SAS Macro #3
Round #3

JC Wang

Interfaces Used During Data Step Execution

- **SYMPUT** routine assigns values to macro variables
- **SYMGET** function retrieves values of macro variables
Using SYMPUT Routine

CALL \texttt{SYMPUT}(name, value)

\begin{itemize}
  \item \textit{name}: name of the macro variable, if it does not exist, SAS creates it
  \item \textit{value}: value of the macro variable to be assigned.
\end{itemize}

Rules for Using SYMPUT

\begin{itemize}
  \item cannot use macro variable reference to retrieve a value in the same DATA step in which SYMPUT assigns that value
  \item to reference a value in a global statement following the DATA step (e.g., a TITLE statement), you must first force the DATA step to execute with a RUN statement
\end{itemize}

\texttt{see macro08.sas}
Specifying Macro Variable Name in SYMPUT

\textit{name} in SYMPUT can be any of

- valid SAS name enclosed in quotes
- DATA step variable name whose values are valid SAS name (see \texttt{macro06.sas} for examples)
- a character expression that ends up with a valid SAS name (see \texttt{macro08.sas} for examples)

Specifying Macro Variable Value in SYMPUT

\textit{value} in SYMPUT can be any of

- a string enclosed in quotes
- name of a DATA step variable, character or numeric; its value (automatic type conversion takes place for numeric variable) becomes the value of the macro variable
- a DATA step expression, its result becomes the macro variable’s value (automatic type conversion takes place for numeric expression)
- use of PUT function is usually recommended for the previous two cases
Using SYMGET Function

\textbf{SYMGET}\,(\texttt{argument}) \textbf{where} \texttt{argument} \textbf{can be:}

\begin{itemize}
  \item macro variable name enclosed in quotes (not a macro variable reference)
  \item a DATA step character variable name
  \item a character expression results in a macro variable name (again, not a macro variable reference)
\end{itemize}

\texttt{see macro08.sas}

Macro Quoting
Special Characters, revisited

Special characters may be misinterpreted by the macro processor when they appear in text strings:

\begin{itemize}
  \item blank, , ; " ' ( ) + - * / < > = ? ^ % &
  \item The following mnemonics also may be misinterpreted: AND OR NOT EQ NE LE LT GE GT
\end{itemize}
Quoting

To prevent the macro processor from misinterpreting, macro quoting functions are used to resolve ambiguities by masking the significance of these special characters and mnemonics:

- Semicolon: End of statement or part of text?
- Blank: Token separator or part of text?
- Quote: Literal delimiter or part of text?
- Ampersand: Macro trigger (% or &) or part of text?
- Comma: Argument separator or part of text?
- NE: Comparison operator or a text abbreviation?

Masking Tokens at Compile Time

The %STR function masks special characters and mnemonics during compilation so the macro processor does not interpret them as macro-level syntax. General form of the %STR function:

\[
\%\text{STR}(\text{character-string})
\]

where \text{character-string} can be any combination of text and macro triggers.
Tokens Masked by the %STR Function

- blank, ; + − * / < > = ? ^ #
- AND OR NOT EQ NE LE LT GE GT IN
- single quotes, double quotes, and parentheses when they appear in pairs
- Unmatched quotes and parentheses must be preceded by a percent sign (%)
- %STR does not mask macro triggers

Masking Macro Triggers

use of %NRSTR function

To mask the & and % characters and treat them as plain text, use the %NRSTR function.
Masking Tokens at Execution Time

- The `%BQUOTE function masks special characters and mnemonics during macro execution or during execution of a macro language statement in open code, and attempts to resolve all embedded macro triggers.
- `%SUPERQ is the only macro quoting function that prevents all resolution within the value of its argument.