string
- %LENGTH determines the length of a character string

```
%let var7=%length(&var1);

%put The length of var1 is &var7..;

Result in the Log: The length of var1 is 9.
```

Macro Function (2/4)

- Utilize the %SYSFUNC macro function

%SYSFUNC(*function(argument(s)) <,format>*)

  - The first argument is required.
  - The second argument is optional.
  - Function(argument(s)) is the name of one of most SAS functions and the corresponding arguments.
  - All SAS functions can be used with %SYSFUNC except: DIF, DIM, HBOUND, IORCMSG, INPUT, LAG, LBOUND, MISSING,PUT, RESOLVE, SYMGET , All Variable Information Functions.

Macro Function (3/4)

- Examples:
- **%put Today is %sysfunc(today(),date9.);**
  Result: Today is 14JUN2009
- Determining the Number of Variables and Observations in a Data Set

```
%macro obsnvars(ds);
%global dset nvars nobs;
%let dset=&ds;
%let dsid = %sysfunc(open(&dset));
%if &dsid %then
%do;
%let nobs =%sysfunc(attrn(&dsid,NOBS));
%let nvars=%sysfunc(attrn(&dsid,NVARS));
%let rc = %sysfunc(close(&dsid));
%put&dset has &nvars variable(s) and
&nobs observation(s).;
%end;
%else
%put Open for data set &dset failed -
%sysfunc(sysmsg());
%mend obsnvars;
```

%obsnvars(sasuser.houses);

```
Open for data set sasuser.houses
failed - ERROR: File
SASUSER.HOUSES.DATA does not
exist.
```

%obsnvars(sashelp.adomsg);

```
sashelp.adomsg has 6
variable(s) and 458
observation(s).
```

Macro Quoting (1/13)

- **Objective**
- Macro quoting functions tell the macro processor to interpret special characters and mnemonics as text rather than as part of the macro language. Here are some examples of the kinds of ambiguities:

  - Is `%sign` a call to the macro SIGN or a phrase "percent sign"?
  - Is OR the mnemonic Boolean operator or the abbreviation for Oregon?
  - Is the quote in O'Malley an unbalanced single quotation mark or just part of the name?
  - Is Boys&Girls a reference to the macro variable &GIRLS or a group of children?
  - Is GE the mnemonic for "greater than or equal" or is it short for General Electric?
  - Which statement does a semicolon end?
  - Does a comma separate parameters, or is it part of the value of one of the parameters?

Macro Quoting (2/13)

- Special characters and mnemonics
  - Special characters and mnemonics might require masking when they appear in text strings:

Special Characters	Must Be Masked
+ -*/<>=^|¬ ~ # LE LT EQ NE GE GT AND OR NOT IN	To prevent it from being treated as an operator in the argument of an %EVAL function
Blank	To maintain, rather than ignore, a leading, trailing, or isolated blank
;(semicolon)	To prevent a macro program statement from ending prematurely
,(comma)	To prevent it from indicating a new function argument, parameter, or parameter value
' " ( )	If it might be unmatched(expect pairs)
% or & immediately followed by a valid macro name	Text other than macro trigger

Macro Quoting (3/13)

- Macro Quoting Functions (1/3)

By Function

Function	Affects Groups	Works at
%STR	A, C*	macro compilation
%NRSTR	A, B, C*	macro compilation
%BQUOTE	A, C	macro execution
%NRBQUOTE	A, B, C	macro execution
%SUPERQ	A, B, C	macro execution (prevents resolution)
%QUOTE	A, C*	macro execution. Requires unmatched quotation marks and parentheses to be marked with a percent sign (%).
%NRQUOTE	A, B, C*	macro execution. Requires unmatched quotation marks and parentheses to be marked with a percent sign (%).

*Unmatched quotation marks and parentheses must be marked with a percent sign (%) when used with %STR, %NRSTR, %QUOTE, and %NRQUOTE.

Macro Quoting (4/13)

- Macro Quoting Functions (2/3)

Function	Action
%BQUOTE(*character-string | text expression*)	Mask special characters and mnemonic operators in a character string or the value of resolved text expression at macro execution. Compared to %QUOTE, %BQUOTE does not require that unmatched quotation marks or unmatched parentheses be marked with a preceding percent sign (%).
%NRBQUOTE(*character-string | text expression*)	Does the same as %BQUOTE and additionally masks ampersands (&) and percent signs (%).
%QUOTE(*character-string | text expression*)	Mask special characters and mnemonic operators in a character string or the value of resolved text expression at macro execution. Compared to %BQUOTE, %QUOTE requires that unmatched quotation marks and unmatched parentheses be marked with a preceding percent sign (%).
%NRQUOTE(*character-string | text expression*)	Does the same as %QUOTE and additionally masks ampersands (&) and percent signs (%).

Macro Quoting (5/13)

- Macro Quoting Functions (3/3)

%STR(*character-string*)	Mask special characters and mnemonic operators in constant text at macro compilation.
%NRSTR(*character-string*)	Does the same as %STR and additionally masks ampersands (&) and percent signs (%).
%SUPERQ(*macro-variable-name*)	Masks all special characters including ampersands (&) and percent signs (%) and mnemonic operators at macro execution and prevents further resolution of the value. Returns the value of a macro variable and does not resolve any macro references contained in that macro variable's value.
%UNQUOTE(*character-string* | *text expression*)	Unmasks all special characters and mnemonic operators in a value at macro execution.

Macro Quoting (7/13)

- **Compile-phase:** translates all statements inside a SAS macro definition into compiled instructions or constant text. it is completed with the **%mend** statement. During compilation, macro processor :

A. Creates an entry in the session catalog

B. Compiles and stores all macro program statements for that macro as macro instructions

C. Stores non-compiled items in the macro as text such as: 1)macro variable references, 2) text written by %put statements, 3) macro functions, 4) arithmetic and logical macro expression.

Macro Quoting (8/13)

```
%macro comp(score=);
 %if %upcase(&score)=A %then
 %do;
   data _null_;
     call symput('score', 'A');
   run;
   %put Your score is  "A";
 %end;
 %else
   %do;
     %put Your score is "&score", not "A";
   %end;
%mend;
```

%***COMP***(score=b)

Macro Quoting (9/13)

- **Execute-phase:** when the compiled macro is invoked, the macro facility executes or runs these instructions in another phase, the execute-phase. During execution, macro processor (repetitively):

A. Opens the session catalog and creates a local symbol table

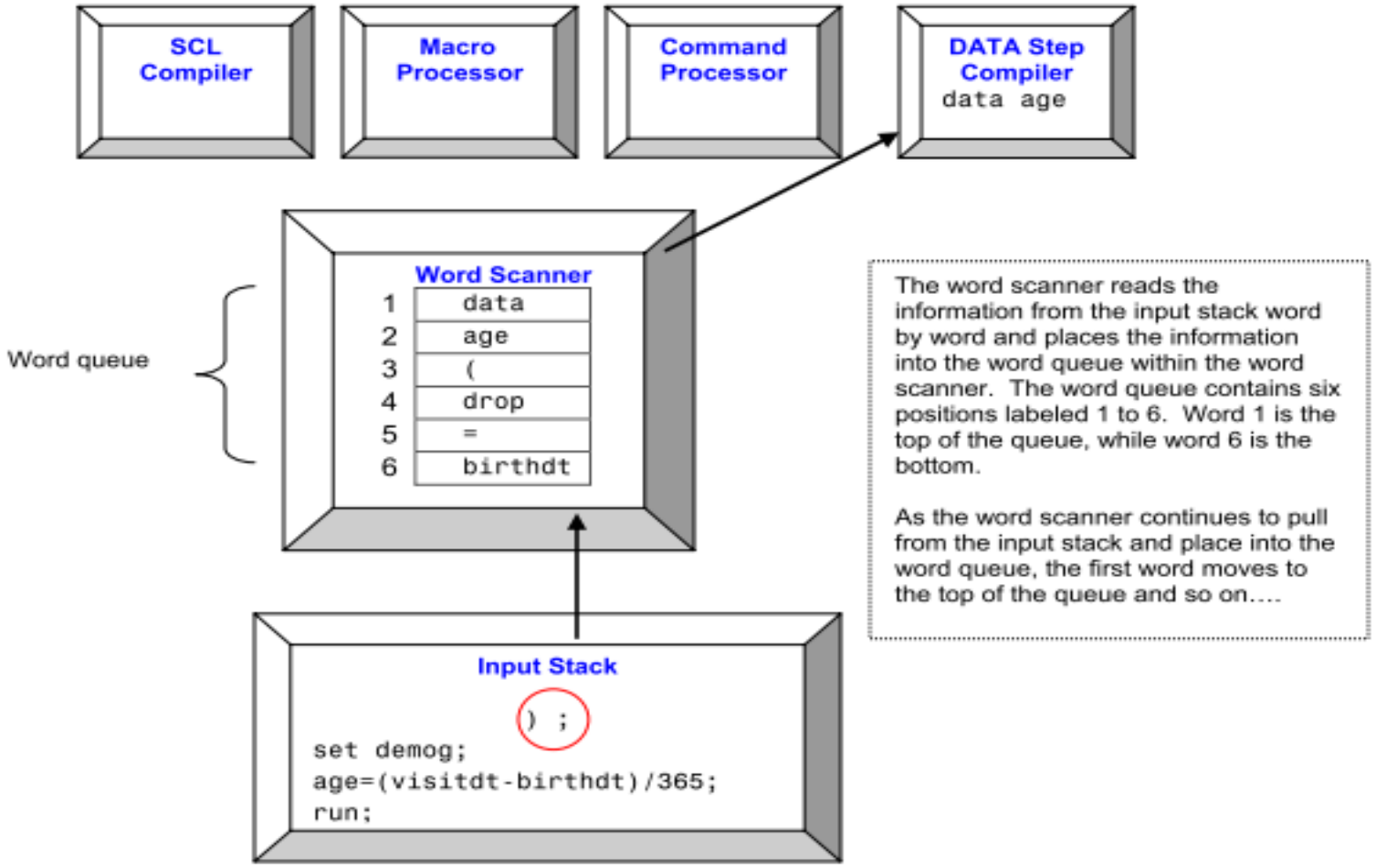
B. Removes tokens for macro call and places any parameter values in the local symbol table

C. Executes compiled macro program statements

D. Places non-compiled items back in to the top of the input stack as text

E. Waits for the word scanner to process the text

Macro Quoting (10/13)



The word scanner reads the information from the input stack word by word and places the information into the word queue within the word scanner. The word queue contains six positions labeled 1 to 6. Word 1 is the top of the queue, while word 6 is the bottom.

As the word scanner continues to pull from the input stack and place into the word queue, the first word moves to the top of the queue and so on….

Macro Quoting (11/13)

- **Example of masking**

```
%let x=%str(msd<=>Merck Sharp & Dohme);
```

- The value stored by SAS is:
- ▲msd▲▲▲Merck Sharp & Dohme▲

Macro Quoting (12/13)

- **Special Characters and Corresponding Quoting Functions (1/2)**

Special Characters	Quoted by Macro Quoting Functions	Select the preferable func?
+ -*/<>=^|¬ ~ # LE LT EQ NE GE GT AND OR NOT IN	%str, %nrstr, %quote, %nrquote, %bquote, %nrbquote, %superq	When do the chars need to be masked? Comp or Exec?
Blank ;(semicolon) ,(comma)	%str, %nrstr, %quote, %nrquote, %bquote, %nrbquote, %superq	When do the chars need to be masked? Comp or Exec?

Macro Quoting (13/13)

- **Special Characters and Corresponding Quoting Functions (2/2)**

Special Characters	Quoted by Macro Quoting Functions	Select the preferable func?
' " ( )	**%str, %nrstr, %quote, %nrquote,** %bquote, %nrbquote, %superq	Use a % sign as prefix when you use functions in RED color.
% or &	**%nrstr**, %nrquote, %nrbquote, %superq	%nrstr do a real masking job.

Macro Program (1/6)

- Define and Call a Macro
- A *macro* or *macro definition* enables you to write *macro programs*.

Define: **%MACRO macroname;**
macro text;
%MEND macroname;

Call: **%macroname;**

- *macro-name* follows SAS naming conventions
- *macro-text* can include
  - any text
  - SAS statements or steps
  - macro variables, functions, statements, or calls
  - any combination of the above.

Macro Program (2/6)

- **Type of Macro Parameter**
- Positional Parameter

```
%MACRO macroname(p1,p2,...,pn);
```

- Keyword Parameter

```
%MACRO macroname(k1=,k2=v2,...,kn=);
```

- Mixture

```
%MACRO macroname(p1,p2,...,k1=,k2=v2,...,pn);
```

Macro Program (3/6)

- **Macro Compilation**
- When a macro definition is submitted,
- macro language statements are
  - checked for syntax errors
  - Compiled
- SAS statements and other text are **not**
  - checked for syntax errors
  - Compiled
- the macro is stored as an entry in a SAS catalog, the temporary catalog **work.sasmacr** by default.
- The MCOMPILENOTE=ALL option issues a note to the SAS log after a macro definition has compiled.

General form of the MCOMPILENOTE= option:

```
OPTIONS MCOMPILENOTE=ALL | NONE;
```

The default setting is MCOMPILENOTE=NONE.

Macro Program (4/6)

- Produce a list of compiled macros stored in the default temporary catalog **work.sasmacr**.

```
proc catalog cat=work.sasmacr;
   contents;
   title "My Temporary Macros";
quit;
```

PROC CATALOG Output

```
                    My Temporary Macros

                Contents of Catalog WORK.SASMACR

# Name Type          Create Date       Modified Date Description
-----------------------------------------------------------------
1 TIME MACRO  11JUN2004:15:55:59  11JUN2004:15:55:59
```

Macro Program (5/6)

- **Calling a Macro**
- A *macro call*
  - causes the macro to execute
  - is specified by placing a percent sign before the name of the macro
  - can be made anywhere in a program (similar to a macro variable reference)
  - represents a macro trigger
  - is **not** a statement (no semicolon required).

General form of a macro call:

```
%macro-name
```

Macro Program (6/6)

- **Program Flow**
  - When the macro processor receives *%macro-name*, it
  - searches the designated SAS catalog (WORK.SASMACR by default) for an entry named *macro-name*.MACRO
  - executes compiled macro language statements
  - sends any remaining text to the input stack for word scanning
  - pauses while the word scanner tokenizes the inserted text and SAS code executes
  - resumes execution of macro language statements after the SAS code executes.

DATA Step and SQL Interfaces

- **Create a Macro Variable using Proc SQL**

```
proc sql noprint;
select name into :names separated by ',' from sashelp.class;
quit;
%put names=&names;
Log:names=Alfred,Alice,Barbara,Carol,Henry,James,Jane,Janet,Jeffre
y,John,Joyce,Judy,Louise,Mary,Philip,Robert,Ronald,Thomas,William
```

DATA Step and SQL Interfaces

- **SYMEXIST to check if a macro variable exists**

```
data _null_;
if symexist('var1') then put '**** var1 exists'; Else put "****
var1 doesn't exist";
if symexist('var11') then put '**** var11 exists';
Else put "**** var11 doesn't exist";
run; Results:
**** var1 exists
**** var11 doesn't exist
```

Other Funcitons: SYMLOCAL: SYMGLOBAL; SYMDEL; SYMGET

DATA Step and SQL Interfaces

- %eval and %sysvalf

%PUT Statement	Results in SAS log
`%put %eval(33 + 44);`	77
`%put %eval(33.2 + 44.1);`	ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was: 33.2 + 44.1
`%put %sysevalf(33.2 + 44.1);`	77.3
`%put %sysevalf(33.2 + 44.1,integer);`	77
`%let a=3;` `%let b=10;`	
`%put %eval(&b/&a);`	3
`%put %sysevalf(&b/&a);`	3.3333333333
`%put %sysevalf(&b/&a,ceil);`	4
`%put %sysevalf(&b/&a,boolean);`	1
`%let missvalu=.;` `%put %sysevalf(&b-&missvalu,boolean);`	NOTE: Missing values were generated as a result of performing an operation on missing values during %SYSEVALF expression evaluation. 0

Using Macros to Comment Out Code

When :

If there is a large section of codes you don't want to run.

How:

Put a "%macro name" at the top and a "%mend" at the end and never call the macro.

Advantages:

This effectively comments out the code without running it.

No need to worry about other comment styles in the code.

DATA Step and SQL Interfaces

- **RESOLVE Function**

```
%let name1=Sanofi;
%let name2=Aventis;
data _null_;
name=resolve('Our company is &name1 &name2'); put name;
run;


Result:
Our company is Sanofi Aventis
```

DATA Step and SQL Interfaces

- **CALL SYMPUT & PROC SQL (1/2)**

```
        %macro test; data test;
              do x=1 to 3;
   y='C'||strip(put(x,3.)); output;


  end;
 run;

data _null_;
   set test;
   call symput(y,put(x,z3.));
              run;

              data _null_;
             %do x=1 %to 3;
   %put The value of Macro Variable    C&x is &&C&x ;
%end; run;
%mend;
%test;
```

```
The value of Macro Variable C1 is 001
The value of Macro Variable C2 is 002
The value of Macro Variable C3 is 003
```

DATA Step and SQL Interfaces

CALL SYMPUT & PROC SQL (2/2)

```
%macro test;

proc sql noprint;
select put(x,z3.) into :D1-:D3
from test;
quit;
data _null_;                Macro Variable D&x is &&D&x ;
  %do x=1 %to 3;
     %put The value of
  %end;
 run;
  %mend
   ;
  %test
```

```
The ;value of Macro Variable D1 is 001
The value of Macro Variable D2 is 002
The value of Macro Variable D3 is 003
```

4
9

How to use SAS Macros

- In order to use a SAS macro
  - You have to tell SAS where it is located
  - You have to call it.

- E.g., We want to call the macro **%ANCOVA** (Performs an ANOVA or ANCOVA analysis and produces several output data sets), the SAS macro %ANCOVA is stored in the file ancova.sas.

- A SAS macro will have a particular name, and will be stored in a file of the same name.

How to use SAS Macros

- Telling SAS where the macro is located
  - You must know the location that all the files containing macros are in. (e.g. **C:\My Documents\SAS Macros**)
  - Place the following statements in the SAS file, e.g., autoexec.sas file: **filename storemacs 'C:\My Documents\SAS Macros\';**
  - Two ways to use the macro %ANCOVA that you developed
    - options mautosource sasautos=(storemacs, sasautos);
    - %include storemacs(ancova.sas)/source;
- then you can call the macro **%ANCOVA** whenever you need

Good Programming Practice (GPP) for SAS Macro Program

- SAS macros variables
  - When creating macro variables ensure that it does not have the same name as a system macro. (e.g., ABORT, SYSDAY)
  - Declare macro variables as local within the local environment.
    - E.g. %local macroname;

```
%macro macname;
    ...
    %local macroname;
    ...
%mend macname;
```

  - Declare all global macro variables in a common place near the beginning of the program as specified;
  - Declare macro variables explicitly as global or local using %GLOBAL or %LOCAL statements.

Good Programming Practice (GPP) for SAS Macro Program

- SAS macros program
  - Write macro procedures only for repetitive tasks;
  - Utilize existing standard macro procedures when possible;
  - Use SAS system options to facilitate debugging of macros (e.g. MPRINT, &sysmacroname);
  - Specify the name of the macro in the %mend statement;
  - Use the recommended comment statement within the macros. They will not appear in the SAS log if the MPRINT option is activated.
    - Beginning with /* and ending with */
    - Begins with a %* and ends with a ; (Preferred)

Good Programming Practice (GPP) for SAS Macro Program

- SAS macros program
  - Left-justify %MACRO and %MEND statements. Indent each level of subsequent statements consistently.
  - Align a %DO with its corresponding %END statement.
  - List each macro parameter on a separate line.
  - Start procedure statements or data step statements in column one if not located in a macro, else put %MACRO and %MEND statements in column one. All other code should be indented within the data step or procedure.
  - Add the header to the macro when you complete to develop the macro.

A Example for GPP

To Develop a Macro is To Build a Building

- You must have a big picture before you developing, the objects and clear logic
- Think more about the efficient process, make sure of using less parameters in your macro
- Fellow the GPP, make sure it's readable
- For the complicated macro, the manual for using it is needed

A good macro is like a beautiful building,

you are the architect