THE POWER TO KNOW.

SAS® 9.3マクロ言語
リファレンス

SAS® ドキュメント

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® 9.3 Macro Language: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Macro Language: Reference

ISBN 978-1-60764-894-9

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

Printing 1, 2011 July
Electronic book 1, 2011 July

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

目次

このドキュメントについて

SAS 言語の構文の表記規則

概要

SAS 言語要素の構文は、標準的な表記規則を使用して文書化されます。標準的な表記規則を使用することで、読者は SAS 構文のコンポーネントを簡単に特定できるようになります。この表記規則は次に示す 3 つの部分に分けられます。

- 構文コンポーネント
- 書体に関する規則
- SAS ライブラリや外部ファイルへの参照

構文コンポーネント

ほとんどの言語要素の構文コンポーネントは、キーワードと引数を含みます。一部の言語要素では、キーワードのみが必要となります。また、一部の言語要素では、キーワードの後に等号(=)を付加する必要があります。

注: ほとんどの場合、SAS ドキュメントでは、コード例はモノスペースフォントを使って小文字で表記されます。ユーザーが実際に記述するコードでは、大文字、小文字、または大文字小文字を混在させて使用できます。

書体に関する規則

SAS 言語の構文を文書化する場合、太字の大文字、大文字、イタリック体の各書体に関して次のような規則が適用されます。

UPPERCASE BOLD (太字の大文字)
関数名やステートメント名などの SAS キーワードを表します。次の例では、ERROR というキーワードが太字の大文字で表記されています。

ERROR<message>;

UPPERCASE (大文字)
リテラルである引数を表します。次の例は、CMPMODEL=システムオプションの引数として、BOTH、CATALOG、XML の各リテラルのいずれかを指定できることを示しています。

CMPMODEL = BOTH | CATALOG | XML

italics (イタリック体)
ユーザーが指定する引数または値を表します。イタリック体で表記されている項目は、ユーザーが指定する値(非リテラル引数、または特定の引数に割り当てられる非リテラル値)を表します。

また、イタリック体で表記されている項目は、ユーザーが選択可能な引数リストの一般的な名前を表す場合もあります(*attribute-list* など)。複数の項目をイタリック体で表記する場合、*item-1, ..., item-n* のように表記します。

SAS ライブラリや外部ファイルへの参照

多くの SAS ステートメントやその他の言語要素は、SAS ライブラリや外部ファイルを参照します。このような参照を行う場合に、ユーザーは論理名(libref や fileref)を使用するか、それとも引用符で囲んだ物理ファイル名を使用するかを選択できます。論理名を使用する場合、通常、ユーザーは論理名と実際の名前との関連付けを行うのに、SAS ステートメント(LIBNAME または FILENAME)を使用するか、それともオペレーティングシステムが提供する制御言語を使用するかを選択できます。SAS ライブラリや外部ファイルを参照するにはいくつかの方法がありますが、どのような方法が使えるかはお使いのオペレーティングシステムによって異なります。

外部ファイルの使用例を表す場合、イタリック体の *file-specification* を使用します。SAS ライブラリの使用例を表す場合、イタリック体の *SAS-library* を使用します。*SAS-library* が引用符で囲まれていることに注意してください。

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

SAS 9.3 マクロ言語機能の新機能

概要

マクロ言語機能には、次のような拡張機能が追加されています。

- 新しい自動マクロ変数。これらの変数を使うことで、一般的なタスクの実行に必要となるテキストの量を削減できます。
- 新しいマクロ関数
- 新しいマクロステートメント
- 新しいマクロシステムオプション。これらのオプションを使うと、マクロの定義/再定義や、マクロの実行のより詳細な制御が可能になります。

自動マクロ変数

SYSADDRBITS
アドレスのビット数を含みます。

SYSENDIAN
現在のセッションのバイトオーダーを表す記号を含みます。取りうる値は LITTLE または BIG のいずれかです。

SYSNOBS
前のプロシジャまたは DATA ステップにより閉じられた最終データセットから読み取るオブザベーション数を含みます。

SYSODSESCAPECHAR
プログラム内の ODS ESCAPECHAR=の値を表示します。

SYSSIZEOFLONG
現在のセッションのロング整数の長さ(バイト)を含みます。

SYSSIZEOFPTR
ポインタのサイズ(バイト)を含みます。

SYSSIZEOFUNICODE
現在のセッションの Unicode 文字の長さ(バイト)を含みます。

マクロ関数

%SYSMACEXEC
マクロが現在実行中かどうかを示します。

%SYSMACEXIST
WORK.SASMACR カタログにマクロ定義があるかどうかを示します。

%SYSMEXECDEPTH
呼び出し点からのネストの深さを返します。

%SYSMEXECNAME
ネストレベルで実行しているマクロ名を返します。

マクロステートメント

%SYSMSTORECLEAR
コンパイル済みマクロを終了し、SASMSTORE=ライブラリをクリアします。

%SYSMACDELETE
WORK.SASMACR カタログからマクロ定義を削除します。

マクロシステムオプション

MAUTOCOMPLOC
自動呼び出しマクロのコンパイル時に、自動呼び出しマクロのソースの場所を SAS ログに表示します。

MAUTOLOCINDES
マクロプロセッサが自動呼び出しソースファイルのフルパス名を、WORK.SASMACR カタログのコンパイル済み自動呼び出しマクロ定義のカタログエントリの説明フィールドに追加するかどうかを指定します。

MCOVERAGE
カバレッジ分析データの生成を可能にします。

MCOVERAGELOC=
カバレッジ分析データファイルの場所を指定します。

推奨資料

- *Carpenter's Complete Guide to the SAS Macro Language*
- *Debugging SAS Programs: A Handbook of Tools and Techniques*
- *SAS Macro Programming Made Easy*
- *Base SAS プロシジャガイド*
- *SAS 言語リファレンス: 解説編*
- *SAS 関数と CALL ルーチン: リファレンス*

SAS の刊行物の総一覧については、support.sas.com/bookstore にてご確認ください。
必要な書籍についてのご質問は、下記までお寄せください。

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
電話: 1-800-727-3228
ファクシミリ: 1-919-677-8166
電子メール: sasbook@sas.com
Web アドレス: support.sas.com/bookstore

1 部

マクロ機能の説明と使い方

1 章

マクロ機能について

入門ガイド: マクロ機能

このドキュメントは、SAS のマクロ機能の言語リファレンスです。このドキュメントは、SAS マクロ言語プロセッサのリファレンスであり、SAS マクロ言語要素を定義します。このセクションでは、簡単な例を使用して SAS マクロ機能を説明します。

*マクロ機能*は、SAS を拡張し、カスタマイズするためのツールです。この機能によって、共通のタスクを実行するために入力する必要のあるテキスト量が減ります。マクロ機能を使用して、文字列または SAS プログラムステートメントのグループに名前を割り当てることができます。テキスト自体を操作する代わりに、作成した名前で操作することができます。

SAS マクロ言語は、ストリングベースの言語です。SAS マクロ言語では、16 進文字定数の使用はサポートされません。

注: SAS マクロ言語では、印刷できない文字を 16 進値を使用して指定することはサポートされません。

SAS プログラムまたはコマンドプロンプトでマクロ機能名を使用すると、マクロ機能は必要に応じて SAS ステートメントやコマンドを生成します。その後、SAS はそれらのステートメントを受け取り、通常の方法で入力したステートメントと同様に使用します。

マクロ機能には、次の 2 つのコンポーネントがあります。

マクロプロセッサ
SAS の一部として動作します。

マクロ言語
マクロプロセッサとの通信に使用される構文です。

SAS がプログラムテキストをコンパイルすると、次の 2 つの区切り文字によってマクロプロセッサの処理が起動されます。

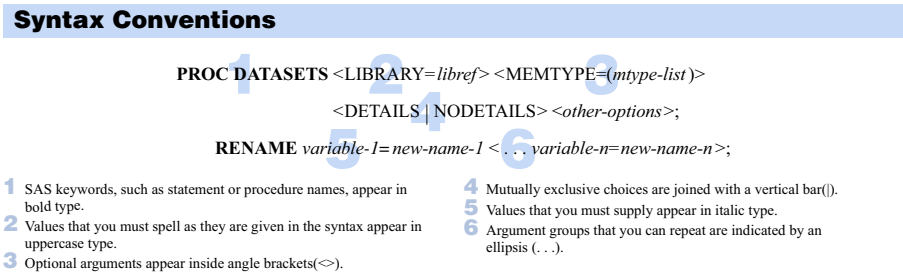
&名前
マクロ変数を参照します。“マクロ変数を使用した文字列の置換” (4 ページ)で、マクロ変数の作成方法が説明されています。*&名前*の形式は、マクロ変数参照と呼ばれています。

%名前
マクロを参照します。“マクロを使用した SAS コードの生成 ” (5 ページ) で、マクロの作成方法が説明されています。*%名前*の形式は、マクロ呼び出しと呼ばれています。

プログラムテキストがコンパイルされて実行される前に、マクロプロセッサによって生成されるテキスト置換が実行されます。マクロ機能は、DATA ステップで使用されるのに似たステートメントと関数を使用します。ただし、重要な違いは、マクロ言語要素がテキスト置換のみを可能にし、プログラムやコマンドの実行中には存在しないということです。

注: %で始まる 3 つの SAS ステートメントは、マクロ機能の一部ではありません。それらの要素は、次のドキュメントに含まれる%INCLUDE、%LIST、および%RUN ステートメントです :*SAS ステートメント: リファレンス*

次の図に、このドキュメントで使用される構文を示します。



マクロ変数を使用した文字列の置換

*マクロ変数*は、SAS コードの文字列を置換する場合に効果的です。マクロ変数を定義する最も簡単な方法は、%LET ステートメントを使用して、標準 SAS 命名規則に従ってマクロ変数に名前と値を割り当てることです。

```
%let city=New Orleans;
```

これで、**New Orleans** というテキストを表示したい SAS ステートメントで、マクロ変数 CITY を使用できます。次の TITLE ステートメントに示すように、変数名の前にアンパサンド(&)を付けてこのマクロ変数を参照します。

```
title "Data for &city";
```

マクロプロセッサは、マクロ変数 CITY への参照を次のように置換します。

```
title "Data for New Orleans";
```

マクロ変数は、マクロ定義内またはマクロ定義の外側のステートメント内(*オープンコード*と呼ばれる)で定義できます。

注: タイトルは、二重引用符で囲みます。オープンコード内の引用符で囲まれた文字列の内、マクロプロセッサは二重引用符で囲まれたマクロ変数参照を置換しますが、一重引用符で囲まれたマクロ変数参照を置換しません。

オープンコード内(マクロ定義の外側)で%LET ステートメントを記述すると、グローバルマクロ変数が作成され、その変数が作成された SAS セッションが実行されている間、SAS コード内の(DATALINES ステートメント、CARDS ステートメント以外の)任意の場所で使用できます。*ローカル*マクロ変数も用意されています。それらは、それらが作成されたマクロ定義の内部でのみ使用できます。グローバルマクロ変数とローカルマクロ変数の詳細については、マクロ変数のスコープ (45 ページ)を参照してください。

マクロ変数には、SAS データセット変数と同じ長さ制限はありません。ただし、マクロ変数に割り当てる値に特定の特殊文字(たとえば、セミコロン、引用符、アンパサンド、パーセント記号)またはニーモニック(たとえば、AND、OR、LT)が含まれている場合、マクロクォーティング関数を使用して特殊文字をマスクする必要があります。そうしない場合、特殊文字やニーモニックが、マクロプロセッサによって誤って解釈される恐れがあります。詳細については、マクロクォーティング (82 ページ)を参照してください。

マクロ変数は、単純なテキスト置換には役立ちますが、条件付き演算、DO ループなどの複雑なタスクを実行できません。そのような処理を実行する場合は、マクロを定義する必要があります。

マクロを使用した SAS コードの生成

マクロの定義

プログラムでマクロを使用すると、テキストの置換に加えて、他の多くのことを実行できます。SAS プログラムに任意の個数のマクロを含めて、1 つのプログラム内で特定のマクロを何度も呼び出すことができます。

独自マクロの定義方法を学習するために、このセクションには、後で独自マクロのモデル化に使用できる例がいくつか含まれています。それぞれの例は極めて単純ですが、さまざまな方法を組み合わせることによって、複雑なタスクを実行できる高度で柔軟なマクロを作成できます。

定義するマクロには、固有の名前を付けます。マクロの名前を選択する場合、SAS 言語のキーワードやコールルーチン名と同じ名前を避けることをお勧めします。選択する名前は、標準 SAS 命名規則に従います。SAS 命名規則の詳細については、Base SAS 言語ドキュメントを参照してください。マクロ定義は、次の例のように、%MACRO ステートメントと%MEND (マクロの終了)ステートメントの間に配置します。

%MACRO *macro-name*;

%MEND *macro-name*;

%MEND ステートメントに指定する *macro-name* は、%MACRO ステートメントに指定した *macro-name* と一致している必要があります。

注: %MEND ステートメントでの *macro-name* の指定は必須ではありませんが、推奨されます。そうすることで、デバッグ中に%MACRO ステートメントと%MEND ステートメントを対応付けることが容易になります。

単純なマクロ定義の例を次に示します。

```
%macro dsn;
```

```
Newdata
%mend dsn;
```

このマクロの名前は DSN です。**Newdata** は、マクロのテキストです。マクロの内側の文字列は、*定数テキスト*または*モデルテキスト*と呼ばれます。これは、この文字列が、SAS プログラムの一部になるテキストとして使用される、モデルまたはパターンであるためです。

マクロを呼び出す(または*起動する*)には、次のようにマクロ名の先頭にパーセント記号(%)を付けます。

%macro-name

マクロの呼び出しは SAS ステートメントに似ていますが、末尾にセミコロンを付ける必要はありません。

例として、DSN マクロの呼び出し方法を次に示します。

```
title "Display of Data Set %dsn";
```

マクロプロセッサは DSN マクロを実行し、マクロの定数テキストを TITLE ステートメントに代入します。

```
title "Display of Data Set Newdata";
```

注: タイトルは、二重引用符で囲みます。オープンコード内の引用符で囲まれた文字列の内、マクロプロセッサは二重引用符で囲まれたマクロの呼び出しを置換しますが、一重引用符で囲まれたマクロの呼び出しを置換しません。

DSN マクロは、次のようにコーディングした場合と全く同じです。

```
%let dsn=Newdata;

title "Display of Data Set &dsn";
```

この結果のコードを次に示します。

```
title "Display of Data Set Newdata";
```

つまりこの場合は、マクロを使用しても、マクロ変数を使用する場合と比べてメリットはありません。ただし、DSN は極めて単純なマクロです。この後の例で示すように、マクロは、DSN マクロよりも非常に多くのことを実行できます。

マクロにコメントを挿入する

すべてのコードはコメント機能を完全に利用できます。マクロコードも例外ではありません。マクロコードへのコメントの追加に使用できる 2 種類の形式が用意されています。

1 つ目の形式は SAS コードのコメントと同じく、**/*** で始まり、***/** で終わります。2 つ目の形式は、**%*** で始まり、**;** で終わります。次のプログラムでは、両方の形式のコメントを使用しています。

```
%macro comment;
/* Here is the type of comment used in other SAS code. */
%let myvar=abc;

%* Here is a macro-type comment.;
%let myvar2=xyz;

%mend comment;
```

マクロコード内では、好きな方の形式のコメントを使用できます。前の例のように、両方を使用することもできます。

SAS コードで使用されるアスタリスクスタイルのコメント(* *コメント記述* ;)をマクロ定義内で使用することはお勧めしません。アスタリスクスタイルは、定数テキストを正しくコメント化しますが、コメントに含まれるマクロステートメントはすべて実行されます。コメントテキスト内に含まれる一致しない引用符が無視されず、それによって予測できない結果を招く恐れがあるため、この形式のコメントは推奨されません。

複数の SAS ステートメントを含むマクロ定義

次のように、SAS プログラムの全体を含むマクロを作成できます。

```
%macro plot;
proc plot;
plot income*age;
run;
%mend plot;
```

その後のプログラムでは、次のようにマクロを呼び出せます。

```
data temp;
set in.permdata;
if age>=20;
run;

%plot

proc print;
run;
```

これらのステートメントを実行すると、次のプログラムが生成されます。

```
data temp;
set in.permdata;
if age>=20;
run;

proc plot;
plot income*age;
run;

proc print;
run;
```

マクロにパラメータを使用して情報を渡す

%MACRO ステートメントで、かっこ内に定義されるマクロ変数は、*マクロパラメータ*です。マクロパラメータによって、マクロに情報を渡すことができます。次に簡単な例を示します。

```
%macro plot(yvar= ,xvar= );
proc plot;
plot &yvar*&xvar;
run;
%mend plot;
```

次のように、パラメータに値を指定してマクロを呼び出します。

```
%plot(yvar=income,xvar=age)

%plot(yvar=income,xvar=yrs_educ)
```

マクロを実行すると、マクロプロセッサは、マクロ呼び出しで指定された値をマクロ定義のパラメータに対応付けます。(このタイプのパラメータは、*キーワードパラメータ*と呼ばれます。)

マクロの実行によって、次のコードが生成されます。

```
proc plot;
plot income*age;
run;

proc plot;
plot income*yrs_educ;
run;
```

パラメータの使用には、いくつかのメリットがあります。まず、%LET ステートメントの記述を減らせます。次に、パラメータを使用すると、マクロの外部のプログラムに変数が影響を与えずに済みます。マクロパラメータは、*ローカル*マクロ変数の一例です。マクロパラメータが存在するのは、それが定義されたマクロが実行されている間だけです。

SAS コードの条件付き生成

%IF-%THEN-%ELSE マクロステートメントを使用することで、マクロによって条件付きで SAS コードを生成できます。次の例を参照してください。

```
%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;
```

この例では、WHATSTEP マクロが、デフォルトで null 値に設定されるキーワードパラメータを使用しています。キーワードパラメータを使用するマクロを呼び出す場合、パラメータ名の後ろに等号を付け、その後にパラメータに割り当てる値を加えて指定します。ここでは、WHATSTEP マクロを、INFO に **print** を設定し、MYDATA に **grocery** を設定して呼び出しています。

```
%whatstep(info=print,mydata=grocery)
```

このコードによって次のステートメントが生成されます。

```
proc print data=grocery;
run;
```

マクロプロセッサでは、大文字と小文字の値は区別されます。そのため、前述の例で **print** の代わりに **PRINT** を指定すると、プログラムは動作しません。マクロをさらに堅牢にするには、%UPCASE マクロ関数を使用します。詳細については、“%UPCASE 関数と%QUPCASE 関数” (280 ページ)を参照してください。

詳細については、“%MACRO ステートメント” (309 ページ) と“%MEND ステートメント” (316 ページ)を参照してください。

高度なマクロの使い方

%DO ループを使用したテキストの反復部分の生成

“SAS コードの条件付き生成” (8 ページ)は、複数の SAS ステートメントを条件付きで実行するための、%DO-%END で囲まれたステートメントグループを提供しています。テキストの反復部分を生成するには、%DO ループによる反復を使用します。たとえば、次のマクロ(NAMES)では、%DO ループによる反復を使用して、DATA ステートメントで使用される一連の名前を作成しています。

```
%macro names(name= ,number= );
%do n=1 %to &number;
&name&n
%end;
%mend names;
```

NAMES マクロは、NAME パラメータの値とマクロ変数 N の値を連結して、一連の名前を作成します。N のストップ値は、次の DATA ステートメントに示すように、NUMBER パラメータの値で指定します。

```
data %names(name=dsn,number=5);
```

このステートメントをサブミットすると、次のような完全な DATA ステートメントが生成されます。

```
data dsn1 dsn2 dsn3 dsn4 dsn5;
```

注: %DO %WHILE ステートメントや%DO %UNTIL ステートメントを使用して、条件付きで%DO ループを実行することもできます。詳細については、“%DO %WHILE ステートメント” (298 ページ)と“%DO %UNTIL ステートメント” (297 ページ)を参照してください。

マクロ変数参照の接尾語の生成

一連の番号付きの名前を生成するときに、接頭語と番号の間に必ず文字 X を挿入したいとします。次の NAMESX マクロは、指定した接頭語の後に X を挿入します。

```
%macro namesx(name=,number=);
%do n=1 %to &number;
&name.x&n
%end;
%mend namesx;
```

&NAME 参照の末尾にあるピリオドは、区切り文字です。マクロプロセッサは、この区切り文字によって、後ろに文字 X が付いた&NAME 参照と&NAMEX 参照を区別します。DATA ステートメントで NAMESX マクロを呼び出す例を次に示します。

```
data %namesx(name=dsn,number=3);
```

このステートメントをサブミットすると、次のステートメントが生成されます。

```
data dsnx1 dsnx2 dsnx3;
```

マクロ変数参照で区切り文字としてピリオドを使用する場合の詳細については、マクロ変数 (21 ページ)を参照してください。

マクロ言語のその他の機能

以降のセクションでマクロ言語のさまざまな要素についてさらに詳細に説明しますが、このセクションでは、一部の機能を紹介し、詳細情報へのリンクを示します。

マクロステートメント
: このセクションでは、%MACRO や%IF-%THEN など、いくつかのマクロステートメントのみを説明しています。この他にも多くのマクロステートメントが存在します。一部のマクロステートメントはオープンコード内で有効ですが、それ以外はマクロ定義内でのみ有効です。マクロステートメントの完全な一覧については、“マクロステートメント ” (158 ページ)を参照してください。

マクロ関数
: マクロ関数とは、マクロ機能によって定義された関数のことです。これらは、1 つ以上の引数を処理して結果を生成します。たとえば、%SUBSTR 関数は他の文字列から部分文字列を作成し、%UPCASE 関数は文字を大文字に変換します。マクロ関数の特殊なカテゴリであるマクロクォーティング関数は、特殊文字がマクロプロセッサによって誤って解釈されないようにするために、それらをマスクします。

  2 つの特殊なマクロ関数、%SYSFUNC および%QSYSFUNC が用意されています。これらは、SAS 言語関数や、ユーザーが SAS/TOOLKIT を使用して作成した関数へのアクセスを提供します。%SYSFUNC と%QSYSFUNC を Base SAS ソフトウェアの新しい関数に対して使用して、SAS のホスト、ベース、グラフィックなどのオプションの値を取得できます。これらの関数を使用して、SAS データセット、テストデータセットの属性を開いたり閉じたりすることや、外部ファイルの読み取り/書き込みを行うこともできます。この他、マクロで浮動小数点演算を実行できる%SYSEVALF という特殊な関数があります。

  マクロ関数の一覧については、次を参照してください。“マクロ関数 ” (160 ページ). マクロクォーティング関数の説明については、マクロクォーティング (82 ページ)を参照してください。選択した Base SAS 関数を%SYSFUNC を使用して呼び出す構文については、%SYSFUNC 関数で使用する関数の構文 (373 ページ)を参照してください。

自動呼び出しマクロ
: 自動呼び出しマクロは、SAS によって定義され、共通のタスクを実行するマクロです。たとえば、マクロ変数の値の先頭または末尾から空白を除去したり、値のデータタイプを返したりします。自動呼び出しマクロの一覧については、“SAS が提供する自動呼び出しマクロ” (170 ページ)を参照してください。

自動マクロ変数
: 自動マクロ変数は、マクロプロセッサによって作成されるマクロ変数です。たとえば、SYSDATE には、SAS が呼び出された日付が格納されます。自動マクロ変数の一覧については、“マクロ言語要素” (157 ページ)を参照してください。

マクロ機能インターフェイス
: マクロ機能インターフェイスは、マクロ機能と SAS の他の部分(DATA ステップ、SCL コード、SQL プロシジャ、SAS/CONNECT ソフトウェアなど)との間の動的な接続を提供します。 たとえば、CALL SYMPUT を使用して DATA ステップ内の値に基づいてマクロ変数と作成することや、リモートホストに格納されたマクロ変数の値を%SYSRPUT マクロステートメントを使用して取り出すことができます。これらのイ

ンターフェイスの詳細については、マクロ機能とのインターフェイス (103 ページ)を参照してください。

2 章

SAS プログラムとマクロ処理

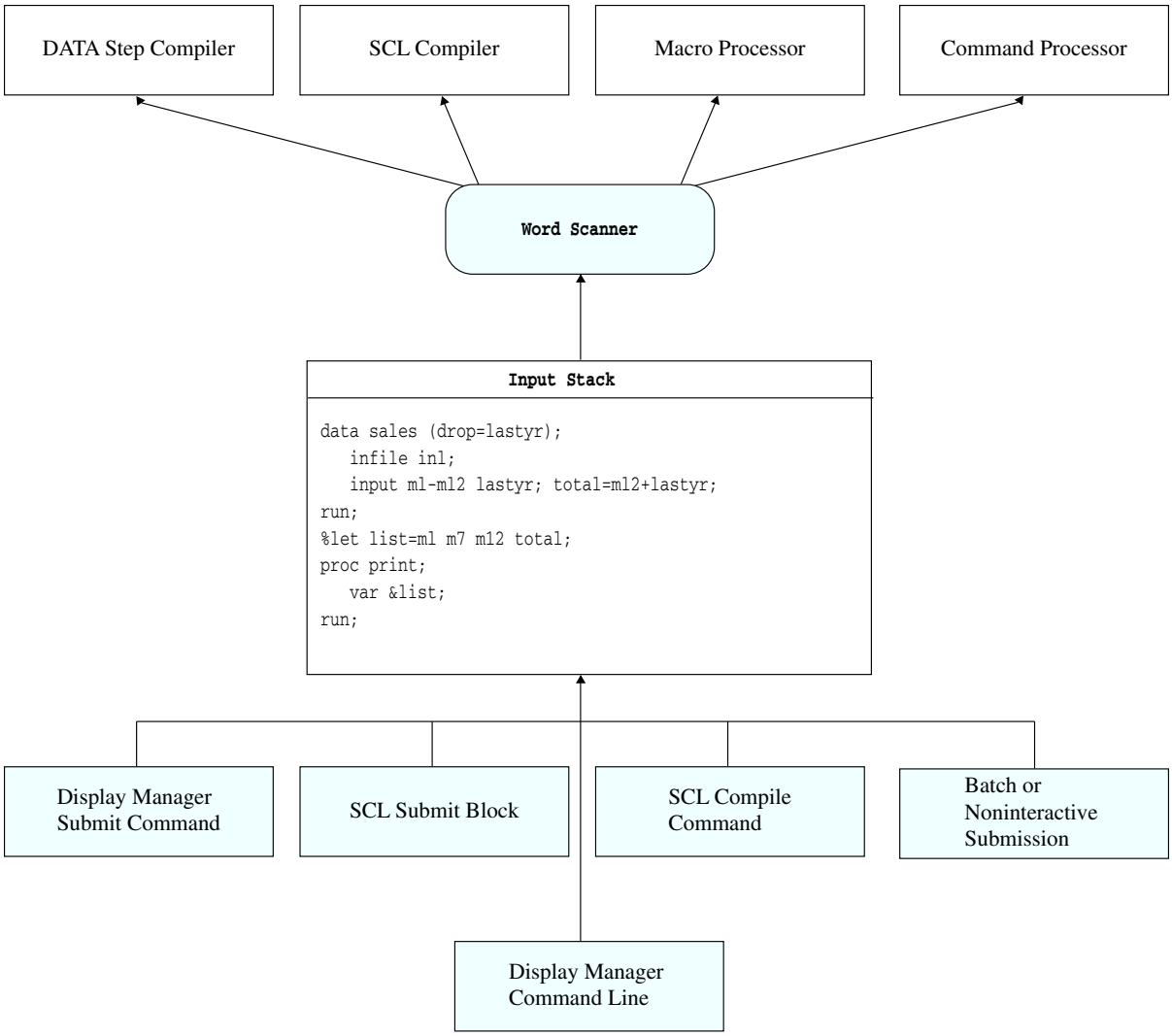
SAS プログラムとマクロ処理

このセクションでは、SAS がプログラムを処理する際に従う典型的なパターンについて説明します。これらの概念は、マクロプロセッサが SAS の他の部分とどのように連携しているかを理解するにの役立ちます。ただしこれらは、ほとんどのマクロのプログラミングでは必要ありません。プログラムの背後で何が実行されているかを理解できるようにするために、これらの概念について説明します。

注: このセクションに含まれる概念は、SAS ソフトウェアの動作について、詳細な具体的表現ではなく、論理的表現で示されます。

プログラムをサブミットすると、プログラムは*入力スタック*と呼ばれるメモリ領域に格納されます。このことは、SAS ウィンドウ環境、SCL SUBMIT ブロック、SCL COMPILE コマンド、バッチセッション、非対話型セッションなどの、プログラムとコマンドのすべてのソースに当てはまります。次の図に示す入力スタックには、販売データを表示する単純な SAS プログラムが格納されています。プログラムの最初の行が、入力スタックの先頭にあります。

図 2.1 サブミットされたプログラムの入力スタックへの送信



プログラムが入力スタックに到着すると、SAS は、文字のストリームを個々のトークンに変換します。これらのトークンは、DATA ステップコンパイラやマクロプロセッサなどの、SAS の他の処理部に転送されます。SAS がどのようトークンを認識して SAS の他の部分に転送しているかを知ることは、SAS のさまざまな部分とマクロプロセッサがどのように連携しているかを理解するのに役立ちます。プログラム内でのマクロ実行タイミングの制御方法についての理解にも役立ちます。次のセクションでは、単純なプログラムがどのようにトークン化され、処理されるかを示します。

マクロ処理を使用しないステートメントの処理方法

入力スタックからワードとシンボルを抽出するために SAS が使用するプロセスを、*トークン化*と呼びます。トークン化は、*ワードスキャナ*と呼ばれる SAS のコンポーネントによって実行されます。その説明は、図 2.2 (15 ページ)に示します。ワードスキャナは、入力スタックの先頭の文字から開始して、各文字を順番に調べます。ワードスキャナは、このようにして文字からトークンを組み立てます。一般的なトークンは、次の 4 種類です。

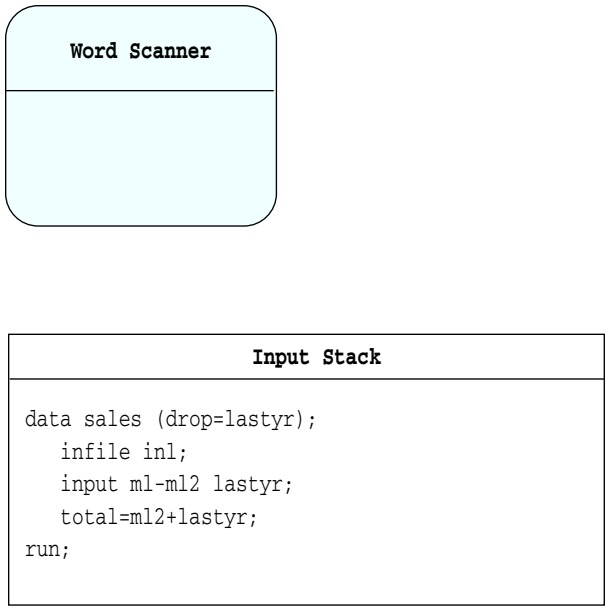
リテラル
　引用符で囲まれた文字列。

数値
　10 進数、日付値、時間値、および 16 進数。

名前
　アンダースコアまたは文字で始まる文字列。

特殊
　SAS で特殊な意味を持つ文字または文字のグループ。特殊文字の例としては、*** / + - ** ; $ ( ) . & % =などがあります。**

図 2.2 *トークン化前のサンプルプログラム*



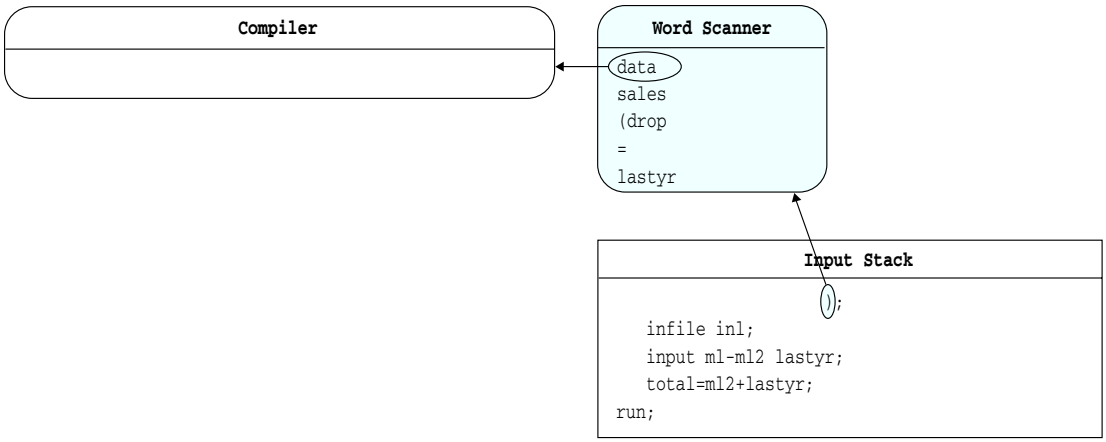
前の図では、入力スタック内の最初の SAS ステートメントには、8 つのトークン(4 つの名前と 4 つの特殊文字)が含まれています。

```
data sales(drop=lastyr);
```

ワードスキャナは、空白または新しいトークンの先頭を検出すると、そのトークンを入力スタックから削除して、キューの最後尾に転送します。

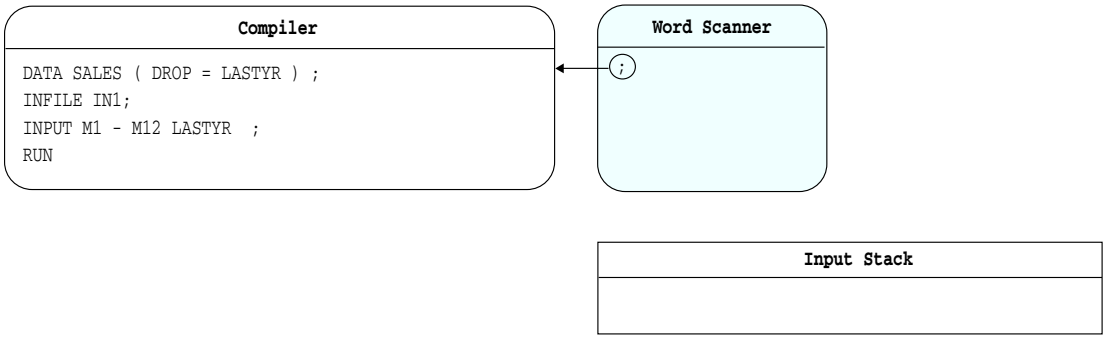
この例では、ワードスキャナは、入力スタックから最初のトークンを取り出すと、そのトークンを DATA ステップの開始として認識します。ワードスキャナによって DATA ステップコンパイラが起動され、トークンの要求を開始します。このコンパイラは、次の図に示すように、キューの先頭からトークンを取り出します。

図 2.3 ワードスキャナによるトークンの取得



コンパイラは、DATA ステップの終了(この場合、RUN ステートメント)を認識するまで、トークンを取り出し続けます。DATA ステップの終了は、DATA ステップの境界とも呼ばれます。これを次の図に示します。DATA ステップコンパイラが DATA ステップの終了を認識すると、DATA ステップが実行されて完了します。

図 2.4 ワードスキャナによるコンパイラへのトークンの送信

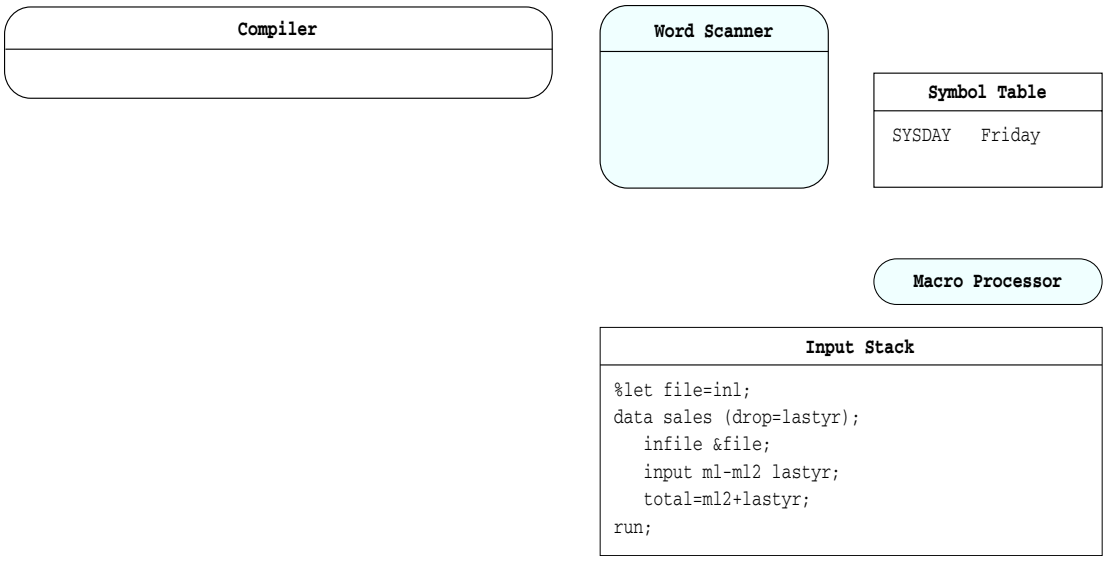


マクロプロセッサ処理を使用しないほとんどの SAS プログラムでは、コンパイラが受信するすべての情報は、サブミットされたプログラムからもたらされます。

マクロ処理を使用したステートメントの処理方法

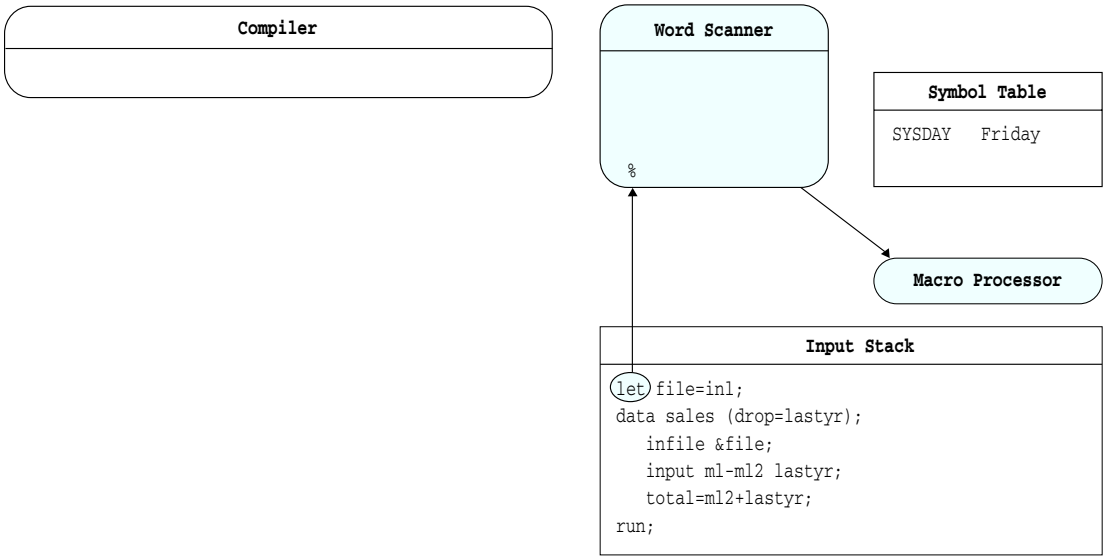
マクロ処理を使用するプログラムでは、マクロプロセッサは、入力スタックに配置されてワードスキャナによってトークン化されるテキストを生成できます。このセクションの例では、マクロプロセッサがどのようにマクロ変数を作成して置換するかを示します。コンパイラとマクロプロセッサがどのように連携するかを説明するために、次の図には、マクロプロセッサおよびマクロ変数シンボルテーブルを示しています。SAS は、自動マクロ変数とグローバルマクロ変数の値を保持するために、SAS セッションの開始時にシンボルテーブルを作成します。SAS は、SAS セッションの開始時に自動マクロ変数を作成します。説明の目的で、シンボルテーブルには、1 つの自動マクロ変数(SYSDAY)のみを示しています。

図 2.5 マクロプロセッサとシンボルテーブル



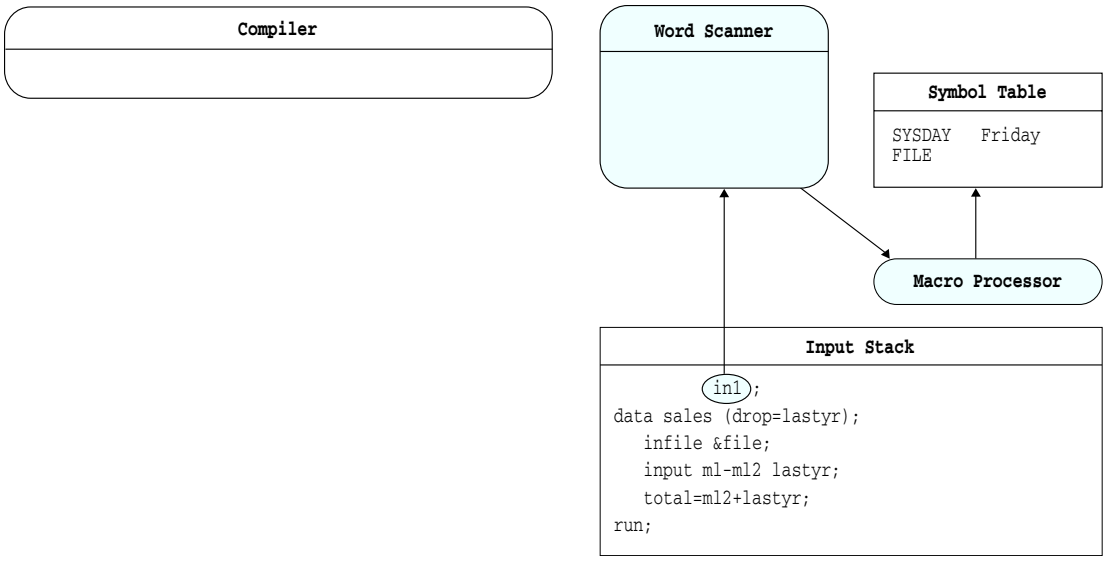
ワードスキャナは、マクロのトリガを検出すると、その情報をマクロプロセッサに送信します。マクロのトリガは、アンパサンド(&)またはパーセント記号(%)の後に空白以外の文字を続けて表されます。前述の例と同様に、ワードスキャナは、入力スタックの先頭の文字を調べることで、このプログラムの処理を開始します。この場合、ワードスキャナは、パーセント記号(%)の後に空白以外の文字が続いているのを検出します。ワードスキャナは、これらの文字の組み合わせについてマクロ言語要素の可能性があると認識し、マクロプロセッサを起動して%と LET を調べます。これを次の図に示します。

図 2.6 マクロプロセッサによる LET の検査



マクロプロセッサは、マクロ言語要素を認識すると、ワードスキャナとの連携を開始します。この場合、マクロプロセッサは、%LET ステートメントを削除して、シンボルテーブルにエントリを書き込みます。これを次の図に示します。

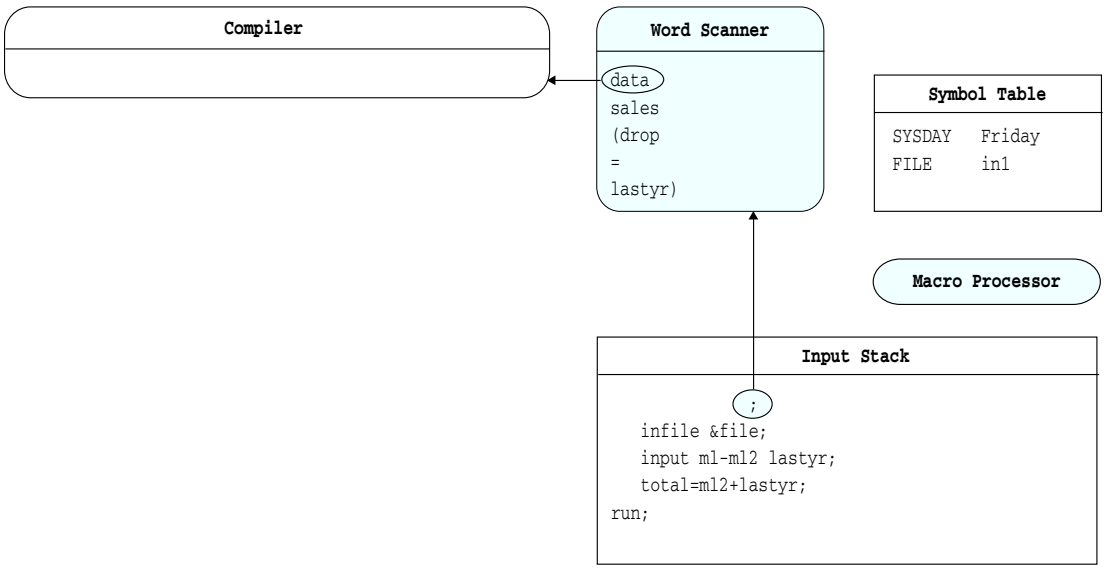
図 2.7 マクロプロセッサによるシンボルテーブルへの書き込み



ワードスキャナによってマクロプロセッサが起動されてから、マクロプロセッサのアクションが完了するまで、すべての処理はマクロプロセッサによって制御されます。マクロプロセッサが実行されている間は、ワードスキャナにも DATA ステップコンパイラにも処理は発生しません。

マクロプロセッサが終了すると、ワードスキャナは次のトークン(この例では、DATA キーワード)を読み込んで、それをコンパイラに送信します。コンパイラがワードスキャナによって起動され、キューの先頭からトークンを取り出し始めます。これを次の図に示します。

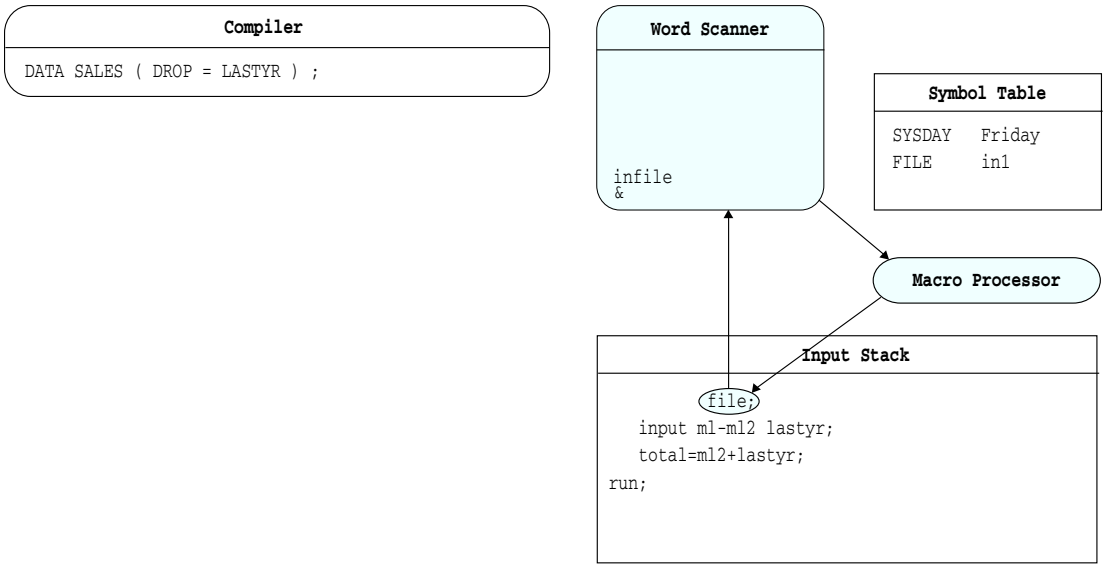
図 2.8 ワードスキャナによるトークン化の再開



各トークンが処理される際に、SAS は、特殊文字とニーモニック演算子をマスクするためにマクロクォーティング関数が提供する保護を削除します。詳細については、“マクロクォーティング” (82 ページ)を参照してください。

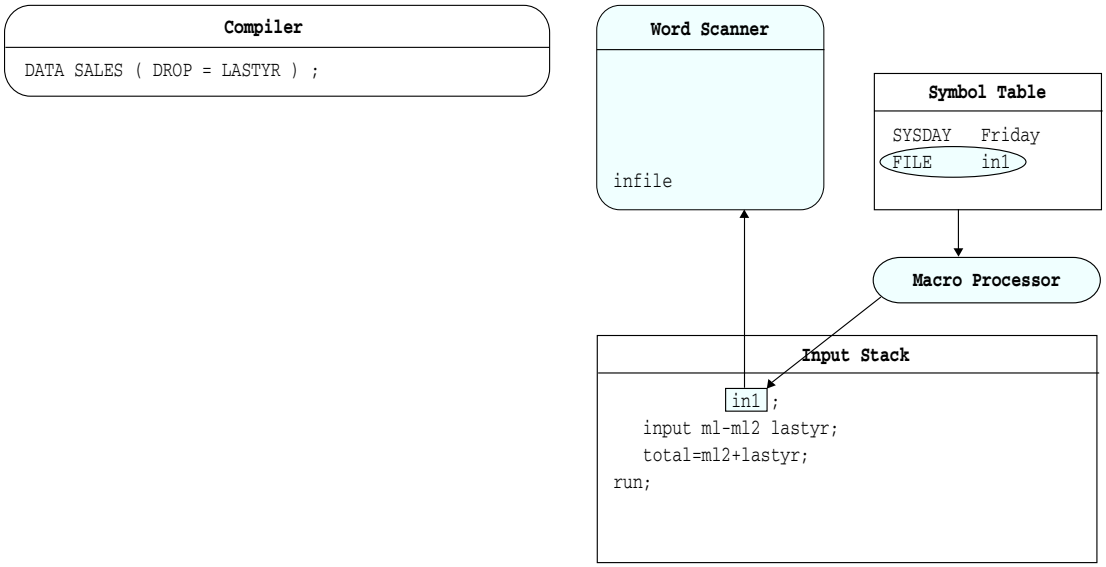
ワードスキャナは、アンパサンドの後に空白以外の文字が続くトークンを検出すると、マクロプロセッサを起動して次のトークンを調べます。これを次の図に示します。

図 2.9 *マクロプロセッサによる&FILE の検査*



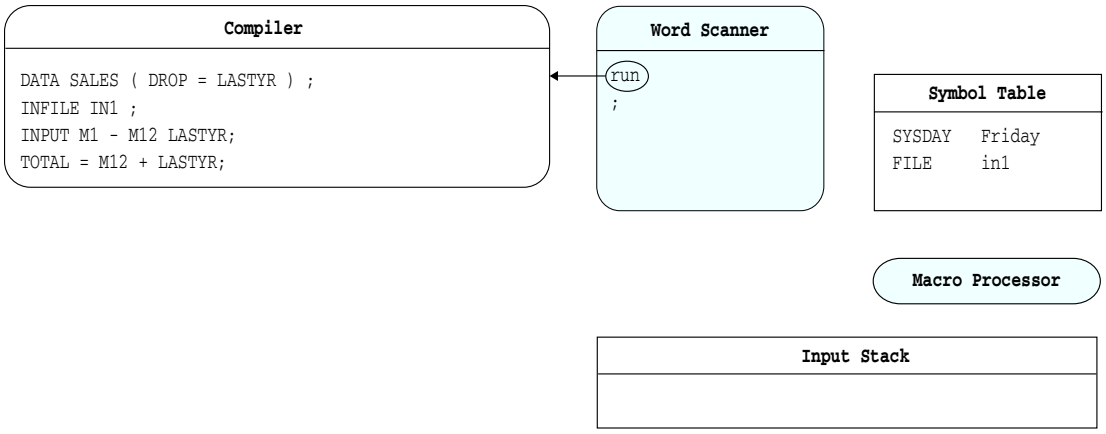
マクロプロセッサは、トークンを調べて、シンボルテーブルに存在するマクロ変数を認識します。マクロプロセッサは、入力スタックからマクロ変数名を削除して、それをシンボルテーブルのテキストで置き換えます。これを次の図に示します。

図 2.10 *マクロプロセッサによる入力スタックでのテキストの生成*



入力スタックがすべて読み込まれるまで、引き続きコンパイラはトークンを要求し、ワードスキャナはそれらを提供します。これを次の図に示します。

図 2.11 ワードスキャナの処理の完了



この例のように、入力スタックの末尾が DATA ステップの境界である場合、DATA ステップがコンパイラによってコンパイルされて実行されます。その後、SAS が DATA ステップのタスクを解放します。プログラムの実行中に作成されたマクロ変数は、すべてシンボルテーブルに残ります。入力スタックの末尾が DATA ステップの境界でない場合、処理されたステートメントはコンパイラ内に残ります。さらにステートメントがサブミットされて入力スタックに送信されると、処理が再開します。

3 章

マクロ変数

マクロ変数

マクロ変数は、シンボリック置換によって SAS プログラム内のテキストを動的に変更可能にするツールです。大量または少量のテキストをマクロ変数に割り当てることができます。その後、テキストが格納された変数を参照するだけで、そのテキストを使用できます。

マクロ変数値の最大長は、65,534 文字です。マクロ変数の長さは、特定の長さの宣言によってではなく、割り当てられたテキストによって決まります。そのため、マクロ変数の長さは、格納される値によって変わります。マクロ変数には、文字データのみを格納します。ただし、マクロ機能には、数値として解釈できる文字データが格納された場合に変数を数値として評価できる機能が備わっています。マクロ変数の値は、特に変更しない限り変わりません。マクロ変数は、SAS データセット変数とは無関係です。

注: マクロ変数には、印刷可能な文字のみを割り当ててください。印刷できない値をマクロ変数に割り当てると、予測できない結果を招く恐れがあります。

マクロプログラマーが定義したマクロ変数は、*ユーザー定義のマクロ変数*と呼ばれます。マクロプロセッサが定義したマクロ変数は、*自動マクロ変数*と呼ばれます。データ行内を除く SAS プログラム内の任意の場所で、マクロ変数を定義して使用できます。

マクロ変数を定義すると、マクロプロセッサは、プログラムのマクロ変数シンボルテーブルのうちのいずれかに、それを追加します。マクロ変数をマクロ定義の外側(*オープンコード*と呼ぶ)のステートメントで定義した場合、またはマクロプロセッサによって自動的に(SYSPBUFF 以外の)変数が作成された場合、変数は、グローバルシンボルテーブルに保持されます。グローバルシンボルテーブルは、SAS セッションの開始時にマクロプロセッサによって作成されます。マクロ変数をマクロ内で定義して、特にグローバルとして定義しなかった場合、通常、マクロ変数はマクロのローカルシンボルテーブルに保持されます。ローカルシンボルテーブルは、マクロの実行開始時に SAS によって作成されます。シンボルテーブルの詳細については、“SAS プログラムとマクロ処理” (13 ページ)および“マクロ変数のスコープ” (45 ページ)を参照してください。

マクロ変数は、グローバルシンボルテーブルに格納された場合、現在の SAS セッションの他の部分で使用されるために存在します。グローバルシンボルテーブル内の変数は、*グローバル マクロ変数*と呼ばれます。SAS セッションの任意の場所(ただし、CARDS ステートメントと DATALINES ステートメントを除く)でこの変数の値を使用できるため、この変数のスコープはグローバルです。SAS の他のコンポーネントによってグローバルマクロ変数が作成される場合もありますが、*自動 マクロ変数*と見なされるのはマクロプロセッサが作成したコンポーネントのみです。

マクロ変数は、ローカルシンボルテーブルに格納された場合、それが定義されたマクロが実行されている間だけ存在します。ローカルシンボルテーブル内の変数は、*ローカル マクロ変数*と呼ばれます。マクロが実行されている間だけこの変数の値を使用できるため、この変数のスコープはローカルです。“SAS プログラムとマクロ処理” (13 ページ) に示す図は、グローバルシンボルテーブルとローカルシンボルテーブルを使用したプログラムを説明しています。

%PUT ステートメントを使用して、現行の SAS セッションで使用可能なすべてのマクロ変数を表示できます。“%PUT ステートメント” (316 ページ) および “マクロ機能のエラーメッセージとデバッグ” (121 ページ)についても参照してください。

マクロプロセッサが定義するマクロ変数

SAS を起動すると、マクロプロセッサによって自動マクロ変数が作成されます。これらの変数は、SAS セッションに関連する情報を提供します。自動マクロ変数は、ローカルである SYSPBUFF を除き、グローバルです。

自動マクロ変数を使用するには、マクロ変数名の前にアンパサンドを付けて参照します(たとえば、&SYSJOBID)。次の FOOTNOTE ステートメントには、自動マクロ変数 SYSDAY および SYSDATE9 への参照が含まれています。

```
footnote "Report for &sysday, &sysdate9";
```

現在の SAS セッションが 2011 年 12 月 16 日に起動されている場合、マクロ変数を置換することによって、SAS は次のステートメントを受け取ります。

```
FOOTNOTE "Report for Friday, 16DEC2011";
```

自動マクロ変数は、多くの場合、条件付き論理(返される値によってアクションが決まる%IF ステートメントなど)で役立ちます。詳細については、“%IF-%THEN/%ELSE ステートメント” (302 ページ)を参照してください。

読み込みおよび書き込みステータスを持つ自動マクロ変数には、値を割り当てることができます。しかし、読み込み専用ステータスを持つ自動マクロ変数に値を割り当てることはできません。次のテーブルに、SAS マクロプロセッサによって作成された自動マクロ変数と、それらの読み込み/書き込みステータスを示します。

使用可能なすべての自動マクロ変数を表示するには、%PUT _AUTOMATIC_を使用します。

特定のプラットフォームでのみ作成される、システム固有のマクロ変数もあります。それらは、ホスト関連のドキュメントに記載されています。共通のマクロ変数については、“効率的なマクロとポータブルマクロの作成” (143 ページ)に示されています。他の SAS ソフトウェア製品でも、マクロ変数が提供されています。それらについては、マクロ変数を使用する製品のドキュメントで説明されています。これらのタイプのマクロ変数は、いずれも自動マクロ変数とは見なされません。

表 3.1 *カテゴリ別自動マクロ変数*

ステータス	変数	内容
読み込みおよび書き込み	SYSBUFFR	%INPUT からの不一致テキスト
	SYSCC	SAS によって動作環境に返される現在の条件コード(動作環境の条件コード)
	SYSCMD	マクロウィンドウのコマンドラインに入力された認識できない最後のコマンド
	SYSDEVIC	現在のグラフィックデバイスの名前
	SYSDMG	損傷したデータセットに対して実行されたアクションを反映するリターンコード
	SYSDSN	2 つのフィールド内の最新の SAS データセットの名前
	SYSFILRC	FILENAME ステートメントによって設定されたリターンコード
	SYSLAST	1 つのフィールド内の最新の SAS データセットの名前
	SYSLCKRC	LOCK ステートメントによって設定されたリターンコード
	SYSLIBRC	LIBNAME ステートメントによって設定されたリターンコード
	SYSLOGAPPLNAME	LOGAPPLNAME オプションの値
	SYSMSG	マクロウィンドウに表示されるメッセージ
	SYSPARM	SYSPARM=システムオプションで指定された値
	SYSPBUFF	マクロパラメータ値のテキスト
	SYSRC	システム関連のさまざまなリターンコード
読み込み専用	SYSADDRBITS	アドレスのビット数

ステータス	変数	内容
	SYSCHARWIDTH	文字の幅の値
	SYSDATE	SAS ジョブまたは SAS セッションの実行が開始された日付を表す文字値(年が 2 桁)
	SYSDATE9	SAS ジョブまたは SAS セッションの実行が開始された日付を表す文字値(年が 4 桁)
	SYSDAY	SAS ジョブまたは SAS セッションの実行が開始された曜日
	SYSENCODING	SAS セッションのエンコーディングの名前
	SYSENDIAN	現在のセッションのバイトオーダー
	SYSENV	フォアグラウンドまたはバックグラウンドのインジケータ
	SYSERR	SAS プロシジャと DATA ステップによって設定されるリターンコード
	SYSERRORTEXT	SAS ログ表示用にフォーマットした最終エラーメッセージの本文
	SYSHOSTNAME	動作環境のホスト名
	SYSINDEX	このセッション中に実行が開始されたマクロの数
	SYSINFO	リターンコード情報
	SYSJOBID	現在のバッチジョブ名またはユーザー ID (ホスト環境によって変わる)
	SYSMACRONAME	現在実行されているマクロの名前
	SYSMENV	現在のマクロ実行環境
	SYSNCPU	SAS が計算に使用できる現在のプロセッサの数
	SYSNOBS	最後のデータセットから読み込まれたオブザベーションの数
	SYSODSESCAPECHAR	プログラム内の ODS ESCAPECHAR=の値
	SYSODSPATH	Output Delivery System(ODS)の PATH 変数の値
	SYSPROCESSID	現在の SAS プロセスのプロセス ID
	SYSPROCESSNAME	現在の SAS プロセスのプロセス名

ステータス	変数	内容
	SYSPROCNAME	処理中の現在のプロシジャの名前
	SYSSCP	オペレーティングシステムの略称
	SYSSCPL	オペレーティングシステムの名前
	SYSSITE	サイトに割り当てられた番号
	SYSSIZEOFLONG	現在のセッションでのロング整数のバイト長
	SYSSIZEOFPTR	ポインタのバイトサイズ
	SYSSIZEOFUNICODE	現在のセッションでのユニコード文字のバイト長
	SYSSTARTID	最後の STARTSAS ステートメントから生成された ID
	SYSSTARTNAME	最後の STARTSAS ステートメントから生成されたプロセス名
	SYSTCPIPHOSTNAME	複数の TCP/IP スタックがサポートされている場合の、ローカル動作環境とリモート動作環境のホスト名
	SYSTIME	SAS ジョブまたは SAS セッションの実行が開始された時刻を表す文字値
	SYSUSERID	現在の SAS プロセスのユーザー ID またはログイン
	SYSVER	実行中の SAS ソフトウェアのリリース番号またはバージョン番号
	SYSVLONG	SAS ソフトウェアのリリース番号とメンテナンスレベルに 2 桁の年を加えた値
	SYSVLONG4	SAS ソフトウェアのリリース番号とメンテナンスレベルに 4 桁の年を加えた値
	SYSWARNINGTEXT	SAS ログ表示用にフォーマットした最終警告メッセージの本文

ユーザー定義のマクロ変数

マクロ変数定義の概要

独自のマクロ変数を作成して、それらの値を変更し、スコープを定義できます。マクロ内でマクロ変数を定義できます。また、%GLOBAL ステートメントを使用して定義することで、明示的にグローバル変数として定義することもできます。マクロ変数名は、文字またはアンダースコアで始める必要があり、その後に文字または数字を続けることができます。マクロ変数には、予約語以外の任意の名前を割り当てることができます。AF、DMS、SQL、および SYS という接頭語の使用は推奨されません。SAS ソフトウェアがマクロ変数を作成する際に、これらの接頭語を頻繁に使用するためです。そのため、これらの接頭語のうちのいずれかを使用すると、SAS ソフトウェアによって作成されたマクロ変数の名前との競合が発生する恐れがあります。マクロ言語の予約語の完全な一覧については、“ マクロ機能の予約語” (369 ページ)を参照してください。無効なマクロ変数名を割り当てた場合、SAS ログにエラーメッセージが出力されます。

%PUT _ALL_を使用して、ユーザーが作成したマクロ変数をすべて表示できます。

マクロ変数の作成と値の割り当て

マクロ変数を作成してそれに値を割り当てる最も簡単な方法は、次のように、マクロプログラムの%LET ステートメントを使用することです。

```
%let dsname=Newdata;
```

DSNAME は、マクロ変数の名前です。**Newdata** は、マクロ変数 DSNAME の値です。マクロ変数の値は、単なる文字列です。文字列には、任意の文字、数字、キーボード上にある印刷可能なシンボル、文字間の空白を含めることができます。大文字小文字の情報は、マクロ変数値に保存されます。一致しない引用符などの一部の文字については、特殊な扱いが必要です。これについては後述します。

マクロ変数がすでに存在する場合、それに値を割り当てると、マクロ変数の現在の値を置き換えます。マクロ変数またはその値にマクロのトリガ(%または&)が含まれる場合、そのトリガが評価されてから値が割り当てられます。次の例では、**&name** が **Cary** に置換されてから、次のステートメントで **city** の値として割り当てられています。

```
%let name=Cary;
%let city=&name;
```

通常、マクロプロセッサは、アルファベット、数字、およびシンボル(&と%を除く)を文字として扱います。マクロプロセッサは、特殊な処理を使用して&や%を文字として扱うこともできます。これについては後述します。マクロプロセッサは、SAS の他の部分とは異なり、文字と数値を区別しません。ただし、“%EVAL 関数” (247 ページ)と“%SYSEVALF 関数” (267 ページ)は、マクロ変数を整数または浮動小数点数として評価できます。

マクロ変数値は、マクロプロセッサが生成するテキスト、またはマクロプロセッサが使用するテキストを表すことができます。値の長さの範囲は、0 から 65,534 文字までです。値の引数を省略した場合、値は null (文字数 0)になります。デフォルトでは、値の前後の空白は値として格納されません。

%LET ステートメントに加えて、マクロ変数を作成するためのその他のマクロ言語の機能として、次のようなものが挙げられます。

- %DO ステートメントによる反復

- %GLOBAL ステートメント
- %INPUT ステートメント
- SQL の SELECT ステートメントの INTO 句
- %LOCAL ステートメント
- %MACRO ステートメント
- SCL の SYMPUT ルーチン、SYMPUTX ルーチン、および SYMPUTN ルーチン
- %WINDOW ステートメント

次の表で、マクロ変数にさまざまなタイプの値を割り当てる方法について説明します。

表 3.2 マクロ変数値の割り当てタイプ

割り当て	値

定数テキスト	文字列。次のステートメントは、**maple** という値をマクロ変数 STREET に割り当てることのできる複数の方法を示しています。いずれの場合も、マクロプロセッサは、**maple** という 5 文字の値を STREET の値として格納します。値の前後の空白は格納されません。
	`%let street=maple;`
	`%let street= maple;`
	`%let street=maple ;`
	注: 引用符は不要です。引用符を使用した場合、それらは値の一部に含まれます。

数字	適切な数字。次の例では、マクロ変数 NUM および TOTALSTRT を作成しています。
	`%let num=123;`
	`%let totalstr=100+200;`
	マクロプロセッサは、**123** を数値として扱わず、式 **100+200** を評価しません。その代わり、マクロプロセッサはすべての数字を文字として扱います。

演算式	次の例のように、%EVAL 関数を使用します。
	`%let num=%eval(100+200); / * produces 300 * /`
	あるいは、次の例のように%SYSEVALF 関数を使用します。
	`%let num=%sysevalf(100+1.597); / * produces 101.597 * /`
	詳細については、“マクロ評価関数” (162 ページ)を参照してください。

null 値	引数の値には何も割り当てられません。次に例を示します。 `%let country=;`

割り当て	値

マクロ変数参照	マクロ変数参照(*&macro-variable*)。次に例を示します。
	`%let street=Maple;`
	`%let num=123;`
	`%let address=&num &street Avenue;`
	この例は、テキスト式に含まれる複数のマクロ変数参照を示しています。マクロプロセッサは、割り当てを実行する前に、テキスト式の置換を試みます。その結果、マクロプロセッサは、マクロ変数 ADDRESS の値を **`123 Maple Avenue`** として格納します。
	%NRSTR 関数を使用して文字をマスクすることで、アンパサンドとパーセント記号をリテラルとして扱うことができます。これによって、マクロプロセッサは、アンパサンドとパーセント記号をマクロ呼び出しとして解釈せず、テキストとして扱うことができます。詳細については、“マクロ言語要素” (157 ページ)とマクロクォーティング (82 ページ)を参照してください。

マクロ呼び出し	マクロ呼び出し(*%macro-name*)。次に例を示します。
	`%let status=%wait;`
	この%LET ステートメントを実行すると、マクロプロセッサによって WAIT マクロも呼び出されます。マクロプロセッサは、WAIT マクロによって生成されたテキストを、STATUS の値として格納します。
	%LET ステートメントを実行したときにマクロが呼び出されないようにするには、次のように%NRSTR 関数を使用してパーセント記号をマスクします。
	`%let status=%nrstr(%wait);`
	マクロプロセッサは、**`%wait`** を STATUS の値として格納します。

空白と特殊文字	値を囲むマクロクォーティング関数、%STR または%NRSTR。マクロプロセッサが空白や特殊文字をテキストとして解釈できるようにするために、このアクションによってそれらの文字をマスクします。“マクロクォーティング関数” (163 ページ)を参照してください。次に例を示します。
	`%let state=%str( North Carolina);`
	`%let town=%str(Taylor%'s Pond);`
	`%let store=%nrstr(Smith&Jones);`
	`%let plotit=%str(`
	`proc plot;`
	`plot income*age;`
	`run;);`
	マクロ変数 TOWN を定義している部分では、%STR を使用して不一致の引用符を含む値をマスクしています。“マクロクォーティング関数” (163 ページ)で、不一致の引用符などのシンボルにマークを付ける必要のあるマクロクォーティング関数について説明しています。
	マクロ変数 PLOTIT を定義している部分では、%STR を使用して、マクロ変数値内の空白と特殊文字(セミコロン)をマスクしています。マクロ変数に完全な SAS ステートメントを含める場合、行を分けて、DATA ステップ内や PROC ステップ内でインデントしてステートメントを入力すると、読みやすくなります。マクロクォーティング関数を使用すると、マクロ変数値内の意味のある空白が維持されます。

割り当て	値

DATA ステップの値	SYMPUT ルーチン。次の例では、データセット内の AGE が 20 を超えるオブザベーションの数を、FOOTNOTE ステートメントに挿入しています。
	`data _null_;`
	`set in.permdata end=final;`
	`if age>20 then n+1;`
	`if final then call symput('number',trim(left(n)));`
	`run;`
	`footnote "&number Observations have AGE>20";`
	DATA ステップの最後の反復を実行するときに、SYMPUT ルーチンによって、N の値が格納された NUMBER という名前のマクロ変数が作成されます。その際、SAS は、数値から文字への変換を示すメッセージを発行します。DATA ステップ変数 N の値をマクロ変数 NUMBER に割り当てる前に、TRIM 関数と LEFT 関数によって、変数 N の値から余分なスペースを削除します。
	数値から文字への変換を示すメッセージを抑制することを含む、SYMPUT の説明については、“CALL SYMPUT ルーチン” (228 ページ)を参照してください。

マクロ変数の使用

マクロ変数の参照

マクロ変数を作成したら、通常は、マクロ変数名の前にアンパサンドを付けて (*&variable-name*)参照します。この参照を、*マクロ変数参照*と呼びます。これらの参照を値に置換するときに、シンボリック置換が実行されます。これらの参照は、SAS プログラム内の任意の場所で使用できます。文字列に現れるマクロ変数参照を置換するには、その文字列を二重引用符で囲みます。一重引用符で囲まれたマクロ変数参照は、置換されません。値をマクロ変数 DSN に割り当て、それを TITLE ステートメントで使用する、次の各ステートメントを比較してください。

```
%let dsn=Newdata;
title1 "Contents of Data Set &dsn";
title2 'Contents of Data Set &dsn';
```

1 番目の TITLE ステートメントでは、マクロプロセッサは、&DSN をマクロ変数 DSN の値に置き換えて参照を置換しています。2 番目の TITLE ステートメントでは、&DNS は DSN の値に置き換えられません。SAS は、各ステートメントを次のように解釈します。

```
TITLE1 "Contents of Data Set Newdata";
TITLE2 'Contents of Data Set &dsn';
```

マクロ変数は、SAS プログラム内で必要なだけ何度でも参照できます。値は、変更しない限り変わりません。たとえば、次のプログラムではマクロ変数 DSN を 2 回参照しています。

```
%let dsn=Newdata;
data temp;
set &dsn;
if age>=20;
run;
```

```
proc print;
title "Subset of Data Set &dsn";
run;
```

マクロプロセッサは、&DSN の参照が現れるたびに、それを **Newdata** に置き換えます。SAS は、各ステートメントを次のように解釈します。

```
DATA TEMP;
SET NEWDATA;
IF AGE>=20;
RUN;

PROC PRINT;
TITLE "Subset of Data Set NewData";
RUN;
```

注: 存在しないマクロ変数を参照した場合、SAS ログに警告メッセージが出力されます。たとえば、次のように、マクロ変数 JERRY のスペルを誤って JERY と記述した場合、予期しない結果が生じます。

```
%let jerry=student;
data temp;
x="produced by &jery";
run;
```

このコードによって、次のメッセージが生成されます。

```
WARNING: Apparent symbolic reference JERY not resolved.
```

マクロ変数参照とテキストを結合する

テキストの先頭または末尾にマクロ変数参照を加えたり(たとえば、DATA=PERSNL&YR.EMPLOYES。ここで、&YR には 2 文字の年が格納される)、隣接する変数(たとえば、&MONTH&YR)を参照したりすることは、多くの場合役立ちます。同じテキストを複数の場所で再利用できます。つまり、使用するごとに値を変更できるため、プログラムを再利用できます。

同じテキストを複数の場所で再利用するには、共通の要素を表すマクロ変数参照を使用してプログラムを記述します。次に示すように、1 つの%LET ステートメントを使用して、すべての場所の値を変更できます。

```
%let name=sales;
data new&name;
set save.&name;
more SAS statements
if units>100;
run;
```

マクロ変数が置換されると、SAS は各ステートメントを次のように解釈します。

```
DATA NEWSALES;
SET SAVE.SALES;
more SAS statements
IF UNITS>100;
RUN;
```

なお、マクロ変数参照では、DATA ステップで必要な連結演算子は不要です。SAS によって、自動的にワードが作成されます。

テキスト内のマクロ変数名を区切る

マクロ変数参照を接頭語として使用したときに、それを単純に連結した場合、参照が期待通りに置換されないことがあります。そうする代わりに、参照の末尾にピリオドを加えて区切ることが必要な場合があります。

マクロ変数参照の直後のピリオドは、区切り文字の役割を果たします。つまり、参照の末尾のピリオドは、マクロプロセッサに参照の末尾を強制的に認識させます。このピリオドは、生成されるテキストには現れません。

引き続き前述の例において、SALES1、SALES2、および INSALES.TEMP という名前を使用する別の DATA ステップが必要になったとします。次のステップをプログラムに追加できます。

```
/* first attempt to add suffixes--incorrect */
data &name1 &name2;
set in&name.temp;
run;
```

マクロ変数が置換されると、SAS は各ステートメントを次のように解釈します。

```
DATA &NAME1 &NAME2;
SET INSALESTEMP;
RUN;
```

マクロ変数参照は、意図したとおりに置換されていません。マクロプロセッサは警告メッセージを発行し、SAS は構文エラーメッセージを発行します。理由は次のとおりです。

NAME1 と NAME2 が有効な SAS 名であるため、マクロプロセッサは、NAME ではなくそれらの名前を持つマクロ変数を検索します。その結果、参照が置換されずに DATA ステートメントに渡されます。

マクロ変数参照では、ワードスキャナは、SAS 名として使用されない文字を検出したときにマクロ変数名の終端を認識します。ただし、ピリオド(.) を、マクロ変数参照の区切り文字として使用できます。たとえばこの例では、NAME というワードの終端をマクロプロセッサに認識させるには、次のように、ピリオドを区切り文字として&NAME と接尾語の間で使用します。

```
/* correct version */
data &name.1 &name.2;
```

これで、SAS はこのステートメントを次のように解釈します。

```
DATA SALES1 SALES2;
```

置換済みテキストの後ろにピリオドを挿入する

マクロプロセッサによって置換されたテキストの後ろに、ピリオドを挿入する必要がある場合があります。たとえば、2 レベルのデータセット名では、ライブラリ参照名とデータセット名の間にピリオドを含める必要があります。

マクロ変数参照名の後ろにピリオドを付ける場合、2 つのピリオドを使用します。1 番目のピリオドがマクロ変数参照の区切り文字になり、2 番目のピリオドがテキストの一部になります。

```
set in&name..temp;
```

マクロ変数が置換されると、SAS はこのステートメントを次のように解釈します。

```
SET INSALES.TEMP;
```

区切り文字を使用して任意のマクロ変数参照を区切ることができますが、区切り文字が必要になるのは、その後ろの文字が SAS 名の一部になる場合のみです。たとえば、次の 2 つはいずれも正しい TITLE ステートメントです。

```
title "&name.--a report";
title "&name--a report";
```

これらによって、次が生成されます。

```
TITLE "sales--a report";
```

マクロ変数値の表示

マクロ変数値を表示する最も簡単な方法は、%PUT ステートメントを使用することです。このステートメントは、SAS ログにテキストを書き込みます。たとえば、次のステートメントによって、その次の結果が書き込まれます。

```
%let a=first;
%let b=macro variable;
%put &a ***&b***;
```

次に結果を示します。

```
first ***macro variable***
```

“%PUT ステートメント” (316 ページ)を使用して、使用可能なマクロ変数を表示することもできます。%PUT には、マクロ変数の個別のカテゴリを表示できる複数のオプションが用意されています。

SYMBOLGEN システムオプションは、マクロ変数の置換結果を表示します。たとえば次の例で、マクロ変数 PROC および DEST に、それぞれ GPLOT および SASUSER.HOUSES という値が格納されているとします。

```
options symbolgen;
title "%upcase(&proc) of %upcase(&dset)";
```

SYMBOLGEN オプションは、次の結果をログに出力します。

```
SYMBOLGEN: Macro variable PROC resolves to gplot
SYMBOLGEN: Macro variable DSET resolves to sasuser.houses
```

マクロプログラムのデバッグの詳細については、“マクロ機能のエラーメッセージとデバッグ” (121 ページ)を参照してください。

マクロ変数の間接的な参照

式を使用して参照を生成する

これまでに示したマクロ変数参照は、1 つのアンパサンドで始まる直接的なマクロ変数参照(*&name*)でした。一方、一連のマクロ変数に属するマクロ変数を間接的に参照できるということも、役に立ちます。これによって、マクロ変数参照が置換されるときに、その名前を決定できます。マクロ機能として、間接的なマクロ変数参照が提供されています。この機能によって、式(たとえば、CITY&N)を使用して一連のマクロ変数のうちの 1 つへの参照を生成できます。たとえば、マクロ変数 N の値を使用して、CITY1 から CITY20 までの名前を持つ一連のマクロ変数のうちの 1 つを参照できます。N の

値が 8 の場合、CITY8 への参照になります。N の値が 3 の場合、CITY3 への参照になります。

この例の場合、必要な参照のタイプは CITY&N ですが、次の例は、期待する値ではなく CITY に&N を加えた値を生成します。

```
%put &city&n; /* incorrect */
```

このコードは、マクロ変数 CITY が存在しないことを示す警告メッセージを生成します。これは、マクロ機能が&CITY を置換してから&N を置換し、それらの値を連結しようとするためです。

間接的なマクロ変数参照を使用する場合、マクロプロセッサに対して、強制的に 2 回以上マクロ変数参照をスキャンさせ、2 回目以降のスキャンで目的の参照を置換させる必要があります。マクロプロセッサに強制的にマクロ変数参照を再スキャンさせるには、マクロ変数参照で 2 つ以上のアンパサンドを使用します。マクロプロセッサが複数のアンパサンドを検出した場合に行う基本的なアクションは、2 つのアンパサンドを 1 つのアンパサンドに置換することです。たとえば、&N の値を CITY に追加して該当する変数名を参照する場合、次のように実行します。

```
%put &&city&n; /* correct */
```

N の値が 6 の場合、マクロプロセッサがこのステートメントを受け取ると、次のステップが実行されます。

1. &&を&に置換します。
2. CITY をテキストとして渡します。
3. &N を 6 に置換します。
4. マクロ変数参照(&CITY6)の先頭に戻り、最初から置換を再開し、CITY6 の値を出力します。

単一のマクロ呼び出しを使用して一連のマクロ変数参照を作成する

間接的なマクロ変数参照を使用すると、%DO ループの反復を使用することにより、単一のマクロ呼び出しによって複数の参照を生成することができます。次の例では、マクロ変数 CITY1 から CITY10 までに、それぞれ Cary、New York、Chicago、Los Angeles、Austin、Boston、Orlando、Dallas、Knoxville、Asheville の値が格納されていることを前提にしています。

```
%macro listthem;
%do n=1 %to 10; &&city&n
%end;
%mend listthem;

%put %listthem;
```

このプログラムは、次の結果を SAS ログに書き込みます。

```
Cary New York Chicago Los Angeles Austin Boston
Orlando Dallas Knoxville Asheville
```

3 つ以上のアンパサンドの使用

間接的なマクロ変数参照では、任意の個数のアンパサンドを使用できますが、4 つ以上を使用することはまれです。このタイプの参照では、使用するアンパサンドの個数に

かかわらず、マクロプロセッサによって次のステップが実行されて参照が置換されます。

```
%let var=city;
%let n=6;
%put &&&var&n;
```

1. 参照全体が左から右に向かって置換されます。アンパサンドのペア(&&)が検出された場合、そのペアは 1 つのアンパサンドに置換されます。その後、参照の次のペアが処理されます。この例では、&&&VAR&N は、&CITY6 になります。
2. 前で得られた結果の先頭に戻り、左から右に向かって置換を再開します。すべてのアンパサンドが完全に処理されると、置換は完了です。この例では、&CITY6 が Boston に置換されて、置換処理が終了します。

注: 間接的なマクロ変数参照の置換の実行中に、置換の一部にマクロ呼び出しを含めることはできません。

ヒント: 三重のアンパサンドによって間接的なマクロ変数参照を使用すると、マクロプロセッサの効率が向上する場合があります。詳細については、“効率的なマクロとポータブルマクロの作成” (143 ページ)を参照してください。

マクロ関数を使用したマクロ変数値の操作

マクロ変数を定義する場合、式にマクロ関数を含めることによって、変数の値を格納する前に操作できます。たとえば、関数を使用して、他の値のスキャン、演算式や論理式の評価、特殊文字(不一致の引用符など)の意味の削除などを実行できます。

マクロ変数値内のワードをスキャンするには、次のように%SCAN 関数を使用します。

```
%let address=123 maple avenue;
%let frstword=%scan(&address,1);
```

最初の%LET ステートメントでは、文字列 **123 maple avenue** をマクロ変数 ADDRESS に割り当てています。2 番目の%LET ステートメントでは、%SCAN 関数を使用して入力データ(1 番目の引数)を検索し、1 番目のワード(2 番目の引数)を取り出しています。マクロプロセッサは、値を格納する前に%SCAN 関数を実行します。そのため、FRSTWORD の値は文字列 **123** になります。

%SCAN の詳細については、“%SCAN 関数と%QSCAN 関数” (254 ページ)を参照してください。マクロ関数の詳細については、“マクロ言語要素” (157 ページ)を参照してください。

4 章

マクロ処理

マクロ処理

このセクションでは、マクロ処理について説明し、SAS がマクロ要素を含むプログラムを処理する際に従う典型的なパターンを示します。ほとんどのマクロのプログラミングでは、これ程詳細な情報は必要ありません。ここでは、プログラムの背後で実行されていることの理解に役立てるために説明します。

マクロの定義および呼び出し

*マクロ*とは、コンパイル済みのプログラムのことです。サブミットされた SAS プログラム内、または SAS コマンドプロンプトから呼び出すことができます。通常、マクロは、マクロ変数と同様にテキストの生成に使用されます。しかしマクロは、それ以外に次の機能を提供します。

- マクロには、テキストの生成方法と生成のタイミングを制御できるプログラムステートメント含めることができます。
- マクロは、パラメータを受け取ることができます。多くの用途に使用できる汎用的なマクロを記述できます。

マクロをコンパイルするには、マクロ定義をサブミットする必要があります。マクロ定義の一般的な形式を次に示します。

%MACRO *macro_name*;
<*macro_text*>
%MEND <*macro_name*>;

macro_name は、マクロを識別する固有の SAS 名です。*macro_text* は、マクロステートメント、マクロ呼び出し、テキスト式、または定数テキストの任意の組み合わせから成ります。

マクロ定義をサブミットすると、マクロプロセッサは、マクロ定義をコンパイルしてセッションカタログにメンバを生成します。メンバは、コンパイル済みマクロプログラムステートメントとテキストから成ります。マクロを実行するためには、コンパイル済み項目とコンパイル対象外(テキスト)の項目を区別することが重要です。テキスト項目の例を次に示します。

- マクロ変数の参照
- ネストされたマクロ呼び出し
- マクロ関数(ただし、%STR および%NRSTR を除く)
- 演算マクロ式および論理マクロ式
- %PUT ステートメントによって書き込まれるテキスト
- %WINDOW ステートメント内のフィールド定義
- SAS ステートメントおよび SAS ウインドウ環境のコマンドのためのモデルテキスト

マクロを呼び出す場合、次の形式を使用します。

%macro_name

マクロプロセッサによるマクロ定義のコンパイル方法

SAS プログラムをサブミットすると、プログラムの内容は入力スタックと呼ばれるメモリ領域に格納されます。次の図のプログラム例には、マクロ定義、マクロ呼び出し、および PROC PRINT ステップが含まれています。このセクションでは、プログラム例のマクロ定義がどのようにコンパイルされて格納されるかについて説明します。

図 4.1 *APP マクロ*

Input Stack

```
%macro app(goal);
   %if &sysday=Friday %then
      %do;
         data thisweek;
           set lastweek;
           if totsales > &goal
             then bonus = .03;
           else bonus = 0;
      %end;
%mend app;
%app(10000)
proc print;
run;
```

ワードスキャナは、“SAS プログラムとマクロ処理” (13 ページ)で説明したのと同じプロセスに従って、プログラムのトークン化を開始します。ワードスキャナは、最初のトークンで%の後に空白以外の文字が続くのを検出すると、マクロプロセッサを起動します。マクロプロセッサは、そのトークンを調べてマクロ定義の開始を認識します。マクロプロセッサは、%MEND ステートメントによってマクロ定義(図 4.2 (37 ページ))が終了するまで、入力スタックからトークンを取り出してコンパイルします。

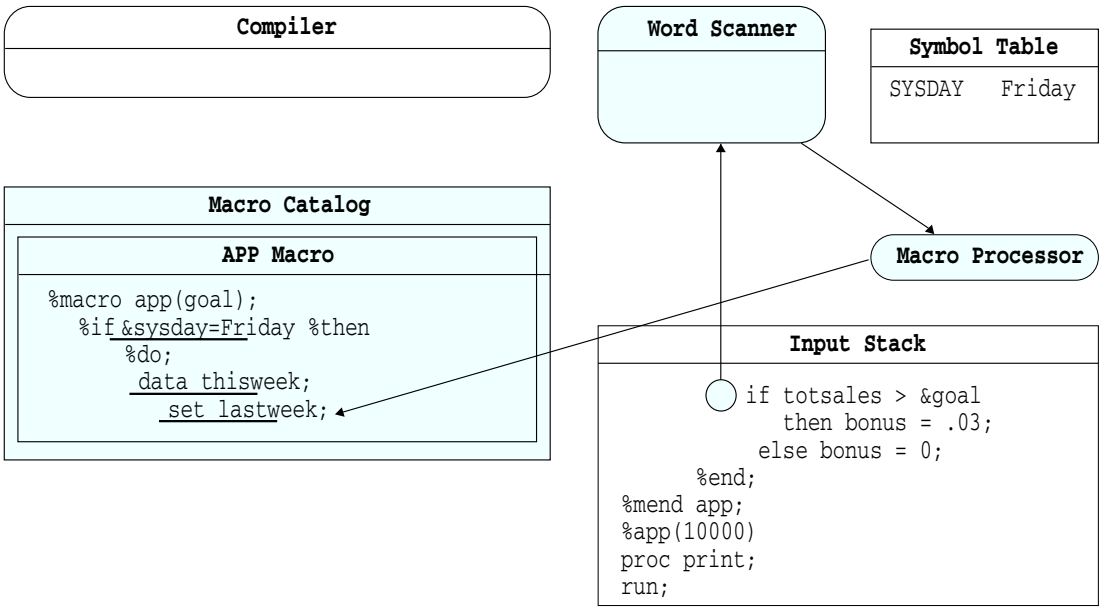
マクロプロセッサは、マクロのコンパイル中に次を実行します。

- セッションカタログ内にエントリを作成します。
- そのマクロのすべてのマクロプログラムステートメントを、マクロ命令としてコンパイルし、格納します。
- マクロ内のすべてのコンパイル対象外項目をテキストとして格納します。

  注: このセクションの図では、テキスト項目にはアンダーラインを引いています。

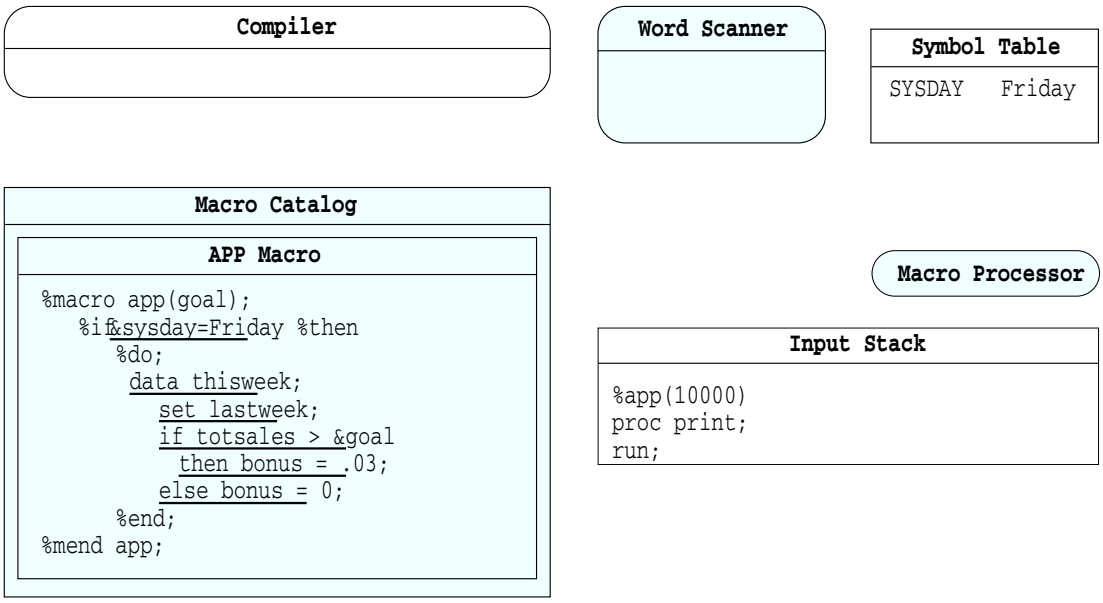
マクロプロセッサは、マクロのコンパイル中に構文エラーを検出した場合、マクロの残りの部分の構文をチェックし、検出したすべてのエラーについてメッセージを発行します。ただし、マクロプロセッサは、実行用のマクロを格納しません。マクロプロセッサによってコンパイルされるが格納されないマクロを、*ダミーマクロ*と呼びます。

図 4.2 *入力スタック内の APP マクロ*



この例では、マクロ定義がコンパイルされて、正常に格納されます。次の図を参照してください。説明のために、コンパイル済み APP マクロを、入力スタックに格納された元のマクロ定義と同じように示しています。実際は、エントリには、コンパイル済みマクロ命令と定数テキストが格納されます。この例では、定数テキストにアンダーラインを引いています。

図 4.3 コンパイル済み APP マクロ



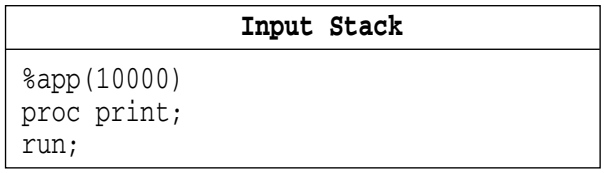
マクロプロセッサによるコンパイル済みマクロの実行方法

マクロの実行は、マクロプロセッサによって SASMACR カタログを開き、該当するマクロエントリを読み込むことから開始されます。マクロプロセッサは、マクロエントリ内のコンパイル済み命令を実行する際に、一連の単純な反復アクションを実行します。マクロプロセッサは、マクロの実行中に次を実行します。

- コンパイル済みマクロプログラム命令を実行します。
- コンパイル対象外の定数テキストを入力スタックに配置します。
- 生成されたテキストをワードスキャナが処理するのを待機します。
- コンパイル済みマクロプログラム命令の実行を再開します。

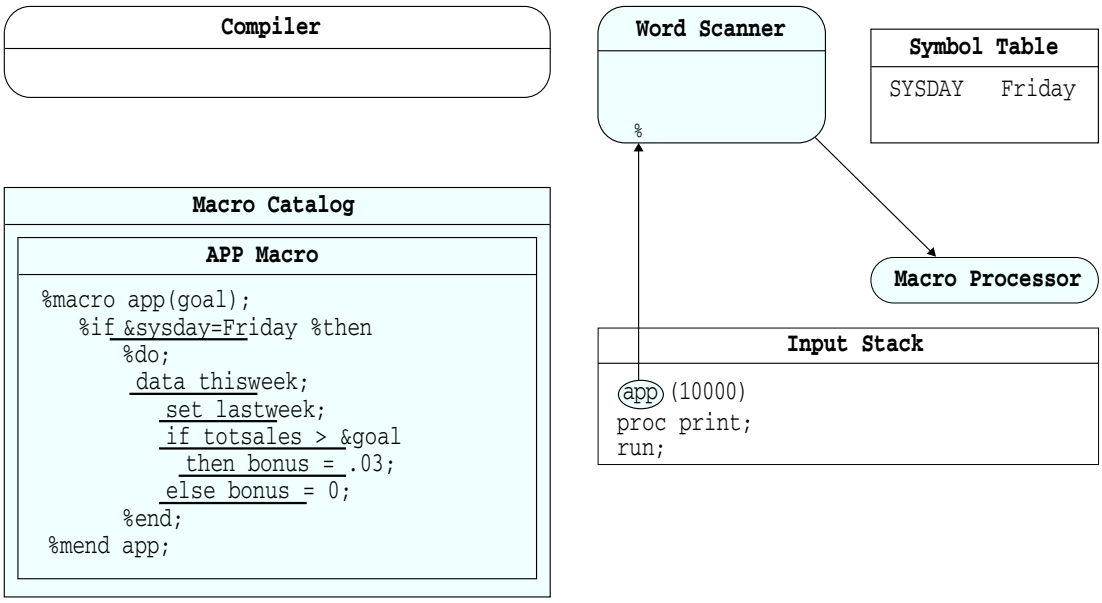
前のセクションの例を引き続き実行するために、次の図に、マクロプロセッサによって APP マクロ定義がコンパイルされた後に入力スタックに残された行を示します。

図 4.4 入力スタック内のマクロ呼び出し



ワードスキャナは、入力スタックを調べ、%の後に空白以外の文字が続いているのを最初のトークンで検出します。これによってマクロプロセッサが起動され、そのトークンを調べます。

図 4.5 ワードキューに入力されるマクロ呼び出し



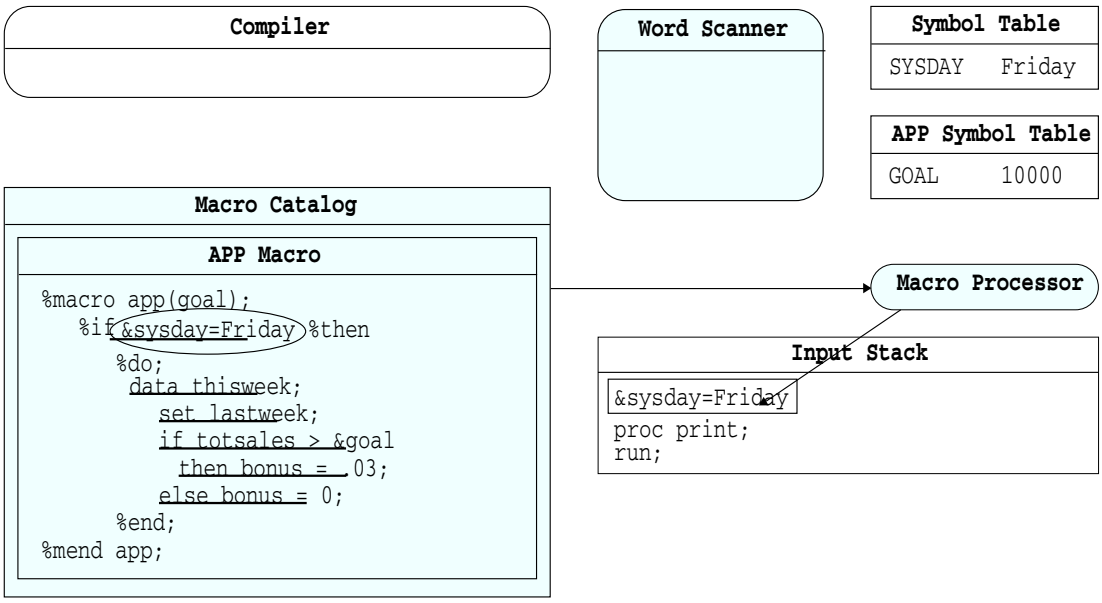
マクロプロセッサは、マクロ呼び出しを認識し、次のようにして APP マクロの実行を開始します。

1. マクロプロセッサは、このマクロ用のローカルシンボルテーブルを作成します。マクロプロセッサは、このマクロのコンパイル済み定義を調べます。マクロ定義にパラメータ、変数宣言、または計算済みの GOTO ステートメントが存在する場合、マクロプロセッサは、パラメータや変数のエントリを新規作成のローカルシンボルテーブルに追加します。
2. マクロプロセッサは、このマクロのパラメータについて、コンパイル済みマクロ定義をさらに調べます。マクロ定義にパラメータが定義されていない場合、マクロプロセッサはマクロのコンパイル済み命令の実行を開始します。マクロ定義にパラメータが含まれている場合、マクロプロセッサは入力スタックからトークンを削除して、位置パラメータの値とデフォルト以外のキーワードパラメータの値を取得します。入力スタックで検出されたパラメータの値は、ローカルシンボルテーブル内の該当するエントリに配置されます。

   注: マクロプロセッサは、コンパイル済み命令を実行する前に、ユーザーによって入力されたマクロ呼び出しに関係するすべてのトークンが削除されたことを確認するため、それに必要なトークンのみを入力スタックから削除します。

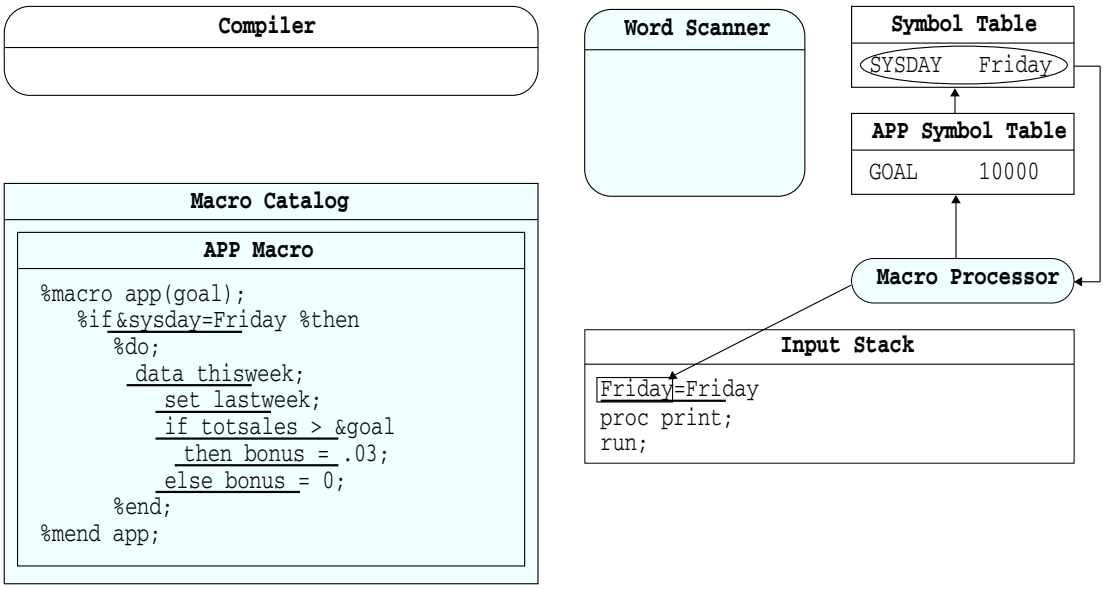
3. マクロプロセッサは、コンパイル済み%IF 命令を検出し、次の項目が条件を含むテキストであることを認識します。
4. マクロプロセッサは、テキスト**&sysday=Friday** を、入力スタックのプログラムのその他のテキストの前に配置します。(次の図を参照。)マクロプロセッサは、生成されたテキストをワードスキャナがトークン化するのを待機します。

図 4.6 入力スタックの%IF 条件のテキスト



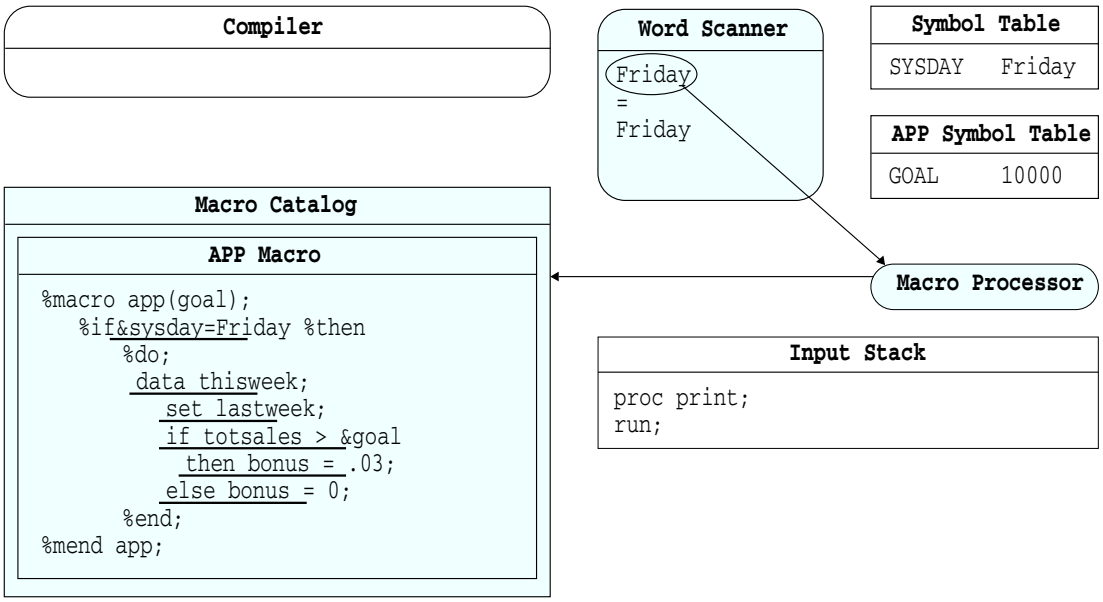
1. ワードスキャナは、生成されたテキストのトークン化を開始し、アンパサンドの後に空白以外の文字が続くのを最初のトークンで認識して、マクロプロセッサを起動します。
2. マクロプロセッサは、トークンを調べて、マクロ変数参照である可能性のある &SYSDAY を検出します。マクロプロセッサは、SYSDAY と一致するエントリを見つけるため、まず APP のローカルシンボルテーブルを検索し、次にグローバルシンボルテーブルを検索します。マクロプロセッサは、グローバルシンボルテーブル内で一致するエントリを検出すると、入力スタック内のマクロ変数をそのエントリの値 **Friday** で置き換えます。(次の図を参照。)
3. マクロプロセッサは停止し、生成されたテキストをワードスキャナがトークン化するのを待機します。

図 4.7 マクロ変数参照が置換された後の入力スタック



1. 次に、ワードスキャナは、入力スタックから **Friday=Friday** を読み込みます。
2. マクロプロセッサは、式 **Friday=Friday** を評価し、評価結果が true であるため、%THEN 命令と%DO 命令に進みます。

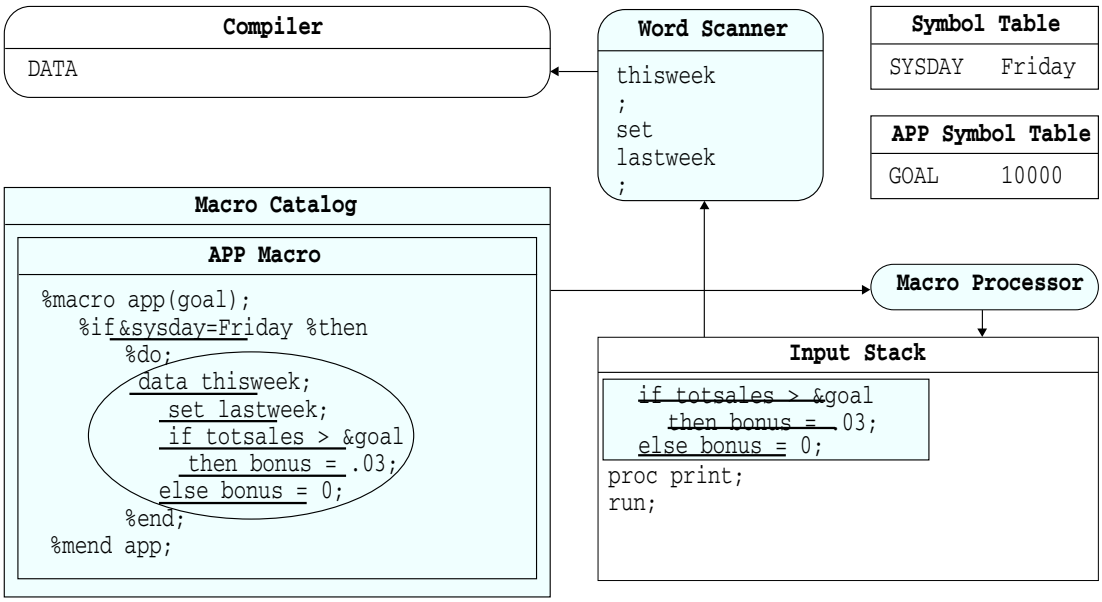
図 4.8 マクロプロセッサでの条件の受信



1. マクロプロセッサは、コンパイル済み%DO 命令を実行し、次の項目がテキストであることを認識します。
2. マクロプロセッサは、このテキストを入力スタックの先頭に配置し、ワードスキャナがトークン化を開始するのを待機します。

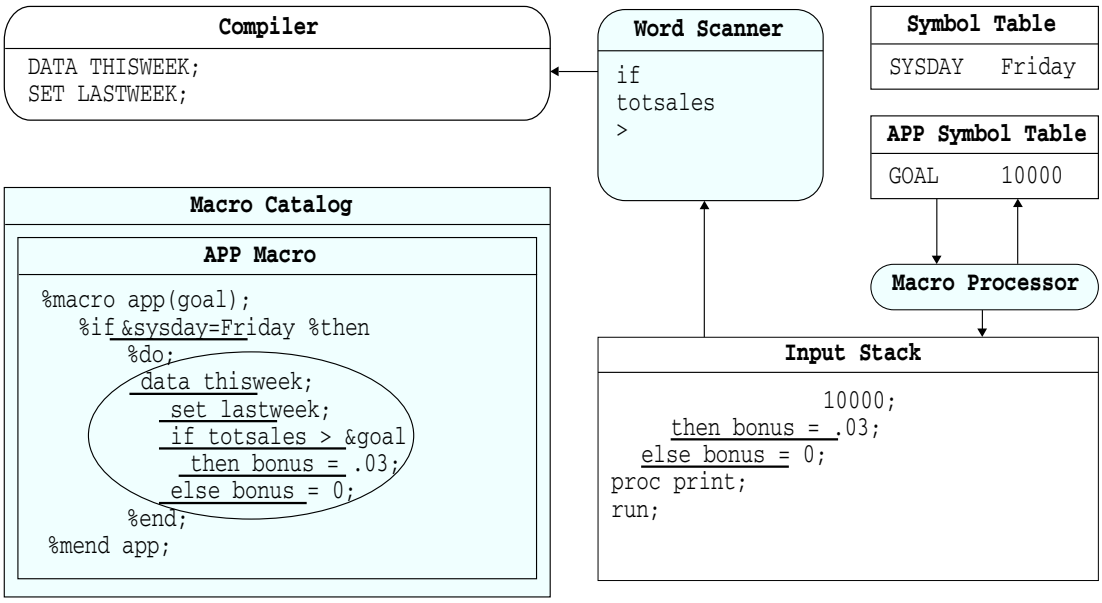
3. ワードスキャナは、生成されたテキストを入力スタックから読み込み、それをトークン化します。
4. ワードスキャナは、DATA ステップの開始を認識し、コンパイラを起動してトークンの受け取りを開始させます。ワードスキャナは、トークンをスタックの先頭からコンパイラに転送します。

図 4.9 *入力スタックの先頭に生成されたテキスト*



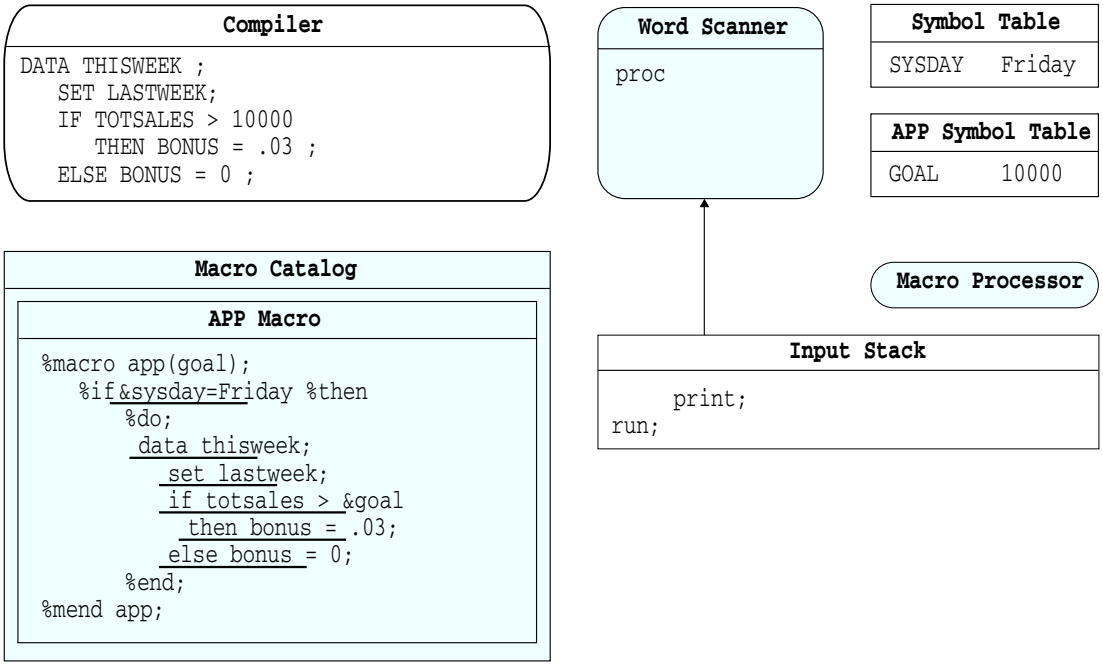
1. ワードスキャナは、&の後ろに空白以外の文字(マクロ変数参照&GOAL)が続いているのを検出すると、マクロプロセッサを起動します。
2. マクロプロセッサは、APP のローカルシンボルテーブルを検索し、マクロ変数参照 &GOAL を `10000` に置換します。マクロプロセッサは、その値を、入力スタックの先頭、つまりプログラムのその他のテキストの前に配置します。

図 4.10 *ワードスキャナによる生成されたテキストの読み込み*



1. ワードスキャナは、トークン化を再開します。生成されたテキストのトークン化が完了すると、マクロプロセッサが起動されます。
2. マクロプロセッサは、コンパイル済みマクロ命令の処理を再開します。マクロプロセッサは、%END 命令で%DO グループの終了を認識し、%MEND に進みます。
3. マクロプロセッサは、%MEND 命令を実行し、APP のローカルシンボルテーブルを削除します。これにより、APP マクロの実行が停止します。
4. マクロプロセッサは、トークン化を再開するためにワードスキャナを起動します。
5. ワードスキャナは、入力スタックの最初のトークン(PROC)を読み込んでステップの境界の開始を認識し、DATA ステップコンパイラを起動します。
6. コンパイル済み DATA ステップが実行され、DATA ステップコンパイラがクリアされます。
7. ワードスキャナは、PRINT プロシジャ(独立して実行され、図には示されていません)に信号を送ります。PRINT プロシジャは、残りのトークンを取り出します。

図 4.11 残りのステートメントのコンパイルと実行



マクロ処理の概要

前のセクションでは、マクロのコンパイルと実行、および DATA ステップのコンパイルと実行との間の関係について説明しました。この関係には、単純な反復アクションのパターンが含まれています。これらのアクションは、テキストが入力スタックにサブミットされて、ワードスキャナがトークン化を開始すると、開始されます。ワードスキャナは、マクロプロセッサがシンボルテーブルの検索やマクロ定義のコンパイルなどの処理を実行するのを何度か待機します。マクロプロセッサは、処理中にテキストを生成した場合、ワードスキャナによってそのテキストがトークン化されて適切なターゲットに送信される間、一時停止します。これらのトークンによって、SAS に含まれる DATA ステップコンパイラ、コマンドプロセッサ、SAS プロシジャなどの他のアクションが起動される場合があります。これらのアクションのいずれかが発生すると、マクロプロセッサはアクションの完了を待機し、その後処理を再開します。マクロプロセッサが停止すると、ワードスキャナがトークン化を再開します。このプロセスは、プログラム全体の処理が完了するまで続きます。

5 章

マクロ変数のスコープ

マクロ変数のスコープ

すべてのマクロ変数には、*スコープ*があります。マクロ変数のスコープは、マクロ変数にどのように値が割り当てられ、マクロプロセッサよってどのようにマクロ変数参照が置換されるかを決定します。

マクロ変数には、*グローバル*と*ローカル*という 2 種類のスコープが存在します。グローバルマクロ変数は、SAS セッションが存続する間存在し、マクロの内部、外部を問わず、プログラム内のどの場所(ただし、CARDS と DATALINES を除く)でも参照できます。ローカルマクロ変数は、それが作成されたマクロが実行中の間だけ存在し、マクロ定義の外側では意味を持ちません。

スコープは、箱の中に箱を入れるようにネストすることができます。たとえば、マクロ変数 LOC1 を作成するマクロ A と、マクロ変数 LOC2 を作成するマクロ B が存在するとします。マクロ B がマクロ A の内部でネスト(実行)されている場合、LOC1 は A と B の両方に対してローカルです。しかし、LOC2 は、B に対してのみローカルです。

マクロ変数は、マクロ変数の名前と値の一覧を保持する*シンボルテーブル*に格納されます。すべてのグローバルマクロ変数を格納する、グローバルシンボルテーブルが存在します。ローカルマクロ変数は、マクロの実行開始時に作成されるローカルシンボルテーブルに格納されます。

%SYMEXIST 関数を使用して、マクロ変数が存在するかどうかを表示できます。詳細については、“%SYMEXIST 関数” (265 ページ) を参照してください。

グローバルマクロ変数

次の図は、次のプログラムの実行中に作成されるグローバルシンボルテーブルを示しています。

```
%let county=Clark;

%macro concat;
data _null_;
length longname $20;
longname="&county"||" County";
put longname;
run;
%mend concat;

%concat
```

マクロ CONCAT を呼び出すと、次のステートメントが生成されます。

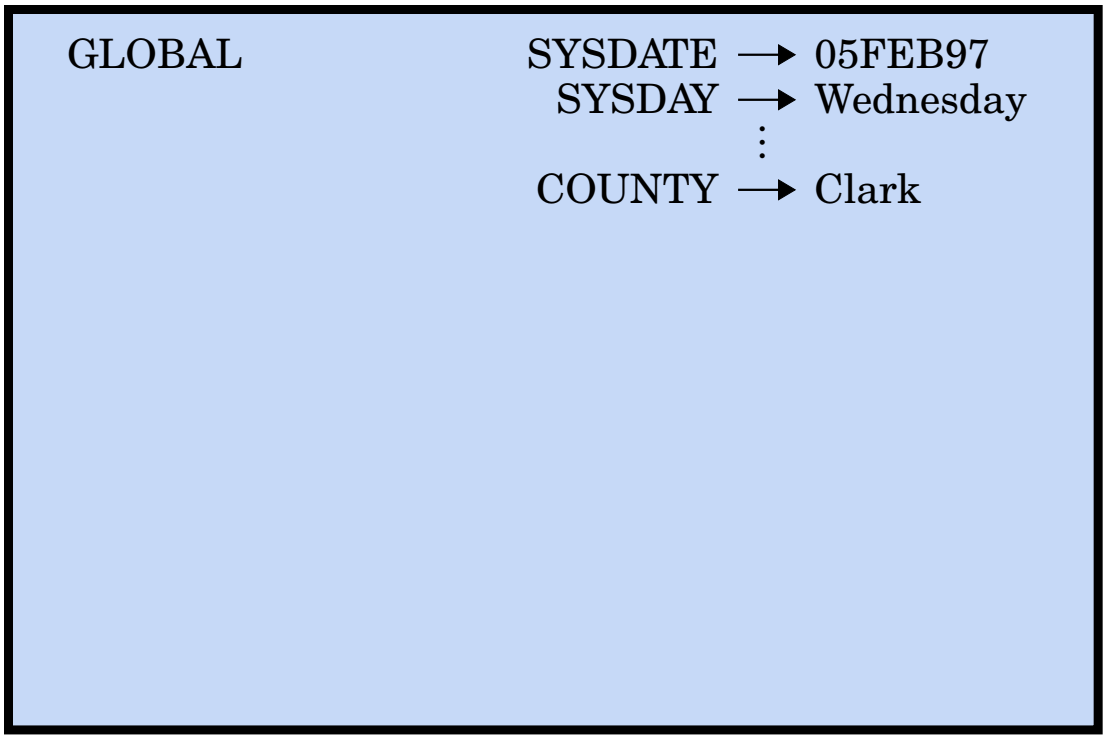
```
data _null_;
length longname $20;
longname="Clark"||" County";
put longname;
run;
```

PUT ステートメントは、

```
Clark County
```

を SAS ログに書き込みます。

図 5.1 グローバルシンボルテーブル



グローバルマクロ変数には、次のものがあります。

- SYSPBUFF 以外のすべての自動マクロ変数。SYSPBUFF などの自動マクロ変数の詳細については、“自動マクロ変数” (166 ページ) を参照してください。
- マクロの外部で作成されたマクロ変数。
- %GLOBAL ステートメントで作成されたマクロ変数。%GLOBAL ステートメントの詳細については、“グローバルマクロ変数の作成” (63 ページ) を参照してください。
- CALL SYMPUT ルーチンによって作成されたほとんどのマクロ変数。CALL SYMPUT ルーチンの詳細については、“CALL SYMPUT ルーチンを使用したスコープの特殊なケース” (65 ページ) を参照してください。

SAS セッションまたは SAS ジョブが存続する間は、いつでもグローバルマクロ変数を作成できます。SAS セッションまたは SAS ジョブが存続する間は、一部の自動マクロ変数を除き、いつでもグローバルマクロ変数の値を変更できます。

ほとんどの場合、グローバルマクロ変数を定義したら、SAS セッションまたは SAS ジョブのどの場所でも、その値を参照したり、変更したりできます。そのため、マクロ定義の内部で参照されるマクロ変数は、それと同じ名前のグローバルマクロ変数がすでに存在する場合、グローバルです(ただし、その変数が%LOCAL ステートメントを使用して、またはパラメータリストでローカルとして特に定義されていないことが前提です)。新しいマクロ変数を定義した場合、既存のグローバルマクロ変数は単に更新されます。グローバルマクロ変数の値を参照できない、次のような例外があります。

- マクロ変数が、グローバルシンボルテーブルとローカルシンボルテーブルの両方に存在する場合、そのローカルマクロ変数を含むマクロからは、グローバルの値を参照できません。この場合、マクロプロセッサは、グローバルの値ではなく、ローカルの値を最初に検出して使用します。
- マクロ変数を SYMPUT ルーチンを使用して DATA ステップ内で作成した場合、プログラムがステップ境界に達するまでは、アンパサンドを使用して値を参照するこ

とはできません。マクロ処理とステップ境界の詳細については、“マクロ処理” (35 ページ) を参照してください。

%SYMGLOBL 関数を使用して、既存のマクロ変数がグローバルシンボルテーブルに存在するかどうかを表示できます。詳細については、“%SYMGLOBL 関数” (265 ページ) を参照してください。

ローカルマクロ変数

ローカルマクロ変数は、個々のマクロ内で定義されます。呼び出されたマクロは、それぞれ専用のローカルシンボルテーブルを作成します。ローカルマクロ変数は、特定のマクロが実行されている間だけ存在します。マクロの実行が停止すると、そのマクロのすべてのローカルマクロ変数は存在しなくなります。

次の図に、次のプログラムの実行中に作成されるローカルシンボルテーブルを示します。

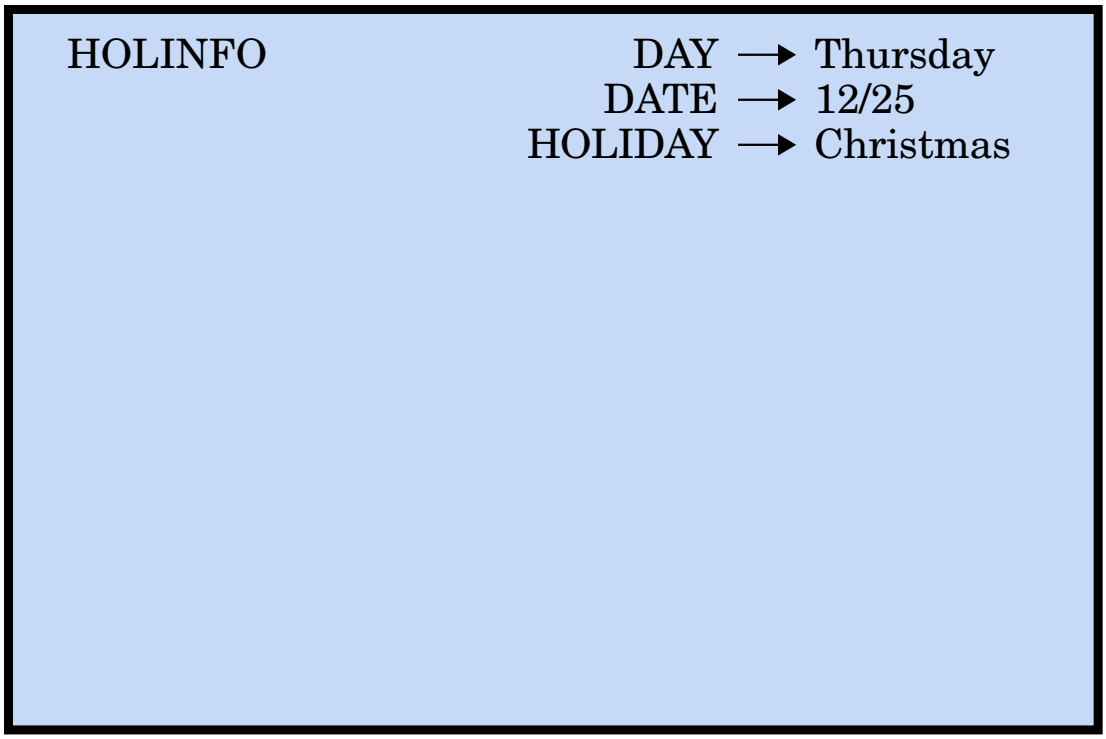
```
%macro holinfo(day,date);
%let holiday=Christmas;
%put *** Inside macro: ***;
%put *** &holiday occurs on &day, &date, 2002. ***;
%mend holinfo;

%holinfo(Wednesday,12/25)

%put *** Outside macro: ***;
%put *** &holiday occurs on &day, &date, 2002. ***;
```

%PUT ステートメントは、次のメッセージを SAS ログに書き込みます。

```
*** Inside macro: ***
*** Christmas occurs on Wednesday, 12/25, 2002. ***

*** Outside macro: ***
WARNING: Apparent symbolic reference HOLIDAY not resolved.WARNING: Apparent symbolic ref
```

このログから分かるように、ローカルマクロ変数 DAY、DATE、と HOLIDAY はマクロ内で置換されていますが、マクロの外部では、それらの変数が存在しないため置換されていません。

図 5.2 ローカルシンボルテーブル



マクロのローカルシンボルテーブルは、マクロによって少なくとも 1 つのマクロ変数が作成されるまでは、空です。ローカルシンボルテーブルは、次のいずれかによって作成できます。

- 1 つ以上のマクロパラメータの存在
- %LOCAL ステートメント
- %LET ステートメントや反復する%DO ステートメントなどの、マクロ変数を定義するマクロステートメント(ただし、その変数がまだグローバルとして存在しないか、その変数に対して%GLOBAL ステートメントが使用されていない場合)

注: マクロパラメータは、それが定義されているマクロに対して常にローカルです。マクロパラメータをグローバルにすることはできません。(ただし、パラメータの値をグローバル変数に割り当てることはできます。“ローカル変数の値に基づくグローバル変数の作成” (64 ページ)を参照してください。)

あるマクロを別のマクロの内部で呼び出すと、ネストされたスコープが作成されます。ネストされるマクロのレベル数に制限はないため、プログラムには、任意のレベル数でネストされたスコープを含めることができます。

%SYMLOCAL 関数を使用して、既存のマクロ変数がローカルシンボルテーブル内に存在するかどうかを表示できます。詳細については、“%SYMLOCAL 関数” (266 ページ)を参照してください。

SAS ログへのシンボルテーブルのコンテンツの書き込み

マクロの開発中に、グローバルシンボルテーブルやローカルシンボルテーブルのコンテンツのすべて、またはその一部を、SAS ログに書き込むと役に立つ場合がありま

す。これを実行するには、次のオプションのいずれかを指定して%PUT ステートメントを使用します。

ALL
スコープに関係なく、現在定義されているすべてのマクロ変数を表示します。この出力には、ユーザー定義のグローバル変数とローカル変数、および自動マクロ変数が含まれます。スコープは、最も内側から外側に向かう順序で表示されます。

AUTOMATIC
すべての自動マクロ変数を表示します。スコープは、AUTOMATIC として表示されます。すべての自動マクロ変数は、SYSPBUFF を除いてグローバルです。特定の自動マクロ変数の詳細については、“自動マクロ変数” (166 ページ) を参照してください。

GLOBAL
マクロプロセッサによって作成されていない、すべてのグローバルマクロ変数に関する説明を出力します。スコープは、GLOBAL として表示されます。自動マクロ変数は表示されません。

LOCAL
現在実行中のマクロ内で定義された、ユーザー定義のローカルマクロ変数を表示します。スコープは、そのマクロ変数が定義されているマクロの名前で表示されます。

USER
スコープに関係なく、ユーザー定義のマクロ変数をすべて表示します。スコープは、グローバルマクロ変数の場合は GLOBAL として表示され、そうでない場合は、そのマクロ変数が定義されているマクロの名前として表示されます。

たとえば、次のプログラムについて考えます。

```
%let origin=North America;

%macro dogs(type=);
data _null_;
set all_dogs;
where dogtype="&type" and dogorig="&origin";
put breed " is for &type.";
run;

%put _user_;
%mend dogs;

%dogs(type=work)
```

%MEND ステートメントの前に記述された%PUT ステートメントによって、ユーザーが作成したすべてのマクロ変数のスコープ、名前、および値が、次のように SAS ログに書き込まれます。

```
DOGS TYPE work
GLOBAL ORIGIN North America
```

TYPE は、マクロパラメータであるためマクロ DOGS に対してローカルであり、その値は **work** です。ORIGIN は、オープンコードで定義されているため、グローバルです。

マクロ変数の割り当て方法と置換方法

マクロプロセッサは、変数を作成し、それに値を割り当てたり変数を置換したりする前に、シンボルテーブルを検索して、その変数がすでに存在するかどうかを判定します。検索は、最もローカルなスコープから始まり、必要に応じてグローバルスコープに向けて外側に進みます。変数の割り当てや置換の要求は、オープンコード(マクロの外部)またはマクロの内部のマクロ変数参照で発生します。

次の図は、マクロ変数参照での変数の作成や値の割り当ての要求をマクロプロセッサが受信したときの、使用される検索順序を示しています。その次の図は、マクロ変数参照を置換する処理を示しています。これらの図は、いずれも最も基本的な検索のタイプを表しており、%LOCAL ステートメントを使用したり、CALL SYMPUT によって変数を作成したりする場合のような特殊なケースには適用されません。

図 5.3 マクロ変数の割り当て時または作成時の検索順序

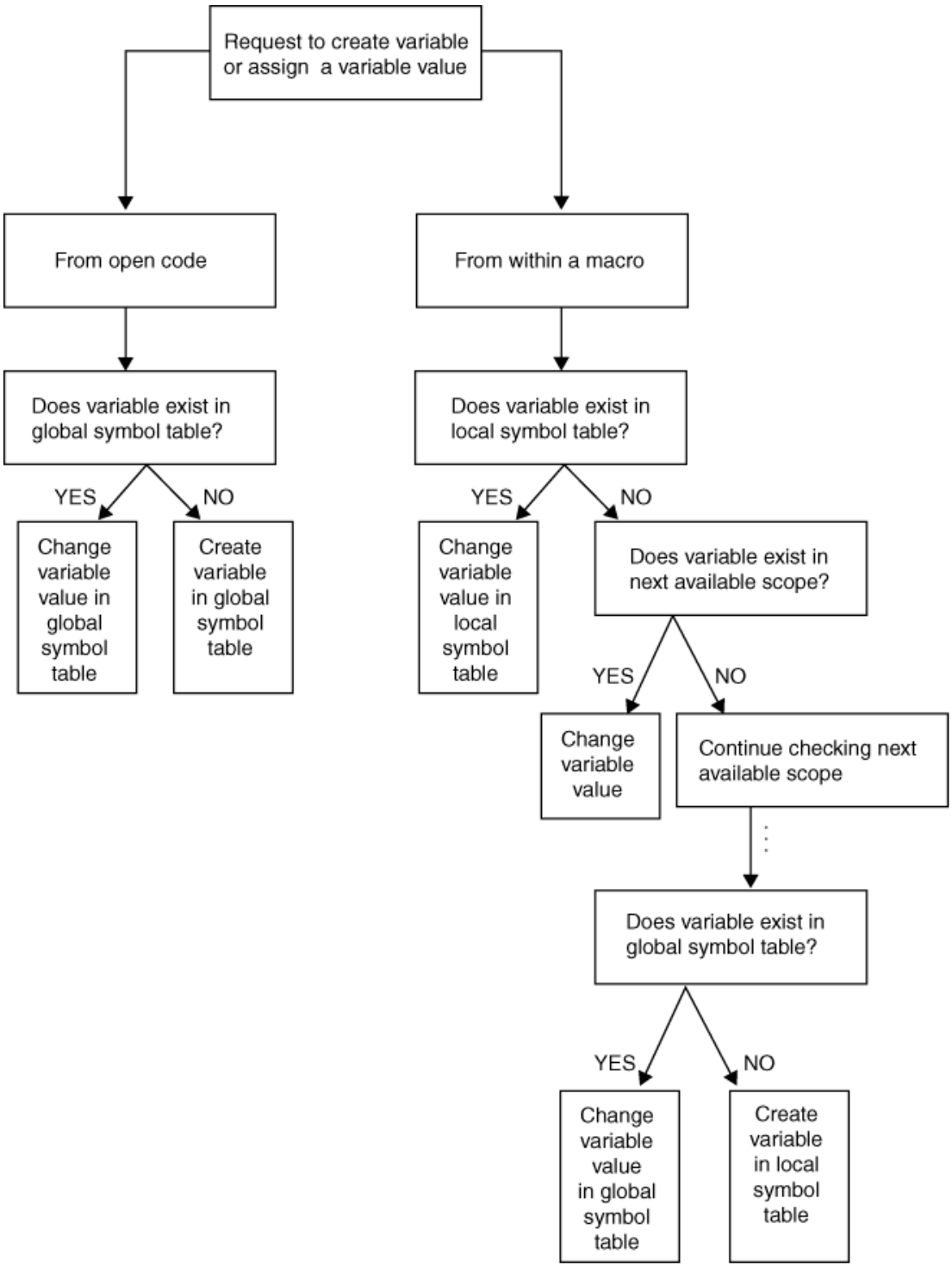
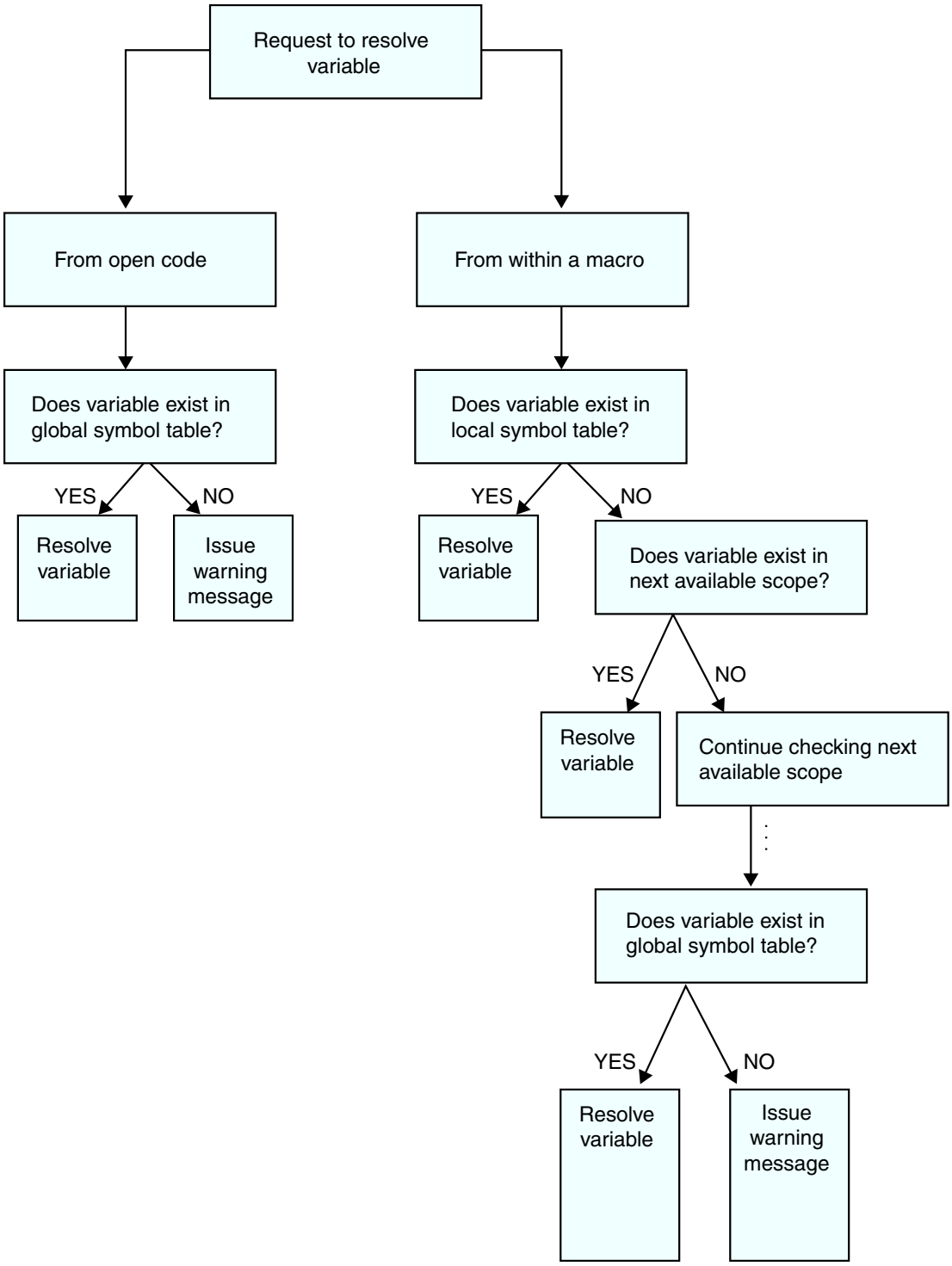


図 5.4 マクロ変数参照を置換するときの検索順序



マクロ変数のスコープの例

既存のマクロ変数の値の変更

マクロプロセッサは、マクロ変数を作成できるマクロプログラムステートメント(%LET ステートメントなど)を実行するときに、新しいマクロ変数を作成するのではなく、既存のマクロ変数の値を変更しようとします。ただし、%GLOBAL ステートメントと%LOCAL ステートメントを除きます。

説明のため、次の%LET ステートメントについて考えます。2 つの%LET ステートメントによって、マクロ変数 NEW に値を割り当てています。

```
%let new=inventry;
%macro name1;
%let new=report;
%mend name1;
```

次のステートメントをサブミットしたとします。

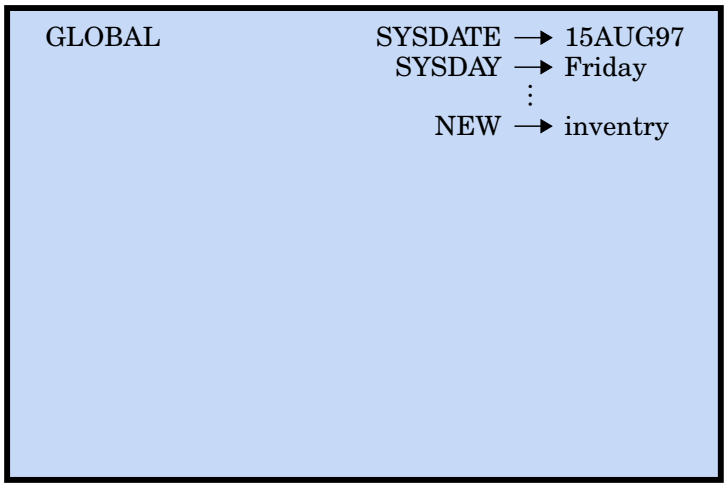
```
%name1

data &new;

data report;
```

NEW がグローバル変数として存在しているため、マクロプロセッサは新しい変数を作成せず、この変数の値を変更します。マクロ NAME1 のローカルシンボルテーブルは空のままです。

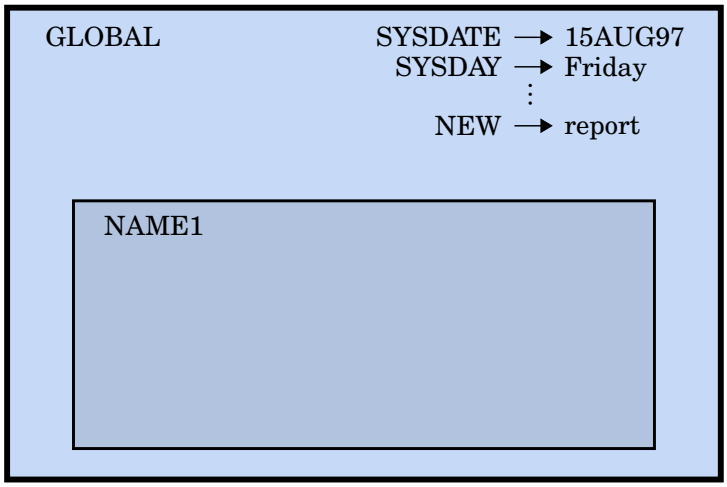
次の図は、NAME1 の実行前、実行中、実行後の、グローバルシンボルテーブルとローカルシンボルテーブルのコンテンツを示しています。

図 5.5 シンボルテーブルのスナップショット

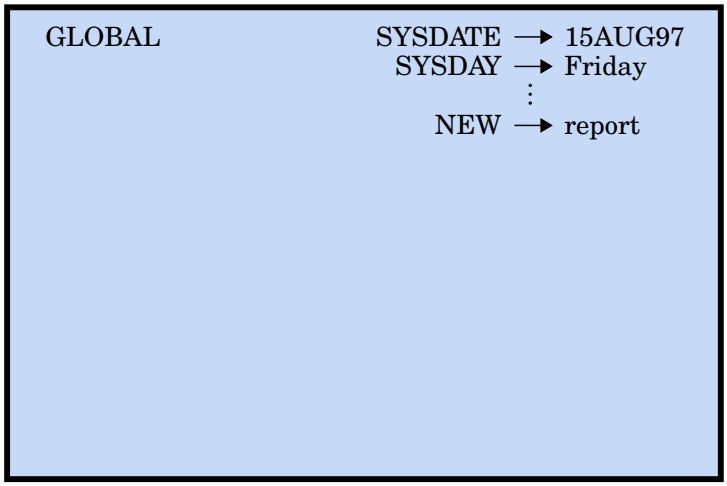
Before NAME1 executes



While NAME1 executes



After NAME1 executes



ローカル変数の作成

マクロプロセッサは、マクロ変数を作成できるマクロプログラムステートメントを実行するときに、使用可能な同じ名前のマクロ変数が存在しなければ、その変数をローカルシンボルテーブルに作成します。次の例について考えます。

```
%let new=inventry;
%macro name2;
%let new=report;
%let old=warehse;
%mend name2;

%name2

data &new;
set &old;
run;
```

NAME2 が実行されると、SAS コンパイラはその後のステートメントを次のように解釈します。

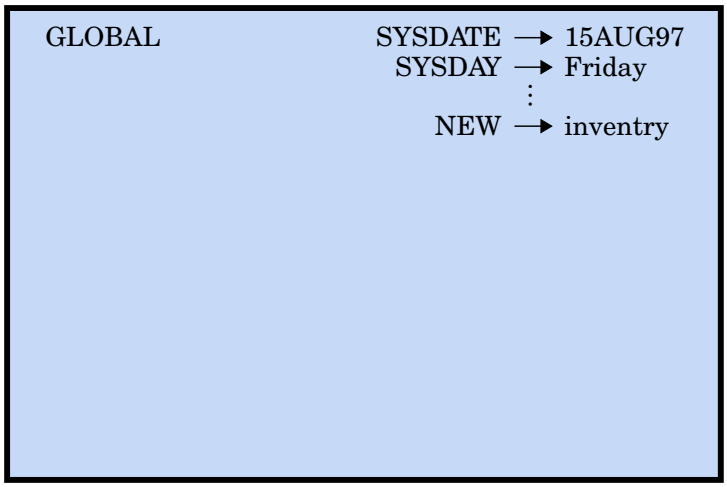
```
data report;
set &old;
run;
```

マクロ NAME2 の実行終了後、マクロプロセッサは&OLD 参照を検出します。このとき、マクロ変数 OLD は存在しません。マクロプロセッサは参照を置換できないため、警告メッセージを発行します。

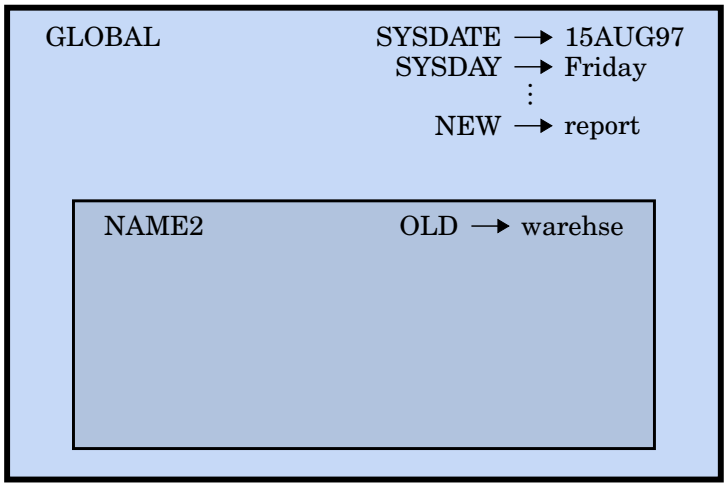
次の図は、さまざまなステージでのグローバルシンボルテーブルとローカルシンボルテーブルのコンテンツを示しています。

図 5.6 さまざまなステージでのシンボルテーブル

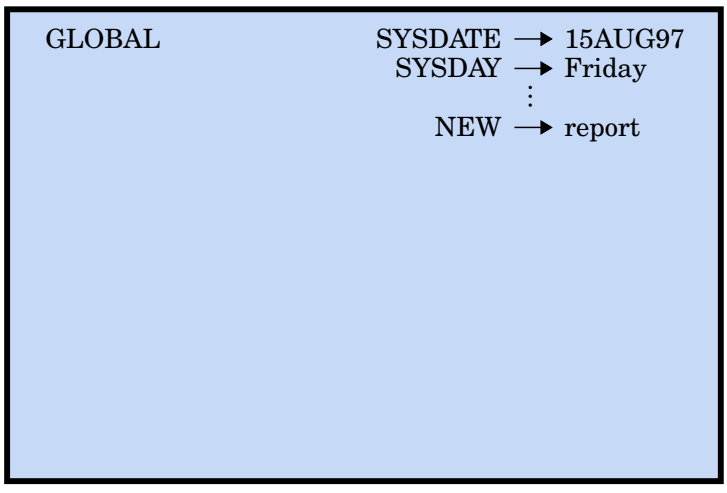
Before NAME2 executes



While NAME2 executes



After NAME2 executes



一方、次のプログラムのように、マクロ NAME2 の内部に SAS ステートメントを配置したとします。

```
%let new=inventry;
```

```
%macro name2;
%let new=report;
%let old=warehse;
data &new;
set &old;
run;
%mend name2;

%name2
```

この場合、マクロプロセッサは、NAME2 の実行中に SET ステートメントを生成するため、NAME2 のローカルシンボルテーブルで OLD を検出します。したがって、このマクロを実行すると次のステートメントが生成されます。

```
data report;
set warehse;
run;
```

ネストのレベル数に関係なく、同じルールが適用されます。次の例について考えます。

```
%let new=inventry;
%macro conditn;
%let old=sales;
%let cond=cases>0;
%mend conditn;

%macro name3;
%let new=report;
%let old=warehse;
%conditn
data &new;
set &old;
if &cond;
run;
%mend name3;

%name3
```

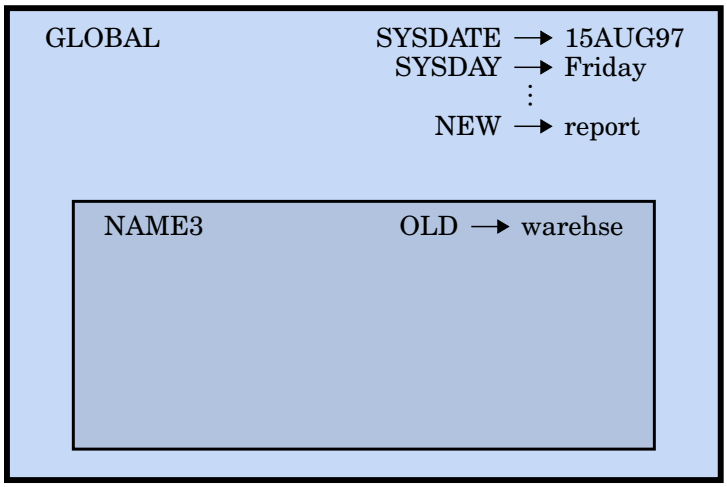
マクロプロセッサは、次のステートメントを生成します。

```
data report;
set sales;
if &cond;
run;
```

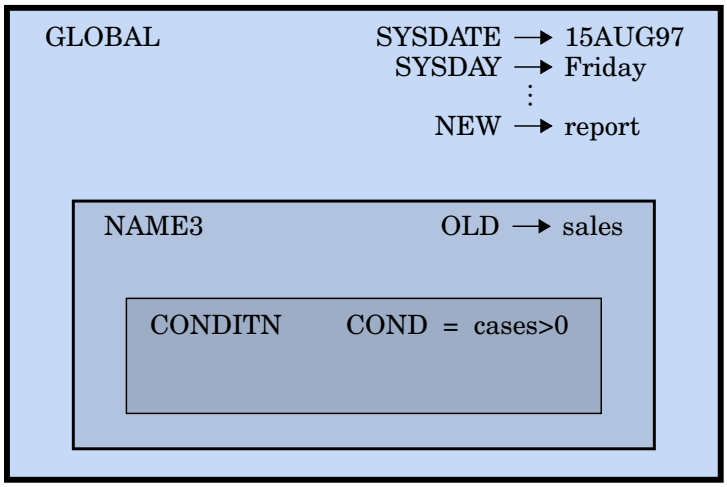
マクロプロセッサが&COND 参照に到達する前に CONDITN の実行が終了しているため、マクロプロセッサが&COND 参照を置換しようとしたときには、すでに COND という名前の変数は存在しません。したがって、マクロプロセッサは警告メッセージを発行し、定数テキストの一部として未置換の参照を生成します。次の図に、各ステップでのシンボルテーブルを示します。

図 5.7 2 つのレベルのネストを示すシンボルテーブル

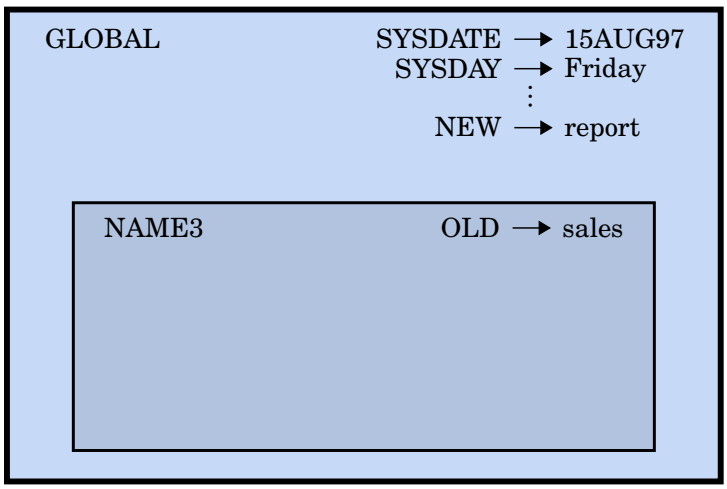
Early execution of NAME3, before CONDITN executes



While NAME3 and CONDITN execute



Late execution of NAME3, after CONDITN executes



マクロ呼び出しの配置は、ネストされたスコープを作成することであって、マクロ定義の配置ではありませんので注意してください。たとえば、NAME3 の内部で CONDITN を

呼び出すと、ネストされたスコープが作成されます。NAME3 の内部で CONDITN を定義する必要はありません。

マクロ変数をローカルにする

マクロプロセッサに、既存のマクロ変数の値を変更させるのではなく、確実にローカルマクロ変数を作成させる必要がある場合があります。このような場合、%LOCAL ステートメントを使用してマクロ変数を作成します。

マクロの実行停止後にマクロ変数の値が必要でない場合、マクロ内に作成するすべてのマクロ変数を必ずローカルにします。マクロ変数の値を誤って変更する可能性を最小限にすると、大規模なマクロプログラムのデバッグが容易になります。また、ローカルマクロ変数を定義するマクロの実行が終了すると、それらのローカルマクロ変数は存在しませんが、グローバルマクロ変数は SAS セッションが存続する限り存在します。したがって、ローカル変数を使用すると、ストレージ全体の使用量が減ります。

たとえば、次に示すように、マクロ NAMELST を使用して VAR ステートメント用の名前のリストを作成するとします。

```
%macro namelst(name,number);
%do n=1 %to &number;
&name&n
%end;
%mend namelst;
```

次のプログラムで NAMELST を呼び出します。

```
%let n=North State Industries;

proc print;
var %namelst(dept,5);
title "Quarterly Report for &n";
run;
```

このマクロを実行すると、SAS コンパイラは各ステートメントを次のように解釈します。

```
proc print;
var dept1 dept2 dept3 dept4 dept5;
title "Quarterly Report for 6";
run;
```

マクロプロセッサは、%DO ループの反復を実行するたびに、グローバル変数 N の値を変更します。(ループの実行が停止すると、N の値は 6 になります。これについては、“%DO ステートメント” (294 ページ) で説明されています。)競合を避けるには、次に示すように、%LOCAL ステートメントを使用してローカル変数 N を作成します。

```
%macro namels2(name,number);
%local n;
%do n=1 %to &number;
&name&n
%end;
%mend namels2;
```

ここで、次のように同じプログラムを実行します。

```
%let n=North State Industries;

proc print;
var %namels2(dept,5);
title "Quarterly Report for &n";
run;
```

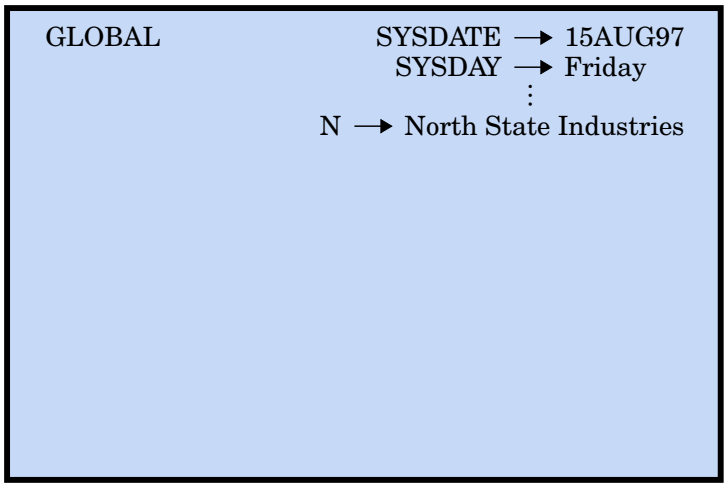
マクロプロセッサは、次のステートメントを生成します。

```
proc print;
var dept1 dept2 dept3 dept4 dept5;
title "Quarterly Report for North State Industries";
run;
```

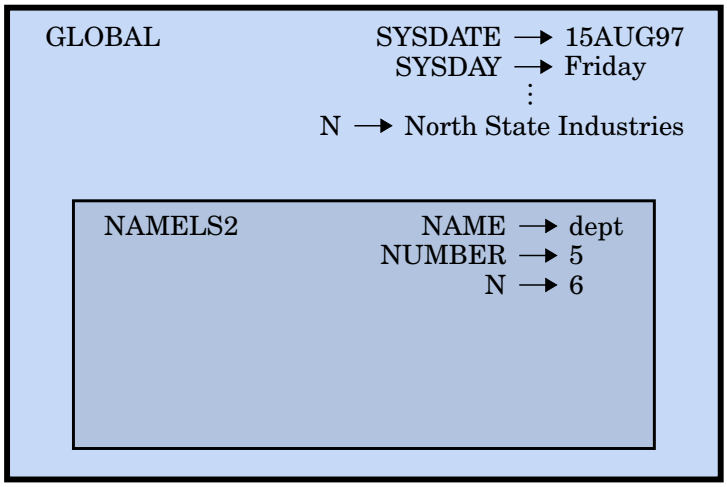
次の図に、NAMELS2 の実行前、NAMELS2 の実行中、およびマクロプロセッサが TITLE ステートメントで&N 参照を検出したときのシンボルテーブルを示します。

図 5.8 同じ名前を持つグローバル変数とローカル変数のシンボルテーブル

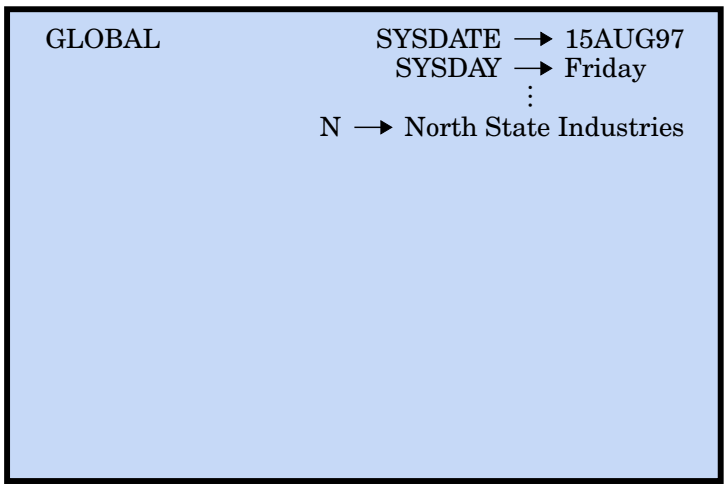
Before NAMELS2 executes



While NAMELS2 executes (at end of last iteration of %DO loop)



After NAMELS2 executes



グローバルマクロ変数の作成

%GLOBAL ステートメントを使用すると、同じ名前の変数がまだそこに存在しなければ、現在のスコープとは無関係にグローバルマクロ変数が作成されます。

たとえば、次のプログラムでは、マクロ CONDITN に、マクロ変数 COND をグローバル変数として作成する%GLOBAL ステートメントが含まれています。

```
%macro conditn;
%global cond;
%let old=sales;
%let cond=cases>0;
%mend conditn;
```

このプログラムの他の部分を次に示します。

```
%let new=inventry;

%macro name4;
%let new=report;
%let old=warehse;
%conditn
data &new;
set &old;
if &cond;
run;
%mend name4;

%name4
```

NAME4 を呼び出すと、次のステートメントが生成されます。

```
data report;
set sales;
if cases>0;
run;
```

NAME4 の外部に SAS DATA ステップステートメントを配置するとします。この場合、マクロプロセッサが参照を置換するには、すべてのマクロ変数をグローバルにする必要があります。CONDITN の実行が開始される時点で、NAME4 の%LET ステートメントによって、すでに OLD が NAME4 に対するローカル変数として作成されているため、CONDITN の%GLOBAL ステートメントに OLD を追加することはできません。(%GLOBAL ステートメントを使用して、既存のローカル変数をグローバルにすることはできません。)

したがって、OLD をグローバルにするには、変数参照が現れる前の任意の場所で、次のマクロ NAME5 に示すように%GLOBAL ステートメントを使用します。

```
%let new=inventry;

%macro conditn;
%global cond;
%let old=sales;
%let cond=cases>0;
%mend conditn;

%macro name5;
%global old;
%let new=report;
```

```
%let old=warehse;
%conditn
%mend name5;

%name5

data &new;
set &old;
if &cond;
run;
```

ここで、NAME5 の%LET ステートメントでは、ローカル変数として OLD を作成するのではなく、既存のグローバル変数 OLD の値を変更しています。SAS コンパイラは、各ステートメントを次のように解釈します。

```
data report;
set sales;
if cases>0;
run;
```

ローカル変数の値に基づくグローバル変数の作成

パラメータなどのローカル変数をマクロの外部で使用するには、次のプログラムに示すように、ローカル変数の値を、%LET ステートメントを使用して別の名前のグローバル変数に割り当てます。

```
%macro namels3(name,number);
%local n;
%global g_number;
%let g_number=&number;
%do n=1 %to &number;
&name&n
%end;
%mend namels3;
```

ここで、マクロ NAMELS3 を次のプログラムから呼び出します。

```
%let n=North State Industries;

proc print;
var %namels3(dept,5);
title "Quarterly Report for &n";
footnote "Survey of &g_number Departments";
run;
```

コンパイラは、各ステートメントを次のように解釈します。

```
proc print;
var dept1 dept2 dept3 dept4 dept5;
title "Quarterly Report for North State Industries";
footnote "Survey of 5 Departments";
run;
```

CALL SYMPUT ルーチンを使用したスコープの特殊なケース

CALL SYMPUT ルーチンの概要

CALL SYMPUT に伴うほとんどの問題は、マクロ変数を作成する CALL SYMPUT ステートメントと、その変数を使用するマクロ変数参照との間に、正確なステップ境界が不足していることに関係しています。(詳細については、“CALL SYMPUT ルーチン” (228 ページ)を参照してください。) ただし、CALL SYMPUT が作成するマクロ変数のスコープに関連して、特殊なケースがいくつか存在します。これらのケースは、なぜ変数へのスコープの割り当てを SAS に任せず、常に自分で変数にスコープを割り当ててから値を割り当てる必要があるのかについての良い例になります。

次の 2 つのルールは、CALL SYMPUT が変数を作成する場所を制御します。

1. CALL SYMPUT は、DATA ステップの実行中に、現在使用可能なシンボルテーブルが空ではないという条件で、そのシンボルテーブルにマクロ変数を作成します。そのテーブルが空の(ローカルマクロ変数を含まない)場合、通常、CALL SYMPUT は最も近い空でないシンボルテーブルに変数を作成します。
2. ただし、ローカルシンボルテーブルが空であっても、CALL SYMPUT によってそのシンボルテーブルに変数が作成される、次の 3 つのケースがあります。
   - SAS バージョン 8 以降、PROC SQL の後で CALL SYMPUT を使用すると、ローカルシンボルテーブルに変数が作成されます。
   - マクロの呼び出し時にマクロ変数 SYSPBUFF を作成すると、ローカルシンボルテーブルに変数が作成されます。
   - 実行中のマクロに計算される%GOTO ステートメントが含まれる場合、ローカルシンボルテーブルに変数が作成されます。計算される%GOTO ステートメントは、&または%が含まれるラベルを使用する%GOTO ステートメントです。つまり、計算される%GOTO ステートメントには、テキスト式を生成するマクロ変数参照またはマクロ呼び出しが含まれます。計算される%GOTO ステートメントの例を次に示します。

     ```
     %goto &home;
     ```

現在 DATA ステップで使用可能なシンボルテーブルは、DATA ステップが完了したと SAS が判断したときに存在するシンボルテーブルです。(SAS は、RUN ステートメント、データ行の後のセミコロン、または別のステップの開始を検出すると、DATA ステップが完了したと見なします。)

簡単に言うと、実行中のマクロに計算される%GOTO ステートメントが含まれる場合、またはマクロの呼び出し時にマクロ変数 SYSPBUFF を作成した場合、ローカルシンボルテーブルが空であっても、空でない場合と同様に CALL SYMPUT が動作してローカルマクロ変数が作成されます。

CALL SYMPUT ルーチンがどのシンボルテーブルに変数を作成したかを調べるために、_USER_オプションを指定して%PUT ステートメントを使用すると役立つことがあります。

完全な DATA ステップと空でないローカルシンボルテーブルを用いた CALL SYMPUT の使用例

次の例について検討します。この例では、マクロ内に、CALL SYMPUT ステートメントを含む完全な DATA ステップが含まれています。

```
%macro env1(param1);
data _null_;
x = 'a token';
call symput('myvar1',x);
run;
%mend env1;

%env1(10)

data temp;
y = "&myvar1";
run;
```

これらのステートメントをサブミットすると、次のエラーメッセージが表示されます。

```
WARNING: Apparent symbolic reference MYVAR1 not resolved.
```

このメッセージは、DATA ステップが ENV1 の環境内で完了しており(つまり、RUN ステートメントがマクロに含まれており)、ENV1 のローカルシンボルテーブルが空でない(パラメータ PARAM1 が含まれる)ために表示されます。したがって、CALL SYMPUT ルーチンによって ENV1 のローカル変数として MYVAR1 が作成されます。この変数は、グローバルマクロ変数を期待するその後の DATA ステップでは使用できません。

スコープを表示するには、_USER_オプション付きの%PUT ステートメントをマクロに追加し、同様のステートメントをオープンコードにも追加します。ここで、前と同様に次のマクロを呼び出します。

```
%macro env1(param1);
data _null_;
x = 'a token';
call symput('myvar1',x);
run;

%put ** Inside the macro: **;
%put _user_;
%mend env1;

%env1(10)

%put ** In open code: **;
%put _user_;

data temp;
y = "&myvar1"; /* ERROR - MYVAR1 is not available in open code. */
run;
```

これらの%PUT _USER_ステートメントを実行すると、次の情報が SAS ログに書き込まれます。

```
** Inside the macro: **
ENV1 MYVAR1 a token
ENV1 PARAM1 10
```

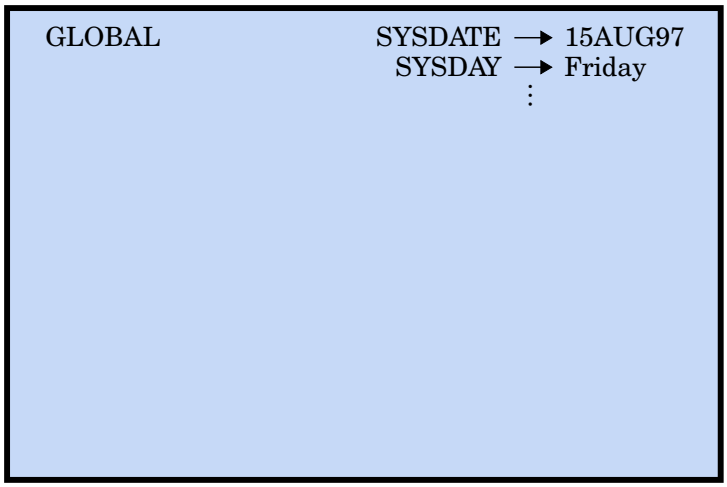
```
** In open code: **
```

CALL SYMPUT によって、ENV1 のローカルシンボルテーブルに、マクロ変数 MYVAR1 が作成されます。グローバルマクロ変数が作成されないため、オープンコード内の%PUT _USER_ステートメントは、SAS ログに何も書き込みません。

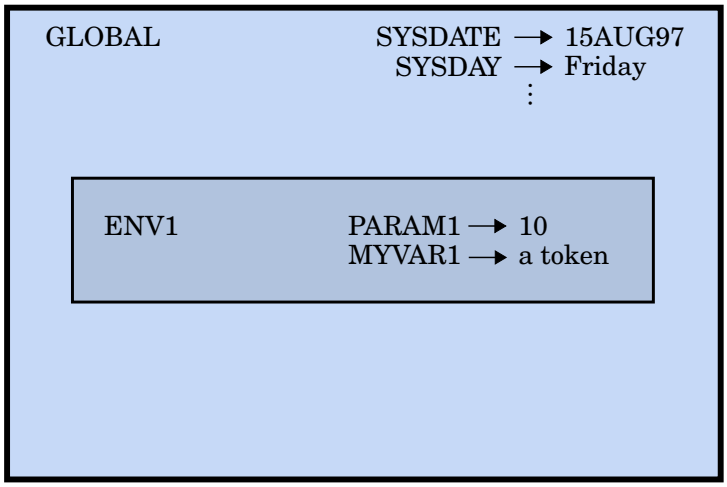
次の図に、この例のすべてのシンボルテーブルを示します。

図 5.9 完全な DATA ステップを生成する CALL SYMPUT ルーチンを使用したシンボルテーブル

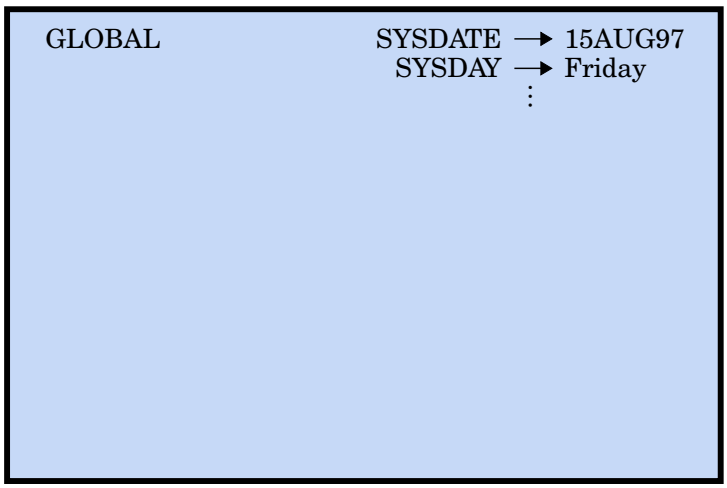
Before ENV1 executes



While ENV1 executes



After ENV1 executes



不完全な DATA ステップを用いた CALL SYMPUT の使用例

次に示すマクロ ENV2 に含まれる DATA ステップは、RUN ステートメントがないため不完全です。

```
%macro env2(param2);
data _null_;
x = 'a token';
call symput('myvar2',x);
%mend env2;

%env2(20)
run;

data temp;
y="&myvar2";
run;
```

これらのステートメントは、エラーを出力せずに実行されます。DATA ステップは、SAS が(この場合はオープンコード内の)RUN ステートメントを検出して、初めて完全になります。したがって、DATA ステップの現在のスコープは、グローバルスコープです。CALL SYMPUT によってグローバルマクロ変数として MYVAR2 が作成され、その後の DATA ステップでこの値を使用できます。

ここでも、スコープを表示するために、次のように_USER_オプション付きで%PUT ステートメントを使用します。

```
%macro env2(param2);
data _null_;
x = 'a token';
call symput('myvar2',x);

%put ** Inside the macro: **;
%put _user_;
%mend env2;

%env2(20)

run;

%put ** In open code: **;
%put _user_;

data temp;
y="&myvar2";
run;
```

ENV2 内の%PUT _USER_ステートメントが実行されると、次のメッセージが SAS ログに書き込まれます。

```
** Inside the macro: **
ENV2 PARAM2 20
```

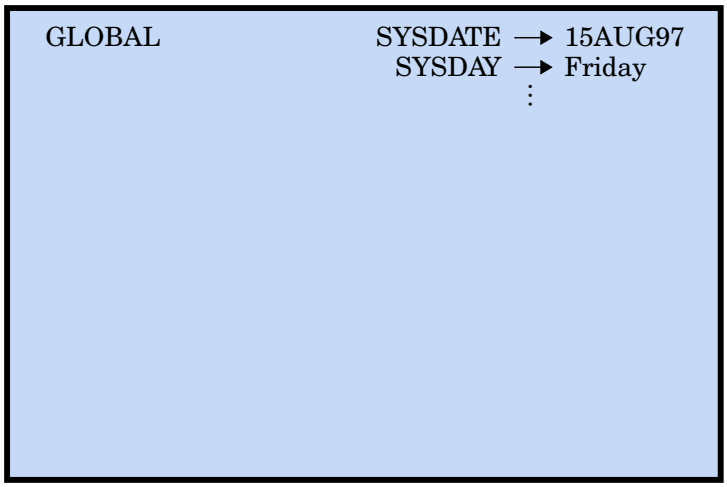
オープンコード内の%PUT _USER_ステートメントによって、次のメッセージが SAS ログに書き込まれます。

```
** In open code: **
GLOBAL MYVAR2 a token
```

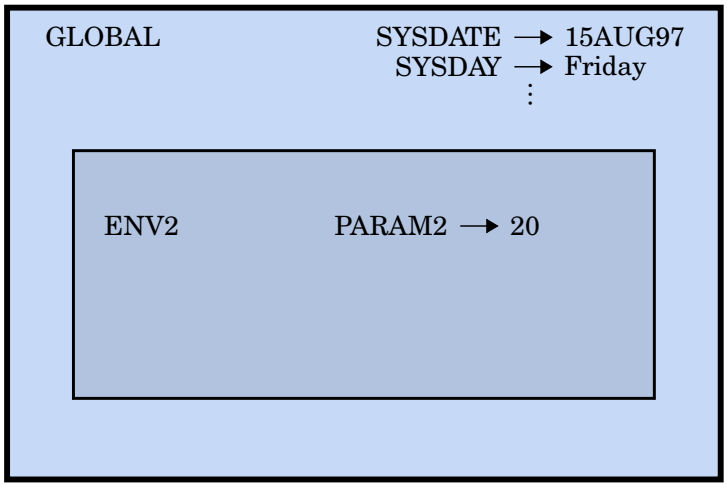
次の図に、この例に含まれるすべてのスコープを示します。

図 5.10 *不完全な DATA ステップを生成する CALL SYMPUT ルーチンを使用したシンボルテーブル*

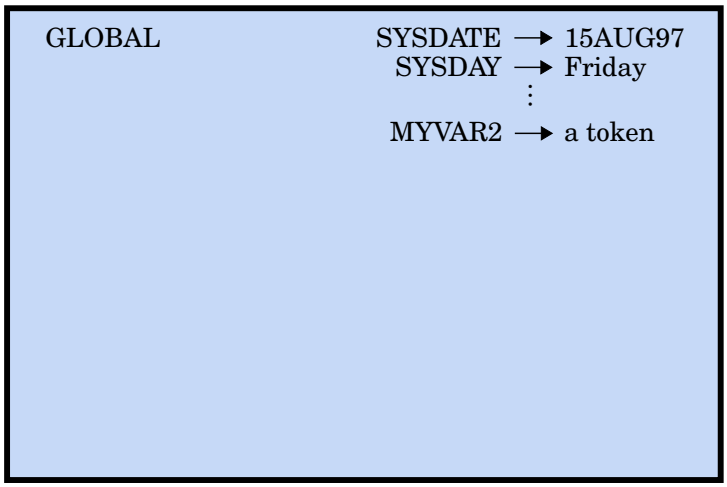
Before ENV2 executes



While ENV2 executes



After ENV2 executes



完全な DATA ステップと空のローカルシンボルテーブルを用いた CALL SYMPUT の使用例

次の例の ENV3 は、マクロパラメータを使用していません。そのため、ローカルシンボルテーブルは空です。

```
%macro env3;
data _null_;
x = 'a token';
call symput('myvar3',x);
run;

%put ** Inside the macro: **;
%put _user_;
%mend env3;

%env3

%put ** In open code: **;
%put _user_;

data temp;
y="&myvar3";
run;
```

この場合、DATA ステップは完全であるためマクロ内で実行されますが、ローカルシンボルテーブルは空です。そのため、CALL SYMPUT は、最も近く空でない使用可能なシンボルテーブル、つまりグローバルシンボルテーブルに MYVAR3 を作成します。どちらの%PUT ステートメントも、次のように、MYVAR3 がグローバルシンボルテーブルに存在することを表示しています。

```
** Inside the macro: **
GLOBAL MYVAR3 a token

** In open code: **
GLOBAL MYVAR3 a token
```

SYSPBUFF と空のローカルシンボルテーブルを用いた CALL SYMPUT の使用例

次の例では、自動マクロ変数 SYSPBUFF の存在によって、パラメータとローカルマクロ変数が両方ともマクロ ENV4 に存在しない場合でも、ローカルシンボルテーブルが空でない場合と同様に CALL SYMPUT が動作します。

```
%macro env4 /parmbuff;
data _null_;
x = 'a token';
call symput('myvar4',x);
run;

%put ** Inside the macro: **;
%put _user_;
%put &syspbuff;
%mend env4;

%env4
```

```
%put ** In open code: **;
%put _user_;
%put &syspbuff;

data temp;
y="&myvar4"; /* ERROR - MYVAR4 is not available in open code */
run;
```

/PARMBUFF を指定することによって、自動マクロ変数 SYSPBUFF が作成されます。そのため、マクロ ENV4 を呼び出すと、パラメータとローカルマクロ変数が両方ともマクロ ENV4 に存在しない場合でも、CALL SYMPUT によって、ENV4 のローカルシンボルテーブルにマクロ変数 MYVAR4 が作成されます。

このことは、%PUT ステートメントの結果で証明されています。つまり、MYVAR4 のスコープが ENV4 として示されており、SYSPBUFF が ENV4 に対してローカルであるため、オープンコードの%PUT ステートメントでの SYSPBUFF への参照は置換されていません。

```
** Inside the macro: **
b ENV4 MYVAR4 a token

** In open code: **
WARNING: Apparent symbolic reference SYSPBUFF not resolved.
```

詳細については、“SYSPBUFF 自動マクロ変数” (214 ページ)を参照してください。

6 章

マクロ式

マクロ式

マクロ式には、テキスト式、論理式、演算式という 3 つのタイプがあります。*テキスト式*は、テキスト、マクロ変数、マクロ関数、またはマクロ呼び出しの任意の組み合わせです。テキスト式は、置換されてテキストを生成します。次に、テキスト式の例をいくつか示します。

- &BEGIN
- %GETLINE
- &PREFIX.PART&SUFFIX
- %UPCASE(&ANSWER)

*論理式*と*演算式*は、評価されて結果を生成する一連の命令から成る、演算子とオペランドの並びです。演算式には、算術演算子が含まれます。論理式には、論理演算子が含まれます。次の表に、単純な演算式と論理式の例を示します。

表 6.1 *演算式と論理式*

演算式	論理式
1 + 2	&DAY = FRIDAY
4 * 3	A < a

演算式	論理式
4 / 2	1 < &INDEX
00FFx - 003Ax	&START NE &END

演算式と論理式の定義

演算式と論理式の評価

特定のマクロ関数やステートメントで、演算式および論理式を使用できます。次の表を参照してください。これらの関数やステートメントにおいて演算式および論理式を使用して、マクロの実行時に生成されるテキストを制御できます。

表 6.2 *演算式と論理式を評価するマクロ言語要素*

%DO*macro-variable=expression* %TO *expression*<%BY *expression*>;
%DO %UNTIL(*expression*);
%DO %WHILE(*expression*);
%EVAL (*expression*);
%IF *expression* %THEN *statement*;
%QSCAN(*argument,expression*<,*delimiters*>)
%QSUBSTR(*argument,expression*<,expression>)
%SCAN(*argument,expression*,<delimiters>)
%SUBSTR(*argument,expression*<,expression>)
%SYSEVALF(*expression,conversion-type*)

テキスト式を使用して、演算式または論理式の一部または全部を生成することができます。マクロプロセッサは、テキスト式を置換してから、演算式と論理式を評価します。たとえば、次のステートメントをサブミットすると、マクロプロセッサは、%EVAL 関数においてマクロ変数&A、&B、および&OPERATOR を置換してから式 2 + 5 を評価します。

```
%let A=2;
%let B=5;
%let operator=+;
%put The result of &A &operator &B is %eval(&A &operator
&B).;
```

これらのステートメントをサブミットすると、%PUT ステートメントによって次のメッセージがログに書き込まれます。

```
The result of 2 + 5 is 7.
```

オペランドと演算子

演算式または論理式の*オペランド*は、常にテキストです。ただし、数値を表すオペランドは、式が評価されるときに一時的に数値に変換できます。デフォルトでは、マクロプロセッサは整数演算を実行します。そのため、整数を表す整数値および 16 進値のみが、数値に変換可能です。ピリオドの文字を含むオペランド(たとえば、1.0)は変換されません。ただし、%SYSEVALF 関数を除きます。この関数は、引数に含まれるピリオドの文字を小数点として解釈し、そのオペランドを、使用しているオペレーティングシステムでの浮動小数点値に変換します。

注: 数値式の値は、–2**64 から 2**64–1 までの範囲に制限されます。

マクロ式の*演算子*は、DATA ステップに含まれる演算子のサブセットです(表 6.3 (75 ページ)参照)。ただし、マクロ言語には MAX 演算子や MIN 演算子がなく、DATA ステップが認識する':'をマクロ言語は認識しません。マクロ言語では、式が評価されたときに実行される演算の順序は、DATA ステップと同じです。かっこ内の演算が最初に実行されます。

注: 比較演算子に囲まれたマクロ式を含む式(10<&X<20 など)は、DATA ステップの複合式と等価である場合があります(ただし、式の置換結果によって変わります)。安全のためには、演算子の結合を、式 10<&X AND &X<20 のように指定します。

表 6.3 *マクロ言語演算子*

演算子	ニーモニック	優先順位	定義	例
**		1	累乗	2**4
+		2	正の値を表す接頭語	+(A+B)
-		2	負値を表す接頭語	-(A+B)
¬^~	NOT	3	論理否定*	NOT A
*		4	乗算	A*B
/		4	除算	A/B
+		5	加算	A+B
-		5	減算	A-B
<	LT	6	より小さい	A<B
<=	LE	6	以下	A<=B
=	EQ	6	等しい	A=B
#	IN	6	リスト内のいずれかと等しい**	A#B C D E

演算子	ニーモニック	優先順位	定義	例
¬= ^= ~=	NE	6	等しくない*	A NE B
>	GT	6	より大きい	A>B
>=	GE	6	以上	A>=B
&	AND	7	論理積	A=B & C=D
|	OR	8	論理和	A=B | C=D

* 使用するシンボルはキーボードによって異なります。

** リスト要素のデフォルトの区切り文字は空白です。詳細については、“MINDELIMITER= システムオプション” (349 ページ)を参照してください。

** IN (#)演算子を使用する前に、“MINOPERATOR システムオプション” (351 ページ)を参照してください。

** IN 演算子を使用する場合、両方のオペランドが値を含んでいる必要があります。オペランドに null 値が含まれていると、エラーが発生します。

注意:

整数式が累乗演算子、乗算演算子、または除算演算子を含み、-9,007,199,254,740,992 から 9,007,199,254,740,992 までの範囲を超える値を使用または計算すると、不正確な結果が得られる場合があります。

マクロプロセッサによる演算式の評価方法

数値オペランドの評価

マクロ機能は、文字列を処理する機能です。ただし、特定の状況において、マクロプロセッサは数値を表すオペランドを数値として評価できます。マクロプロセッサは、算術演算子および数値を表すオペランドを含む式を評価するときに、一時的にそのオペランドを数値に変換してから整数算術演算を実行します。評価結果はテキストになります。

デフォルトでは、ほとんどのマクロステートメントおよびマクロ関数における算術演算は、整数演算を使用して評価されます。ただし、%SYSEVALF 関数を除きます。詳細については、“浮動小数点オペランドの評価” (77 ページ)を参照してください。次のマクロステートメントで、整数演算の評価を説明します。

```
%let a=%eval(1+2);
%let b=%eval(10*3);
%let c=%eval(4/2);
%let i=%eval(5/3);
%put The value of a is &a;
%put The value of b is &b;
%put The value of c is &c;
%put The value of I is &i;
```

これらのステートメントをサブミットすると、次のメッセージがログに表示されます。

```
The value of a is 3
The value of b is 30
```

```
The value of c is 2
The value of I is 1
```

最後のステートメントの結果に注目してください。整数に対して、通常は小数を含む結果が得られるはずの除算を実行すると、整数演算によって小数部が切り捨てられます。

マクロプロセッサは、文字オペランドを含む整数演算式を評価すると、エラーを生成します。整数値または 16 進値を表す文字を含むオペランドのみが、数値に変換されます。次のステートメントは、誤った使用方法を示しています。

```
%let d=%eval(10.0+20.0); /*INCORRECT*/
```

%EVAL 関数が整数演算のみをサポートしているため、マクロプロセッサはピリオドを含む値を数値に変換せず、このオペランドは文字オペランドとして評価されます。このステートメントによって次のエラーメッセージが生成されます。

```
ERROR: A character operand was found in the %EVAL function or %IF
condition where a numeric operand is required. The condition was:
10.0+20.0
```

浮動小数点オペランドの評価

%SYSEVALF 関数は、浮動小数点値を表すオペランドを含む演算式を評価します。たとえば、%SYSEVALF 関数に与えられた次の式は、浮動小数点演算を使用して評価されます。

```
%let a=%sysevalf(10.0*3.0);
%let b=%sysevalf(10.5+20.8);
%let c=%sysevalf(5/3);
%put 10.0*3.0 = &a;
%put 10.5+20.8 = &b;
%put 5/3 = &c;
```

%PUT ステートメントによって、次のメッセージがログに表示されます。

```
10.0*3.0 = 30
10.5+20.8 = 31.3
5/3 = 1.6666666667
```

%SYSEVALF 関数は、演算式を評価するときに、数値を表すオペランドを一時的に浮動小数点値に変換します。評価結果は、浮動少数点値で表される場合がありますが、整数演算式と同様に常にテキストです。

%SYSEVALF 関数には、BOOLEAN、INTEGER、CEIL、および FLOOR という変換タイプ仕様が用意されています。たとえば、次の%PUT ステートメントは、それぞれ 1、2、3、および 2 を返します。

```
%let a=2.5;
%put %sysevalf(&a,boolean);
%put %sysevalf(&a,integer);
%put %sysevalf(&a,ceil);
%put %sysevalf(&a,floor);
```

これらの変換タイプは、整数値やブール値を必要とする他のマクロ式で使用できるように、%SYSEVALF が返す値を調整します。

注意:

%SYSEVALF 関数に対して、変換タイプを指定してください。 %SYSEVALF 関数をマクロ式で使用した場合、または%SYSEVALF 関数の結果を他のマクロ式で使用されるマクロ変数に割り当てた場合、%SYSEVALF 関数が欠損値または浮動小数点値を返すと、エラーまたは予期しない結果が生じる恐れがあります。エラーを防ぐ

には、他のマクロ式と互換性のある値を返す変換タイプを指定します。変換タイプの使用の詳細については、“%SYSEVALF 関数” (267 ページ)を参照してください。

マクロプロセッサによる論理式の評価方法

論理式での数値オペランドの比較

論理(つまりブール)式は、true または false として評価される値を返します。マクロ言語では、0 以外のすべての数値は true になり、0 の値は false になります。

マクロプロセッサは、数値を表すオペランドを含む論理式を評価するときに、その文字を一時的に数値に変換します。マクロプロセッサによって数値オペランドを含む論理式がどのように評価されるかを説明するために、次のマクロ定義について考えます。

```
%macro compnum(first,second);
%if &first>&second %then %put &first is greater than &second;
%else %if &first=&second %then %put &first equals &second;
%else %put &first is less than &second;
%mend compnum;
```

次の値を使用して、マクロ COMPNUM を呼び出します。

```
%compnum(1,2)
%compnum(-1,0)
```

次の結果がログに表示されます。

```
1 is less than 2
-1 is less than 0
```

この結果は、論理式のオペランドが数値として評価されたことを示しています。

浮動小数点値または欠損値の比較

浮動小数点値または欠損値含む論理式を評価するには、%SYSEVALF 関数を使用する必要があります。浮動小数点値と欠損値を含む比較を説明するために、渡されたパラメータを%SYSEVALF 関数を使用して比較し、その結果をログに出力する次のマクロについて考えます。

```
%macro compflt(first,second);
%if %sysevalf(&first>&second) %then %put &first is greater than
&second;
%else %if %sysevalf(&first=&second) %then %put &first equals
&second;
%else %put &first is less than &second;
%mend compflt;
```

次の値を使用してマクロ COMPFLT を呼び出します。

```
%compflt (1.2,.9)
%compflt (-.1,.)
%compflt (0,.)
```

次の値がログに書き込まれます。

```
1.2 is greater than .9
-.1 is greater than .
```

```
0 is greater than .
```

この結果は、%SYSEVALF 関数によって浮動小数点値と欠損値が評価されたことを示しています。

論理式での文字オペランドの比較

マクロプロセッサが論理式をどのように評価するかを説明するために、マクロ COMPCHAR について考えます。呼び出されたマクロ COMPCHAR は、パラメータとして渡された値を比較し、その結果をログに出力します。

```
%macro compchar(first,second);
%if &first>&second %then %put &first comes after &second;
%else %put &first comes before &second;
%mend compchar;
```

次の値を使用して、マクロ COMPCHAR を呼び出します。

```
%compchar(a,b)
%compchar(.,1)
%compchar(Z,E)
```

次の結果がログに出力されます。

```
a comes before b
. comes before 1
Z comes after E
```

マクロプロセッサは、文字オペランドを含む式を評価するときに、ホストオペレーティングシステムの並べ替え順を比較で使用します。これらの例の比較は、EBCDIC と ASCII の両方の並べ替え順を使用して動作します。

文字オペランドの特殊ケースとして、数値に見えるがピリオドを含むオペランドがあります。ピリオドを含むオペランドを式で使用した場合、両方のオペランドは文字値として比較されます。これによって、予期しない結果が生じる場合があります。結果について理解し、より良い結果が期待できるようにするために、次の例を参照してください。

次の値を使用して、マクロ COMPNUM を呼び出します。

```
%compnum(10,2.0)
```

次の値がログに書き込まれます。

```
10 is less than 2.0
```

マクロ COMPNUM 内の%IF-THEN ステートメントは、整数の評価を使用しているため、小数点を含むオペランドを数値に変換しません。オペランドは、ホストの並べ替え順を使用して文字列として比較されます。つまり、最小値から最大値までの値を持つ文字の比較になります。たとえば、小文字は大文字よりも小さい値を持つ場合があり、大文字は数字よりも小さい値を持つ場合があります。

注意:

ホストの並べ替え順によって比較結果が決まります。 複数のオペレーティングシステム上で同じマクロ定義を使用した場合、ホストオペレーティングシステム間で並べ替え順が異なる可能性があるため、比較結果が変わる場合があります。ホストの並べ替え順の詳細については、“SORT プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

7 章

マクロクォーティング

マクロクォーティング

特殊文字とニーモニックのマスク

マクロ言語は、文字ベースの言語です。数値として表示される変数でも、通常は文字変数として扱われます(ただし、式の評価中を除きます)。したがって、マクロプロセッサを使用して、あらゆる特殊文字をテキストとして生成できます。ところが、マクロ言語には同じ特殊文字がいくつか含まれているため、頻繁にあいまいさが発生します。マクロプロセッサは、特定の特殊文字(たとえば、セミコロンや%記号)またはニーモニック(たとえば、GE や AND)をテキストとして解釈するのか、それともマクロ言語のシンボルとして解釈するのかを知る必要があります。マクロクォーティング関数は、特殊文字の意味をマスクすることによってこれらのあいまいさを解決し、マクロプロセッサがそれらを誤って解釈しないようにします。

次の特殊文字およびニーモニックは、文字列に現れたときにマスクする必要がある場合があります。

表 7.1 *特殊文字とニーモニック*

空白	)	=	LT
;	(	|	GE
¬	+	AND	GT
^	—	OR	IN
~	*	NOT	%
, (カンマ)	/	EQ	&
'	<	NE	#
“	>	LE	

マクロクォーティングの必要性について

*マクロクォーティング関数*は、特殊文字やニーモニックを、マクロ言語の一部としてではなくテキストとして解釈するようにマクロプロセッサに指示します。マクロクォーティング関数を使用して特殊文字をマスクしないと、マクロプロセッサや、SAS のその他の処理で、特殊文字に意図していない意味が与えられる場合があります。文字列に特殊文字やニーモニックが含まれる場合に発生する可能性のあるあいまいさの種類について、次にいくつか例を示します。

- `%sign` は、マクロ SIGN の呼び出しなのか、それとも"パーセント記号"という語句なのか
- OR は、ニーモニックのブール演算子なのか、それとも Oregon の略称なのか

- O'Malley に含まれる引用符は、一致しない一重引用符なのか、それとも単に名前の一部なのか
- Boys&Girls は、マクロ変数&GIRLS への参照なのか、それとも子供のグループなのか
- GE は、"以上"を意味するニーモニックなのか、それとも General Electric の略称なのか
- セミコロンが末尾を示すのは、どのステートメントか
- カンマは、パラメータを区切っているのか、それともいずれかのパラメータの値の一部なのか

マクロクォーティング関数を使用すると、特殊文字やニーモニックをどう解釈すべきかについて、マクロプロセッサに対して明確に指定できます。

ここでは、最も単純なマクロクォーティング関数、%STR の使用例を示します。PROC PRINT ステートメントと RUN ステートメントを、マクロ変数 PRINT に割り当てるとします。次のステートメントは、間違っています。

```
%let print=proc print; run;; /* undesirable results */
```

このコードはあいまいです。PRINT と RUN の後ろのセミコロンは、マクロ変数 PRINT の値に含まれているとも解釈できますし、どちらかが%LET ステートメントの末尾を示しているとも解釈できます。どう解釈するべきかをマクロプロセッサに指示しないと、PRINT の後ろのセミコロンが、%LET ステートメントの末尾として解釈されます。このため、マクロ変数 PRINT の値は次のようになります。

```
proc print
```

残りの文字(RUN;;)は、単にプログラムの次の部分になります。

あいまいさを避け、PRINT の値を正しく割り当てるには、マクロクォーティング関数%STR を次のように使用してセミコロンをマスクする必要があります。

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

マクロクォーティング関数の概要

最も一般的に使用されるマクロクォーティング関数は、次のとおりです。

- %STR および%NRSTR
- %BQUOTE および%NRBQUOTE
- %SUPERQ

対になったマクロクォーティング関数の場合、名前が NR で始まる関数は、NR の付かない関数がマスクする特殊文字のカテゴリに加えて、アンパサンドとパーセント記号に影響を与えます。つまり、NR 関数は、マクロおよびマクロ変数の置換を抑制します。どの関数がどれをマスクするかを覚えやすくするために、マクロクォーティング関数名に含まれる NR から、"置換されない(not resolved)"という言葉を連想するようにしてください。つまり、NR の付いた関数を使用すると、マクロおよびマクロ変数は置換されません。

名前に B を含むマクロクォーティング関数は、一致しない引用符およびかっこをマクロクォーティングするのに役立ちます。この B の意味を覚えやすくするために、B から"単独で(by itself)"を連想するようにしてください。

%SUPERQ マクロクォーティング関数は、仲間を持たず、異なる動作をするという点で、他のマクロクォーティング関数とは違っています。詳細については、“%SUPERQ 関数” (263 ページ)を参照してください。

また、マクロクォーティング関数は、それらが有効になるタイミングに基づいて、次の 2 種類に分類できます。

コンパイル関数
: オープンコードのマクロプログラムステートメントにおいて、またはマクロのコンパイル(作成)中に、特殊文字をテキストとしてマクロプロセッサに解釈させます。%STR 関数と%NRSTR 関数は、コンパイル関数です。詳細については、“%STR 関数と%NRSTR 関数” (258 ページ)を参照してください。

実行関数
: マクロ式を置換することによって得られた特殊文字(マクロ変数参照、マクロ呼び出し、%EVAL 関数の引数など)を、マクロプロセッサにテキストとして扱わせます。これらは、マクロの実行中、またはオープンコード内のマクロプログラムステートメントの実行中に置換が発生するため、実行関数と呼ばれます。マクロプロセッサは、可能な限り式を置換して、置換できないマクロ変数参照やマクロ呼び出しについては警告メッセージを発行し、その結果をクォーティングします。%BQUOTE 関数と%NRBQUOTE 関数は、実行関数です。詳細については、“%BQUOTE 関数と%NRBQUOTE 関数” (246 ページ)を参照してください。

%SUPERQ 関数は、引数としてマクロ変数名(またはマクロ変数名を生成するマクロ式)を受け取ります。この関数の引数に、マスク対象の値が格納されたマクロ変数への参照を渡さないでください。つまり、マクロ変数名の前に&を付けないでください。

注: この他に、%QUOTE および%NRQUOTET という 2 つの実行マクロクォーティング関数があります。これらは、マクロクォーティングに固有の必要性のため、および以前のマクロアプリケーションとの互換性を保つために役立ちます。詳細については、“%QUOTE 関数と%NRQUOTE 関数” (252 ページ)を参照してください。

特殊文字とニーモニックを含むパラメータを渡す

置換された値に特殊文字が含まれる可能性がある場合、それをマクロプロセッサに渡す最も簡単かつ最良の方法は、マクロ定義で実行マクロクォーティング関数を使用することです。ただし、実行マクロクォーティング関数を使用してマクロを定義していないときに、or のようなパラメータ値を渡す必要があることが判明した場合、マクロ呼び出しで値をマスクすることによって、それを渡すことができます。この処理のロジックは、次のとおりです。

1. マクロクォーティング関数を使用して特殊文字をマスクすると、その特殊文字は、マクロ機能の内部にある間はマスクされたままになります(ただし、“%UNQUOTE 関数” (279 ページ)を使用した場合を除きます)。
2. マクロプロセッサは、完全なマクロ呼び出しを構築してから、そのマクロの実行を開始します。
3. したがって、%STR 関数を使用して、呼び出しに含まれる値をマスクできます。マクロプロセッサが呼び出しを構築しているときにマスクは不要ですが、マクロの実行が開始された時点で、値はすでにマクロクォーティング関数によってマスクされています。そのため、マクロの実行中にその値によって問題が発生することはありません。

たとえば、ORDERX というマクロで%BQUOTE 関数を使用していないと仮定します。次の呼び出しによって、マクロ ORDERX に値 or を渡すことができます。

```
%orderx(%str(or))
```

ただし、マクロクォーティング関数をマクロ定義内に配置しておくと、マクロの呼び出しが非常に簡単になります。

いつ、どのマクロクォーティング関数を使用するのかについて

マクロ言語の一部として解釈できる特殊文字をマクロ変数に割り当てる場合、常にマクロクォーティング関数を使用します。次の表で、文字列の一部として使用するときにマスクする特殊文字と、各状況においてどのマクロクォーティング関数が役立つかについて説明します。

表 7.2 特殊文字とマクロクォーティングのガイドライン

特殊文字	マスクが必要な状況	すべてのマクロクォーティング関数によってクォーティングされるか	備考
+-*/<>=^|¬ ~ # LE LT EQ NE GE GT AND OR NOT IN	%EVAL 関数の引数で、演算子として扱われないようにするため。	はい	AND、OR、IN、および NOT は、%EVAL および%SYSEVALF によってニーモニック演算子として解釈されるため、マスクする必要があります。
空白	値の前後の空白または単独の空白が無視され、維持されるようにするため。	はい	
;	マクロプログラムステートメントの末尾を誤って示さないないようにするため。	はい	
, (カンマ)	関数の新しい引数、パラメータ、またはパラメータ値を示さないようにするため。	はい	

特殊文字	マスクが必要な状況	すべてのマクロクォーティング関数によってクォーティングされるか	備考
' " ( )	不一致である可能性がある場合。	いいえ	マクロ機能が一重引用符、二重引用符、およびかっこを、マクロ言語のシンボルまたは SAS 言語の一致しない引用符やかっことしてではなく、テキストとして解釈するために、引用符やかっこを含む可能性のある引数をマクロクォーティング関数を使用してマスクする必要があります。%STR、%NRSTR、%QUOTE、および%NRQUOTE では、%記号を使用して、一致しない引用符とかっこにマークを付ける必要があります。%BQUOTE、%NRBQUOTE、および%SUPERQ の引数に含まれる一致しないシンボルには、マークを付ける必要はありません。
%name &name	(式の置換結果によって変わります。)	いいえ	%NRSTR、%NRBQUOTE、および%NRQUOTE は、これらのパターンをマスクします。マクロ変数を渡して%SUPERQ を使用するには、*名前*からアンパサンドを省略します。

マクロ機能は、マクロの設計において、最大限の柔軟性を提供します。マクロプロセッサが特殊文字を、テキストとしてではなく、マクロ言語の一部として別の解釈をする場合にのみ、マクロクォーティング関数を使用して特殊文字をマスクする必要があります。たとえば次のステートメントでは、最初の 2 つのセミコロンをテキストの一部にするために、マクロクォーティング関数を使用してそれらをマスクする必要があります。

```
%let p=%str(proc print; run;);
```

しかし、次に示すマクロ PR では、PRINT と RUN の後ろのセミコロンを、マクロクォーティング関数を使用してマスクする必要がありません。

```
%macro pr(start);
%if &start=yes %then
%do;
```

```
%put proc print requested;
proc print;
run;
%end;
%mend pr;
```

マクロプロセッサは、%DO グループ内でセミコロンが使用されることを期待していません。そのため、PRINT と RUN の後ろのセミコロンはあいまいではなく、テキストとして解釈されます。

すべての状況に当てはまる一連のルールを提供することはできませんが、以降のセクションでは、各マクロクォーティング関数の使用方法について説明します。表 7.6 (96 ページ)では、マスクが必要な場合のあるさまざまな文字の概要について示されています。そこに含まれるマクロクォーティング関数は、各状況で役立ちます。

注: マクロクォーティング関数の逆を実行することもできます。つまり、マクロクォーティング関数が提供したトークン化を削除できます。%UNQUOTE 関数が役立つ場合の例については、“テキストのクォーティング解除 ” (97 ページ)を参照してください。

%STR 関数と%NRSTR 関数

%STR 関数と%NRSTR 関数の使用

特殊文字またはニーモニックが、マクロプロセッサによるマクロプログラムステートメントの構築方法に影響を与える場合、マクロクォーティング関数%STR または%NRSTR のいずれかを使用して、マクロのコンパイル中(または、オープンコード内のマクロプログラムステートメントのコンパイル中)にそれらの項目をマスクする必要があります。

これらのマクロクォーティング関数は、次の特殊文字とニーモニックをマスクします。

表 7.3 *%STR 関数と%NRSTR 関数によってマスクされる特殊文字*

空白	)	=	NE	
;	(	|	LE	
¬	+	#	LT	
^	—	AND	GE	
~	*	OR	GT	
, (カンマ)	/	NOT		
'	<	IN		
“	>	EQ		

これらの特殊文字とニーモニックに加えて、%NRSTR は&と%もマスクします。

注: 一致しない一重引用符、二重引用符、左かっこ、右かっこを%STR または%NRSTR で使用する場合、これらの文字の前にパーセント記号(%)を付ける必要があります。

%STR または%NRSTR を使用した場合、マクロプロセッサは、マクロの実行時にはこれらの関数とその引数を受け取りません。これらの関数はマクロのコンパイル時に動作するため、マクロプロセッサはこれらの関数の結果のみを受け取ります。マクロが実行される時点で、すでに文字列はマクロクォーティング関数によってマスクされています。したがって、%STR と%NRSTR は、SAS コードのセクションなどの変化しない文字列をマスクするにの役立ちます。特に、%記号や&記号を含む文字列をマスクする場合は、%NRSTR を使用することをお勧めします。ただし、これらの関数は、マクロ変数への参照を含む文字列のマスクにはあまり役立ちません。これは、マクロ変数が、%STR や%NRSTR ではクォーティングできない値に置換される可能性があるためです。たとえば、その文字列に、マークが付いていない不一致の左かっこが含まれている可能性があります。

一致しない引用符とかっこを%STR と%NRSTR と共に使用する

%STR または%NRSTR の引数に、一致しない一重引用符や二重引用符、または一致しない左かっこや右かっこが含まれる場合、それらの各文字の前に%記号を付ける必要があります。この方法について、いくつかの例を次の表に示します。

表 7.4 *%STR と%NRSTR に渡す一致しない引用符とかっこにマークを付ける例*

表記	説明	例	クォーティングされて格納される値
%'	一致しない一重引用符	`%let myvar= %str(a%');`	`a'`
%"	一致しない二重引用符	`%let myvar= %str(title %”first);`	`title “first`
%(	一致しない左かっこ	`%let myvar= %str (log %(12);`	`log(12`
%)	一致しない右かっこ	`%let myvar= %str (345%));`	`345)`

%記号を%STR と共に使用する

通常は、コンパイル時にマクロクォーティング関数を使用して%記号をマスクする場合、%NRSTR を使用します。%STR を使用して%記号をマスクできるケースが 1 つだけあります。それは、%記号の後ろに何もテキストが続かない場合です。この場合、マクロプロセッサによってマクロ名として解釈されます。%記号には、別の%記号によってマークを付ける必要があります。次に、例をいくつか示します。

表 7.5 *%STR に渡す%記号をマスクする例*

表記	説明	例	クォーティングされて格納される値
'%'	一致する一重引用符の前の%記号	`%let myvar= %str('%');`	'%'

表記	説明	例	クォーティングされて格納される値
%%%'	一致しない一重引用符の前の%記号	`%let myvar= %str(%%%');`	`%'`
""%%	一致する二重引用符の後の%記号	`%let myvar= %str(""%%);`	`""%`
%%%%	1 行に含まれる 2 つの%記号	`%let myvar= %str(%%%%);`	`%%`

%STR の使用例

次の%LET ステートメント内の%STR 関数は、PROC PRINT の後ろのセミコロンが、%LET ステートメントの末尾を示すセミコロンとして解釈されないようにしています。

```
%let printit=%str(proc print; run;);
```

さらに複雑な例として、次のマクロ KEEPIT1 は、マクロ定義内での%STR 関数の動作を示しています。

```
%macro keepit1(size);
%if &size=big %then %put %str(keep city _numeric_;);
%else %put %str(keep city;);
%mend keepit1;
```

このマクロを次のように呼び出します。

```
%keepit1(big)
```

このコードは、次のステートメントを生成します。

```
keep city _numeric_;
```

%IF-%THEN ステートメントで%STR 関数を使用すると、マクロプロセッサは、ワード%THEN の後の最初のセミコロンをテキストとして解釈します。2 番目のセミコロンは%THEN ステートメントの末尾を示し、その直後に%ELSE ステートメントが続きます。したがって、マクロプロセッサは、これらのステートメントを意図したとおりにコンパイルします。しかし%STR 関数を省略した場合、マクロプロセッサは、ワード%THEN の後の最初のセミコロンを%THEN 句の末尾として解釈します。その次のセミコロンは、定数テキストとして解釈されます。%ELSE 句の前に記述できるのは%THEN 句のみであるため、定数テキストのセミコロンがあることで、マクロプロセッサはエラーメッセージを発行し、このマクロをコンパイルしません。

%ELSE ステートメントでは、%STR 関数を使用することで、マクロプロセッサはステートメント内の最初のセミコロンをテキストとして扱い、2 番目のセミコロンを%ELSE 句の末尾として解釈できます。したがって、KEEP ステートメントの末尾を示すセミコロンは、条件付き実行の一部になります。%STR 関数を省略した場合、最初のセミコロンが%ELSE 句の末尾になり、2 番目のセミコロンは条件付き実行の外側に位置します。これは、マクロが実行されるたびにテキストとして生成されます。(この例では、セミコロンの存在は SAS コードに影響を与えません。)この場合も、%STR を使用することでマクロ KEEPIT1 を意図したとおりにコンパイルできます。

%STR を使用して、一致しない一重引用符が含まれる文字列をマスクする例を、次に示します。引用符の前で%記号を使用していることに注意してください。

```
%let innocent=%str(I didn%'t do it!);
```

%NRSTR の使用例

マクロ変数の(値ではなく)名前を%PUT ステートメントによって出力する場合を考えます。これを行うには、次の例のように、%NRSTR 関数を使用して&をマスクし、マクロ変数が置換されないようにする必要があります。

```
%macro example;
%local myvar;
%let myvar=abc;
%put %nrstr(The string &myvar appears in log output,);
%put instead of the variable value.;
%mend example;

%example
```

このコードによって、次のテキストが SAS ログに書き込まれます。

```
The string &myvar appears in log output,
instead of the variable value.
```

%NRSTR 関数を使用しなかった場合、または%STR 関数を使用した場合、次のような望ましくない出力が SAS ログに表示されます。

```
The string abc appears in log output,
instead of the variable value.
```

%NRSTR 関数は、&がマクロ変数置換のトリガにならないようにします。

マクロ定義に、通常はマクロプロセッサによってマクロ変数参照として認識されるパターンが含まれる場合にも、%NRSTR 関数は役立ちます。次にプログラム例を示します。

```
%macro credits(d=%nrstr(Mary&Stacy&Joan Ltd.));
footnote "Designed by &d";
%mend credits;
```

%NRSTR を使用することで、マクロプロセッサは、&STACY と&JOAN を単に D の値のテキストの一部として扱うことができます。そのため、マクロプロセッサは、マクロ変数参照を置換できないことを示す警告メッセージを発行しません。D のデフォルト値を使用して、次のようにマクロ CREDITS を呼び出したとします。

```
%credits()
```

このプログラムをサブミットすると、次の FOOTNOTE ステートメントが生成されます。

```
footnote "Designed by Mary&Stacy&Joan Ltd.";
```

%NRSTR 関数を省略した場合、マクロプロセッサは、&STACY 参照と&JOAN 参照を、FOOTNOTE ステートメント内の&D の置換の一部として置換しようとします。そのようなマクロ変数は存在しないため、マクロプロセッサは次の警告メッセージを発行します(ただし、“マクロのシステムオプション” (335 ページ)で説明されている SERROR システムオプションが有効であることが前提です)。

```
WARNING: Apparent symbolic reference STACY not resolved.
WARNING: Apparent symbolic reference JOAN not resolved.
```

%NRSTR を使用する最後の例を示します。マクロ関数名を含む、**`This is the result of %NRSTR`** という文字列を作成する場合を考えます。次に、このプログラムを示します。

```
%put This is the result of %nrstr(%nrstr);
```

コンパイル時に、%NRSTR を使用して%記号をマスクする必要があります。そうすることで、マクロプロセッサは 2 回目の%NRSTR を呼び出しません。%NRSTR を使用して文字列`%nrstr` をマスクしないと、マクロプロセッサは、関数の左かっこが欠損していることを示すエラーメッセージを発行します。

%BQUOTE 関数と%NRBQUOTE 関数

%BQUOTE 関数と%NRBQUOTE 関数の使用

%BQUOTE と%NRBQUOTE は、マクロまたはオープンコード内のマクロ言語ステートメントの実行中に、値をマスクします。これらの関数は、可能な限りマクロ式を置換してから結果をマスクするようにマクロプロセッサに指示し、置換できないマクロ変数参照またはマクロ呼び出しに対して警告メッセージを発行します。これらの関数は、%STR および%NRSTR がマスクするすべての文字に加えて、マークが付いていないパーセント記号、マークが付いていない不一致の一重引用符と二重引用符、およびマークが付いていない不一致の左かっこと右かっこをマスクします。つまり、%STR や%NRSTR を使用する場合のように、一致しない引用符の前に%記号を付ける必要はありません。

%BQUOTE 関数は、マクロ変数参照またはマクロ呼び出しを置換することによって生成されたすべてのかっこと引用符を、実行時にマスクするべき特殊文字として扱います。(この関数は、コンパイル時には、引数に含まれるかっこや引用符をマスクしません。)したがって、置換された値に含まれる引用符やかっこが一致しているかどうかは問題になりません。それぞれは、個別にマスクされます。

%NRBQUOTE 関数は、可能であれば最初に値を検出した時点で置換したいが、その結果に含まれるどのアンパサンド、パーセント記号も%EVAL 関数によって演算子として解釈されたくない場合に役立ちます。

置換できないマクロ変数参照またはマクロ呼び出しが%NRBQUOTE 関数の引数に含まれている場合、マクロプロセッサは、警告メッセージを発行してからアンパサンドまたはパーセント記号をマスクします(ただし、“マクロのシステムオプション” (335 ページ)で説明されている SERROR システムオプションまたは MERROR システムオプションが有効であることが前提です)。置換できないマクロ変数に対するメッセージを抑制するには、代わりに、このセクションで後述する%SUPERQ 関数を使用します。

%BQUOTE 関数と%NRBQUOTE 関数は、実行時に動作するため、%STR や%NRSTR よりも柔軟です。そのため、マクロ変数参照を含む文字列をマスクする場合、これらの関数を使用することをお勧めします。

%BQUOTE の使用例

次の%IF-%THEN ステートメントでは、%BQUOTE を使用して、マクロ変数 STATE が OR(Oregon を表す)に置換されることによるエラーを防いでいます。マクロプロセッサは、この OR を、誤って論理演算子の OR と解釈します。

```
%if %bquote(&state)=%str(OR) %then %put Oregon Dept. of
Revenue;
```

注: %STR を使用してもこの例は動作しますが、そのようなプログラムは堅牢でなく、良い実装でもありません。&STATE が何に置換されるのか保証できないため、コンパイル時に変数自体の名前をマスクするのではなく、%BQUOTE を使用して実行時にマクロ変数の置換結果をマスクする必要があります。

次の例の DATA ステップでは、一重引用符を含む文字値を作成し、その値をマクロ変数に割り当てています。その後のマクロ READIT では、%BQUOTE 関数を使用して、一致しない一重引用符を%IF 条件として受け取ることを可能にしています。

```
data test;
store="Susan's Office Supplies";
call symput('s',store);
run;

%macro readit;
%if %bquote(&s) ne %then %put *** valid ***;
%else %put *** null value ***;
%mend readit;

%readit
```

DATA ステップで **Susan's Office Supplies** という値を STORE に割り当てるときに、この文字列を二重引用符で囲むことで、一致しない一重引用符を文字列内で使用できるようにしています。SAS は、STORE に次の値を格納します。

```
Susan's Office Supplies
```

CALL SYMPUT ルーチンは、一致しない一重引用符を含むこの値を、マクロ変数 S に割り当てます。マクロ READIT 内で S を参照するときに%BQUOTE 関数を使用しない場合、マクロプロセッサは、%IF 条件のオペランドが無効であることを示すエラーメッセージを発行します。

このコードをサブミットすると、次のメッセージが SAS ログに書き込まれます。

```
*** valid ***
```

クォーティング済み変数の参照

マクロクォーティング関数によってマスクされた項目(次のプログラムの WHOSE の値など)は、それがマクロプロセッサによって使用されている限り、マスクされたままです。後でマクロプログラムステートメントで WHOSE の値を使用するときには、その参照を再びマスクする必要はありません。

```
/* Use %STR to mask the constant, and use a % sign to mark */
/* the unmatched single quotation mark. */
%let whose=%str(John%'s);

/* You don't need to mask the macro reference, because it was */
/* masked in the %LET statement, and remains masked. */
%put *** This coat is &whose ***;
```

この%PUT ステートメントによって、次の出力が SAS ログに書き込まれます。

```
*** This coat is John's ***
```

マクロクォーティング関数でマスクするテキスト量を決める

次の各ステートメントにおいて、マクロプロセッサはマスクされたセミコロンをテキストとして扱います。

```
%let p=%str(proc print; run;);
%let p=proc %str(print;) %str(run;);
%let p=proc print%str(;) run%str(;);
```

各ケースで、P の値は、次に示す値と同じになります。

```
proc print; run;
```

これら 3 つの%LET ステートメントの結果が同じになるのは、マクロクォーティング関数を使用してテキストをマスクすると、この関数が認識する項目のみがマクロプロセッサによってクォーティングされるためです。この関数で囲まれた他のテキストは、変更されません。したがって、3 番目の%LET ステートメントが、マクロクォーティングするための最小限の方法です。ただし、マクロクォーティング関数を使用して大きなテキストブロックをマスクしても問題はなく、1 番目の%LET ステートメントなどのように、実際にはコードが非常に読みやすくなります。

%SUPERQ 関数

%SUPERQ の使用

%SUPERQ 関数は、引数で指定されたマクロ変数を検索し、いかなる置換の実行も許可せず、そのマクロ変数の値をクォーティングします。この関数は、マクロの実行時に、マクロクォーティングを必要とする可能性のあるすべての項目をマスクします。%SUPERQ は、引数に対していかなる置換も試みません。そのためマクロプロセッサは、マクロ変数参照またはマクロ呼び出しが置換されなかったことを示す警告メッセージを発行しません。したがって、プログラムが%NRBQUOTE 関数を使用して正しく動作できる場合でも、代わりに%SUPERQ 関数を使用すれば、不必要な警告メッセージを SAS ログから除去できます。%SUPERQ は、アンパサンドを付けないマクロ変数名、またはマクロ変数名を生成するテキスト式のいずれかを、引数で受け取ります。

%SUPERQ は、マクロシンボルテーブルからマクロ変数の値を取り出して、即座にその値をクォーティングし、置換で生じたすべての値の置換をマクロプロセッサに実行させないようにします。たとえば、マクロ変数 CORPNAME が **Smith&Jones** に置換される場合、%SUPERQ を使用することで、マクロプロセッサがさらに**&Jones** の置換を試みないようにします。次の%LET ステートメントによって、値 **Smith&Jones** が正常に TESTVAR に割り当てられます。

```
%let testvar=%superq(corpname);
```

%SUPERQ の使用例

次の例は、%SUPERQ 関数が 2 つのマクロ呼び出し(1 つは定義済みマクロ、もう 1 つは未定義のマクロ)にどのように影響を与えるかを示しています。

```
%window ask
#5 @5 'Enter two values:'
```

```
#5 @24 val 15 attr=underline;

%macro a;
%put *** This is a. ***;
%mend a;

%macro test;
%display ask;
%put *** %superq(val) ***; /* Note absence of ampersand */
%mend test;
```

マクロ TEST を呼び出し、表示されたプロンプトに次のように入力したとします。

```
%test
Enter the following: %a %x
```

この%PUT ステートメントは、単に次の行を書き込みます。

```
*** %a %x ***
```

マクロ A は呼び出されず、%X が置換されなかったことを示す警告メッセージも発行されません。以降の 2 つの例で、%SUPERQ 関数と他のマクロクォーティングを関数を比較します。

%SUPERQ 関数を使用した警告メッセージの回避

%NRBQUOTE 関数に関するセクションには、この関数を使用すると、マクロプロセッサが、マクロの実行中に*&name* や*%name* のパターンを最初に検出したときに、これらのパターンの置換を試みることが示されています。マクロプロセッサは、これらを置換できなかった場合、その値を後で使用したときにアンパサンドまたはパーセント記号を認識しないようにするために、アンパサンドまたはパーセント記号をクォーティングします。ただし、MERROR オプションまたは SERROR オプションが有効な場合、マクロプロセッサは、参照または呼び出しが置換されなかったことを示す警告メッセージを発行します。

次に示すマクロ FIRMS3 は、%SUPERQ 関数によって不必要な警告メッセージを回避する方法を示しています。

```
%window ask
#5 @5 'Enter the name of the company:'
#5 @37 val 50 attr=underline;

%macro firms3;
%global code;
%display ask;
%let name=%superq(val);
%if &name ne %then %let code=valid;
%else %let code=invalid;
%put *** &name is &code ***;
%mend firms3;

%firms3
```

マクロ FIRMS3 を 2 回呼び出して、次の会社名を入力したとします。

```
A&A Autos
Santos&D'Amato
```

マクロの実行後に、次のメッセージがログに書き込まれます。

```
*** A&A Autos is valid ***
```

```
*** Santos&D'Amato is valid ***
```

%SUPERQ 関数を使用したマクロキーワードの入力

例として、ユーザーが問題と質問を入力し、それらを別のマクロによって後で印刷できるオンライントレーニングシステムを作成することを考えます。%WINDOW ステートメントへのユーザーの入力は、ローカルマクロ変数に割り当てられてからグローバルマクロ変数に割り当てられます。ユーザーは、マクロに関して質問するため、問題の例に従うすべての種類のマクロ変数参照およびマクロ呼び出しだけでなく、マークが付いていない不一致の引用符およびかっこを入力する可能性があります。%BQUOTE を使用して入力をマスクした場合は、いくつかの%PUT ステートメントを使用して、問題を引き起こす入力についてユーザーに警告しています。%SUPERQ 関数を使用すれば、より少ない指示ですみます。マクロ ASK1 と ASk2 は、マクロクォーティング関数を変更すると、マクロコードがどのように単純化されるかを示しています。

次のマクロ ASK1 に、%BQUOTE 関数を使用した場合のマクロコードを示します。

```
%window ask
#5 @5 'Describe the problem.'
#6 @5 'Do not use macro language keywords, macro calls,'
#7 @5 'or macro variable references.'
#9 @5 'Enter /// when you are finished.'
#11 @5 val 100 attr=underline;

%macro ask1;
%global myprob;
%local temp;

%do %until(%bquote(&val) eq %str(///));
%display ask;
%let temp=&temp %bquote(&val);
%end;
%let myprob=&temp
%mend ask1;
```

マクロ ASK1 には、一致しない引用符およびかっこに関する警告が含まれていません。次のようにマクロ ASK1 を呼び出し、問題を入力できます。

```
%ask1

Try entering:
Why did my macro not run when I called it? (It had three
parameters, but I wasn't using any of them.)
It ran after I submitted the next statement.
///
```

入力の 1 行目と 2 行目の両方に、マークが付いていない不一致の引用符とかっこが含まれていることに注目してください。%BQUOTE は、実行中にこれらの文字を処理できます。

次に示すマクロ ASK2 は、%SUPERQ 関数を使用して ASK1 に変更を加えたものです。ここでは、%WINDOW ステートメントはマクロ言語キーワードを受入れ、マクロ呼び出しおよびマクロ変数参照の置換を試みません。

```
%window ask
#5 @5 'Describe the problem.'
#7 @5 'Enter /// when you are finished.'
#9 @5 val 100 attr=underline;
```

```
%macro ask2;
%global myprob;
%local temp;

%do %until(%superq(val) eq %str(///)); /* No ampersand */

%display ask;
%let temp=&temp %superq(val); /* No ampersand */
%end;
%let myprob=&temp
%mend ask2;
```

次のようにマクロ ASK2 を呼び出して、入力できます。

```
%ask2

Try entering:
My macro ADDRESS starts with %MACRO ADDRESS(COMPANY,
CITY);. I called it with %ADDRESS(SMITH-JONES, INC., BOSTON),
but it said I had too many parameters. What happened?
///
```

この入力には、マクロ言語キーワード、マクロ呼び出し、および一致しないかっこが含まれています。

マクロクォーティング関数およびマスクされる文字の概要

マクロ機能が特殊文字やニーモニックをマクロ言語のシンボルとしてではなくテキストとして解釈できるようにするために、さまざまなマクロクォーティング関数によって異なった特殊文字とニーモニックがマスクされます。

次の表は、各シンボルをカテゴリごとに分類し、どのマクロクォーティング関数がどのシンボルをマスクするかを示しています。

表 7.6 *特殊文字と項目別のマクロクォーティング関数の概要*

グループ	項目	マクロクォーティング関数
A	+ — */<>=¬^|~;, # blank AND OR NOT EQ NE LE LT GE GT IN	すべて
B	&%	%NRSTR、%NRBQUOTE、%SUPERQ、%NRQUOTE
C	一致しない' “()	%BQUOTE、%NRBQUOTE、%SUPERQ、%STR*、%NRSTR*、%QUOTE*、%NRQUOTE*

表 7.7 関数による影響

関数	影響を受けるグループ	動作するタイミング
%STR	A、C*	マクロのコンパイル時
%NRSTR	A、B、C*	マクロのコンパイル時
%BQUOTE	A、C	マクロの実行時
%NRBQUOTE	A、B、C	マクロの実行時
%SUPERQ	A、B、C	マクロの実行時(置換は実行されない)
%QUOTE	A、C*	マクロの実行時。一致しない引用符とかっこには、パーセント記号(%)でマークを付ける必要がある。
%NRQUOTE	A、B、C*	マクロの実行時。一致しない引用符とかっこには、パーセント記号(%)でマークを付ける必要がある。

*一致しない引用符およびかっこを%STR、%NRSTR、%QUOTE、および%NRQUOTE で使用する場合、パーセント記号(%)を使用してそれらにマークを付ける必要があります。

テキストのクォーティング解除

シンボルの意味の復元

値を*クォーティング解除する*とは、それまでマクロクォーティング関数によってマスクされていた項目のシンボルの意味を復元するという意味です。

通常、ある項目がマクロクォーティング関数によってマスクされると、次のいずれかが発生するまで、その項目は特殊な状態に置かれます。

- その項目を%UNQUOTE 関数で囲む。((279 ページ)を参照してください。)
- その項目がワードスキャナから削除され、DATA ステップコンパイラ、SAS プロシジャ、SAS マクロ機能、または SAS システムの他の部分に渡される。
- その項目が、%SCAN 関数、%SUBSTR 関数、または%UPCASE 関数によって、クォーティング解除された結果として返される。(これらのいずれかの操作中に値のマスク状態を維持するには、%QSCAN 関数、%QSUBSTR 関数、または%QUPCASE 関数を使用します。詳細については、“マクロクォーティングを実行するその他の関数” (100 ページ)を参照してください。)

項目は、ワードスキャナから SAS の他の処理に渡されるときに、自動的にクォーティング解除されます。そのため、原則として項目をクォーティング解除する必要はありません。ただし、次の 2 つの状況においては、マスクされた項目に%UNQUOTE 関数を使用して、元の意味を復元する必要がある場合があります。

- それまでマクロクォーティング関数によって値をマスクしていたが、同じマクロ内で、後からその値の意味を復元して使用する場合。
- まれなケースとして、マクロクォーティング関数を使用してテキストをマスクすることで、ワードスキャナによるそのテキストのトークン化方法を変更し、一見正しく見えるが SAS コンパイラによって認識されない SAS ステートメントを生成する場合。

クォーティング解除の例

1 回はマクロクォーティングされた形式、もう 1 回はクォーティング解除された形式で、値を 2 回使用する例を次に示します。マクロ ANALYZE が 2 つの統計モデルの出力を対話的に比較できるようにするシステムの一部であると仮定します。まず、演算子を入力して、テストする関係(一方の結果が他方よりも大きい、他方と等しいなど)を指定します。マクロ ANALYZE は、マクロクォーティングされた演算子の値をテストして正しく入力されたことを確認し、クォーティング解除された値を使用して指定された値を比較し、メッセージを書き込みます。コメント内の番号と、次のパラグラフを対応付けてください。

```
%macro analyze(stat);
data _null_;
set out1;
call symput('v1',&stat);
run;

data _null_;
set out2;
call symput('v2',&stat);
run;

%put Preliminary test. Enter the operator.;
%input;
%let op=%bquote(&sysbuffr);
%if &op=%str(=<) %then %let op=%str(<=);
%else %if &op=%str(=>) %then %let op=%str(>=);
%if &v1 %unquote(&op) &v2 %then
%put You might proceed with the analysis.;
%else
%do;
%put &stat from out1 is not &op &stat from out2.;
%put Please check your previous models.;
%end;
%mend analyze;
```

SYSBUFFR の値を、%BQUOTE 関数を使用してマスクします。この関数は、マークが付いていない不一致の引用符やかっこを含む(ただし、アンパサンドとパーセント記号を除く)置換済み項目をマスクします。

%IF 条件は、マクロ変数 OP の値を文字列と比較して、OP の値に正しい演算子のシンボルが含まれているかどうかを調べます。間違った順序のシンボルが値に含まれている場合、%THEN ステートメントによって正しく修正されます。マクロクォーティング関数によってマスクされた値はマスクされたままになるため、%IF 条件の左側の&OP 参照をマスクする必要はありません。

マクロを定義するときに、%IF 条件の右側の文字と%LET ステートメントの文字の値は見ることができます。そのため、%STR 関数を使用してこれらの文字をマスクできます。これらをコンパイル時にマスクしておくと、ANALYZE を実行するたびにマスクするよりも効率的です。

マクロ変数 OP の値を%IF 条件の演算子として使用するには、%UNQUOTE 関数を使用して演算子の意味を復元する必要があります。

自動的にクォーティング解除されない場合の対処方法

マクロクォーティング関数によってマスクされた項目からマクロプロセッサがテキストを生成する場合、通常は、マクロクォーティングされた項目の自動的なクォーティング解除を SAS に行わせることができます。たとえば、マクロ変数 PRINTIT を次のように定義するとします。

```
%let printit=%str(proc print; run;);
```

次に、このマクロ変数をプログラム内で次のように使用します。

```
%put *** This code prints the data set: &printit ***;
```

マクロプロセッサがマクロ変数からテキストを生成するときに、マクロクォーティング関数によってマスクされた項目は自動的にクォーティング解除されます。それらの項目が SAS の他の処理に渡されると、それまでマスクされていたセミコロンが正常に機能します。

まれに、マクロクォーティング関数を使用してテキストをマスクし、ワードスキャナによるテキストのトークン化方法を変更することがあります。(ワードスキャナとトークン化については、“SAS プログラムとマクロ処理” (13 ページ)および“マクロ処理” (35 ページ)で説明されています。) たとえば、%BQUOTE 関数で置換されて生成された一重引用符または二重引用符は、別々のトークンになります。ワードスキャナは、入力スタック内のこのトークンを、リテラルトークンの境界として使用しません。%BQUOTE 関数によって一度マスクされて生成されたテキストが、一見正しく見えるのに SAS に受入れられない場合、%UNQUOTE 関数を使用して正常なトークン化を復元する必要がある場合があります。

マクロクォーティングの機能

マクロプロセッサは、文字列をマスクするときに、コーディングスキーマに含まれる特殊文字およびニーモニックをマスクし、その文字列に*デルタ文字*と呼ばれる 16 進文字の接頭語および接尾語を付加します。接頭語は、文字列の先頭にマークを付け、文字列に適用されるマクロクォーティングのタイプも示します。接尾語は、文字列の末尾にマークを付けます。接頭語および接尾語が付加されても、文字列に含まれる先頭および末尾の空白は失われません。特殊文字とニーモニックのマスクに使用される 16 進文字、および接頭語と接尾語に使用される 16 進文字は、変わる場合があるため移植不可能です。

各バイトには、キーボード上のシンボルを表すのに必要な数よりも多い、使用可能な 16 進数の組み合わせがあります。したがって、マクロクォーティング関数がマスク対象の項目を認識すると、マクロプロセッサは、まだ使用されていない 16 進数の組み合わせを接頭語と接尾語に使用します。

%EVAL や%SUBSTR などのマクロ関数は、これらの接頭語と接尾語を無視します。したがって、これらの接頭語と接尾語が比較に影響を与えることはありません。

マクロプロセッサは、マクロクォーティングされた文字列の処理を終えると、マクロクォーティングコード付き置換文字を削除し、それらを元の文字に置き換えます。システムの他の処理には、マスク解除された文字が渡されます。場合によっては、マスク解除に関するメッセージが表示されることがあります。次に、その例を示します。

```
/* Turn on SYMBOLGEN so you can see the messages about unquoting. */
options symbolgen;
```

```
/* Assign a value to EXAMPLE that contains several special */
/* characters and a mnemonic. */
%let example = %nrbquote( 1 + 1 = 3 Today's Test and More );

%put *&example*;
```

このプログラムをサブミットすると、次のメッセージが SAS ログに表示されます。

```
SYMBOLGEN: Macro variable EXAMPLE resolves to 1 + 1 = 3 Today's
Test and More
SYMBOLGEN: Some characters in the above value which were subject
to macro quoting have been unquoted for printing.
* 1 + 1 = 3 Today's Test and More *
```

このログから分かるように、変数の値の先頭と末尾の空白および特殊文字は、維持されています。マクロプロセッサが文字列を処理している間、実際の文字列には、本来の文字を置換したコード付き文字が含まれています。置換文字には、文字列の先頭と末尾を表すコード付き文字が含まれています。先頭と末尾の空白は維持されています。また、各文字は、特殊文字+、=、'、およびニーモニック **AND** を置換しています。マクロの処理が終了し、各文字が SAS の他の処理に渡されるときに、コードが削除されて本来の文字に置き換えられます。

マスクされた文字列をクォーティング解除するときに何が起きるかについては、“テキストのクォーティング解除 ” (97 ページ)で詳しく説明されています。詳細については、“SYMBOLGEN システムオプション” (364 ページ)を参照してください。

マクロクォーティングを実行するその他の関数

文字 Q で始まる関数

次に示す一部のマクロ関数は、一対で使用できます。これらのうち、一方の関数名は、文字 Q で始まります。

- %SCAN と%QSCAN
- %SUBSTR と%QSUBSTR
- %UPCASE と%QUPCASE
- %SYSFUNC と%QSYSFUNC

マクロ関数は、引数がマクロクォーティング関数によってマスクされた場合でも、デフォルトでクォーティング解除された結果を返します。そのため、Q*xxx* 関数が必要になります。%QSCAN 関数、%QSUBSTR 関数、%QUPCASE 関数、%QSYSFUNC 関数は、実行時に返される値をマスクします。マスクされた項目は、%NRBQUOTE 関数によってマスクされた項目と同じになります。

%QSCAN 関数の使用例

次のマクロでは、%QSCAN 関数を使用して、SYSBUFFR(“自動マクロ変数” (196 ページ)で説明されています)の値に含まれる項目を、別のマクロ変数の値として割り当てています。コメント内の番号は、マクロコードの後にある各説明に対応しています。

```
%macro splitit;
%put What character separates the values?; 1
%input;
```

```
%let s=%bquote(&sysbuffr); 2
%put Enter three values.;
%input;
%local i;
%do i=1 %to 3; 3
%global x&i;
%let x&i=%qscan(%superq(sysbuffr),&i,&s);
4
%end;
%mend splitit;

%splitit
What character separates the values?
#
Enter three values.
Fischer Books#Smith&Sons#Sarah's Sweet Shoppe
5
```

1. この質問は、入力する値に現れない、%QSCAN 関数が使用する区切り文字の入力を要求しています。
2. %BQUOTE 関数を使用して SYSBUFFR の値をマスクすることで、必要な場合、引用符またはかっこを区切り文字として選択できるようにしています。
3. 反復する%DO ループによって SYSBUFFR のセグメントごとにグローバルマクロ変数を作成し、その変数にセグメントの値を割り当てます。
4. %QSCAN 関数の第 1 引数に渡される SYSBUFFR の値を、%SUPERQ 関数によってマスクしています。これによって、SYSBUFFR の値に対するあらゆる置換が抑制されます。
5. %QSCAN 関数によって、マクロクォーティングされた SYSBUFFR の値のセグメントが返されます。したがって、**`Sarah's Sweet Shoppe`** に含まれる一致しない引用符、と **`Smith&Sons`** に含まれる&*name* のパターンは、問題を引き起こしません。

8 章

マクロ機能とのインターフェイス

マクロ機能とのインターフェイス

マクロ機能との *インターフェイス*は、マクロプロセッサではなく、SAS ソフトウェア機能に含まれています。このインターフェイスによって、SAS 言語の他の部分は、実行中にマクロ機能と相互作用することができます。たとえば、DATA ステップインターフェイスを使用して、DATA ステップからマクロ変数にアクセスできます。一般にマクロは、DATA ステップ、SQL、SCL、または SAS/CONNECT の実行前に処理されるため、通常、マクロ機能と SAS の他の部分との間の接続は動的ではありません。このため、マクロ機能インターフェイスが役立ちます。マクロ機能とのインターフェイスを使用することで、マクロ機能を SAS の他の部分に動的に接続できます。

注: %SYSFUNC マクロ関数と%QSYSFUNC マクロ関数によって、SAS 言語の関数をマクロプロセッサで使用することが可能になります。%SYSCALL マクロステートメントによって、SAS 言語の CALL ルーチンをマクロプロセッサで使用することが

可能になります。マクロ言語のこれらの要素は、true のマクロ機能インターフェイスとは見なされていませんが、これについては後述します。これらのマクロ言語要素の詳細については、“マクロ言語要素” (157 ページ) を参照してください。

DATA ステップインターフェイス

DATA ステップ実行時にマクロ機能と相互作用する

*DATA ステップインターフェイス*は、DATA ステップの実行中にプログラムとマクロ機能との相互作用を可能にする、8 つのツールから成ります。DATA ステップの実行が開始される前にマクロ機能が処理されるため、DATA ステップの実行時には、マクロステートメントによって提供される情報の処理がすでに完了しています。DATA ステップインターフェイスのいずれかを使用すると、DATA ステップの実行中にマクロ機能を操作できます。DATA ステップインターフェイスを使用して、次のことを実行できます。

- DATA ステップから SAS プログラムのその後のステップに、情報を渡せます。
- DATA ステップの実行時にのみ利用可能な情報に基づいてマクロを呼び出せます。
- DATA ステップの実行中にマクロ変数を置換できます。
- マクロ変数を削除できます。
- マクロ機能から DATA ステップに、マクロ変数に関する情報を渡せます。

次の表に、カテゴリ別の DATA ステップインターフェイスと、それらの用途を示します。

表 8.1 *マクロ機能との DATA ステップインターフェイス*

カテゴリ	ツール	説明
実行	CALL EXECUTE ルーチン	渡された引数を置換し、置換した値を次のステップ境界で実行するか(値が SAS ステートメントの場合)、即座に実行します(値がマクロ言語要素の場合)。
置換	RESOLVE 関数	DATA ステップの実行中に、テキスト式の値を置換します。
削除	CALL SYMDEL ルーチン	引数で指定されたマクロ変数を削除します。
情報	SYMEXIST 関数	マクロ変数が存在するかどうかを示す値を返します。
読み込みまたは書き込み	SYMGET 関数	DATA ステップの実行中に、マクロ変数の値を返します。
情報	SYMGLOBL 関数	マクロ変数のスコープがグローバルかどうかを示す値を返します。

カテゴリ	ツール	説明
情報	SYMLOCAL 関数	マクロ変数のスコープがローカルかどうかを示す値を返します。
読み込みまたは書き込み	CALL SYMPUT ルーチン	DATA ステップで生成された値をマクロ変数に割り当てます。

CALL EXECUTE ルーチンのタイミングの詳細

CALL EXECUTE は、マクロを条件付きで実行する場合に役立ちます。ただし、CALL EXECUTE によってマクロ言語要素が生成された場合、それらの要素が即座に実行されるということを覚えておく必要があります。CALL EXECUTE によって SAS 言語ステートメントが生成された場合、またはマクロ言語要素によって SAS 言語ステートメントが生成された場合、それらのステートメントは、DATA ステップの実行終了後に実行されます。

注: マクロ参照は即座に実行されますが、SAS ステートメントはステップ境界に達するまで実行されません。そのため、あるマクロにマクロ変数の参照が含まれており、そのマクロ変数が同じマクロ内で CALL SYMPUT によって作成されたものである場合、CALL EXECUTE を使用してそのマクロを呼び出すことはできません。

CALL EXECUTE の誤った使用例

次の例では、CALL EXECUTE ルーチンの使用方法が間違っています。

```
data prices; /* ID for price category and actual price */
input code amount;
datalines;
56 300
99 10000
24 225
;

%macro items;
%global special;
%let special=football;
%mend items;

data sales; /* incorrect usage */
set prices;
length saleitem $ 20;
call execute('%items');
saleitem="&special";
run;
```

DATA SALES ステップ内の、SALEITEM に値を割り当てるステートメントでは、DATA ステップのコンパイル時のマクロ変数 SPECIAL の値が必要です。CALL EXECUTE は、DATA ステップが実行されるまで、この値を生成しません。そのため、マクロ変数が置換されなかったことを示すメッセージが表示され、SALEITEM には **&special** という値が割り当てられます。

この例の場合、マクロ定義を除去するか(%LET マクロステートメントは、オープンコードでも有効です)、DATA SALES ステップをマクロ ITEM 内に移動することが望まれま

す。いずれの場合も、CALL EXECUTE は必要なく、有効でもありません。次に、このプログラムが動作する例を示します。

```
data prices; /* ID for price category and actual price */
input code amount;
datalines;
56 300
99 10000
24 225
;

%let special=football; /* correct usage */

data sales;
set prices;
length saleitem $ 20;
saleitem="&special";
run;
```

このバージョンでは、%GLOBAL ステートメントは不要です。%LET ステートメントがオープンコード内で実行されるため、グローバルマクロ変数が自動的に作成されます。(マクロ変数のスコープの詳細については、“マクロ変数のスコープ” (45 ページ)を参照してください。)

CALL EXECUTE によくある問題の例

次の例は、エラーを引き起こす、よくあるパターンを示しています。

```
/* This version of the example shows the problem. */

data prices; /* ID for price category and actual price */
input code amount;
cards;
56 300
99 10000
24 225
;
data names; /* name of sales department and item sold */
input dept $ item $;
datalines;
BB Boat
SK Skates
;

%macro items(codevar=); /* create macro variable if needed */
%global special;
data _null_;
set names;
if &codevar=99 and dept='BB' then call symput('special', item);
run;
%mend items;

data sales; /* attempt to reference macro variable fails */
set prices;
length saleitem $ 20;
if amount > 500 then
call execute('%items(codevar=' || code || ')' );
```

```
saleitem="&special";
run;
```

この例でも、DATA SALES ステップは、コンパイル時の SPECIAL の値を必要としています。この例の場合、条件付き IF ステートメントがあるため、CALL EXECUTE ルーチンは有効です。しかし、最初の例と同じく、CALL EXECUTE は、コンパイル時ではなく DATA ステップの実行中にマクロ ITEMS を呼び出しています。マクロ ITEMS は、DATA SALES ステップの実行終了後に実行される DATA _NULL_ステップを生成します。DATA _NULL_ステップは SPECIAL を作成しますが、SPECIAL の値は、_NULL_ステップの実行終了後、つまり、この値が必要とされるときよりもずっと後に使用可能になります。

次に示す例では、この問題を修正しています。

```
/* This version solves the problem. */

data prices; /* ID for price category and actual price */
input code amount;
datalines;
56 300
99 10000
24 225
;

data names; /* name of sales department and item sold */
input dept $ item $;
cards;
BB Boat
SK Ski
;
%macro items(codevar=); /* create macro variable if needed */
%global special;
data _null_;
set names;
if &codevar=99 and dept='BB' then
call symput('special', item);
run;
%mend items;

data _null_; /* call the macro in this step */
set prices;
if amount > 500 then
call execute('%items(codevar=' || code || ')' );
run;

data sales; /* use the value created by the macro in this step */
set prices;
length saleitem $ 20;
saleitem="&special";
run;
```

このバージョンでは、1 つの DATA _NULL_ステップを使用してマクロ ITEMS を呼び出しています。このステップの実行終了後、ITEMS によって生成された DATA _NULL_ステップが実行されて、マクロ変数 SPECIAL を作成します。その後、DATA SALES ステップが、通常どおり SPECIAL の値を参照します。

DATA ステップおよびマクロ機能での SAS 言語関数の使用

マクロ関数%SYSFUNC および%QSYSFUNC は、SAS 言語関数、および SAS/TOOLKIT ソフトウェアを使用して記述した関数を呼び出して、マクロ機能のテキストを生成できます。%SYSFUNC と%QSYSFUNC には、次の違いがあります。%QSYSFUNC は特殊文字とニーモニックをマスクしますが、%SYSFUNC はそれらをマスクしません。これらの関数の詳細については、“%SYSFUNC 関数と%QSYSFUNC 関数” (273 ページ)を参照してください。

%SYSFUNC の引数は、1 つの SAS 言語関数と、オプションの出力形式です。次の例を参照してください。

```
%sysfunc(date(),worddate.)
%sysfunc(attrn(&dsid,NOBS))
```

%SYSFUNC 内で SAS 言語関数をネストすることはできません。ただし、次のステートメントのように、SAS 言語関数を呼び出す%SYSFUNC 関数をネストすることはできます。

```
%sysfunc(compress(%sysfunc(getoption(sasautos)),%str(%)%('')))
```

この例では、COMPRESS 関数を使用して、SASAUTOS=システムオプションの値から、左かっこ、右かっこ、および一重引用符を除去した結果を返します。%STR 関数が使用され、一致しないかっこと引用符にパーセント記号(%)のマークが付けられていることに注意してください。

%SYSFUNC 内の SAS 言語関数の引数は、すべてカンマで区切る必要があります。OF というワードで始まる引数リストは使用できません。

%SYSFUNC はマクロ関数であるため、SAS 言語関数で行うように、文字値を引用符で囲む必要はありません。たとえば、OPEN 関数を単独で使用する場合、引数を引用符で囲みますが、OPEN 関数を%SYSFUNC 内で使用する場合、引数に引用符は不要です。

次の例は、関数を単独で使用する場合と%SYSFUNC 内で使用する場合を比較しています。

- `dsid = open("sasuser.houses","i");`
- `dsid = open("&mydata","&mode");`
- `%let dsid = %sysfunc(open(sasuser.houses,i));`
- `%let dsid = %sysfunc(open(&mydata,&mode));`

%SYSFUNC と%QSYSFUNC を使用して、DATA ステップのすべての SAS 関数を呼び出すことができます(ただし、表 表 17.2 (274 ページ)に示された SAS 関数を除きます)。マクロ機能では、%SYSFUNC によって呼び出された SAS 言語関数は、最大で 32000 の長さの値を返すことができます。ただし、DATA ステップ内では、戻り値の長さはデータセットの文字変数の長さに制限されます。

%SYSCALL マクロステートメントを使用すると、SAS 言語の CALL ルーチンをマクロプロセッサで使用できます。これについては、“マクロステートメント” (287 ページ)で説明されています。

SQL プロシジャとのインターフェイス

PROC SQL の使用

構造化照会言語(SQL)は、データベースやリレーショナルテーブル内のデータの取り出しや更新を行うために広く使用されている標準化された言語です。SAS ソフトウェアの SQL プロセッサを使用して、次のことを実行できます。

- テーブルおよびビューの作成
- テーブルに格納されたデータの検索
- SQL ビューおよび SAS/ACCESS ビューに格納されたデータの検索
- テーブル内の値の追加または変更
- SQL ビュー内および SAS/ACCESS ビュー内の値の追加または変更

INTO 句

SAS マクロ変数を作成するために、SQL には、SELECT ステートメントの INTO 句が用意されています。1 つの INTO 句を使用して、複数のマクロ変数を作成できます。INTO 句は、%LET ステートメントと同じスコープ規則に従います。マクロ変数の作成方法の概要については、“マクロ変数” (21 ページ)を参照してください。INTO 句に関する詳細と例については、“INTO 句” (283 ページ)を参照してください。

ジョブの実行の制御

PROC SQL には、次を実行するマクロツールも用意されています。

- エラーが発生した場合のジョブの実行停止
- データ値に基づく、プログラムの条件付き実行

SQL によって作成され、ジョブの実行に影響を与えるマクロ変数に関する情報を、次の表に示します。

表 8.2 ジョブの実行に影響を与えるマクロ変数

マクロ変数	説明
SQLEXITCODE	SQL の挿入失敗などの、ある種の障害により発生する、最上位のリターンコード格納されます。このリターンコードは、PROC SQL の終了時に SYSERR マクロ変数に書き込まれます。
SQLOBS	SELECT ステートメントによって生成された行(オブザベーション)の数が格納されます。
SQLOOPS	PROC SQL の内部ループによって処理された反復の回数が格納されます。
SQLRC	SQL ステートメントからのリターンコードが格納されます。リターンコードについては、SAS SQL のドキュメントを参照してください。

マクロ変数	説明
SQLXMSG	パススルー機能が返したエラーに関する説明、および DBMS 固有のリターンコードが格納されます。
SQLXRC	パススルー機能が返した DBMS 固有のリターンコードが格納されます。

SAS コンポーネント言語とのインターフェイス

SCL プログラムの使用

SAS マクロ機能を使用して、SCL プログラムのマクロおよびマクロ変数を定義できます。その後、マクロと SCL プログラムの他の部分との間で、パラメータを渡すことができます。また、自動呼び出しマクロ機能とコンパイル済みマクロ機能を使用することで、マクロを複数の SCL プログラムで使用できます。

注: マクロモジュールは、シンボルとマクロのクォーティングが必要になる場合があるため、プログラムセグメントよりも管理が複雑になることがあります。さらに、モジュールをマクロとして実装しても、コンパイル済み SCL コードのサイズは減少しません。マクロによって生成されたプログラムステートメントは、コンパイル済みコードに追加されます。これは、それらの行をプログラム内のその場所に記述するのと同じことです。

次の表に、SCL マクロ機能インターフェイスを示します。

表 8.3 *マクロ機能との SCL インターフェイス*

カテゴリ	ツール	説明
読み込みまたは書き込み	SYMGET	SCL の実行中に、グローバルマクロ変数の値を返します。
	SYMGETN	グローバルマクロ変数の値を数値として返します。
	CALL SYMPUT	SCL で生成された値をグローバルマクロ変数に割り当てます。
	CALL SYMPUTN	数値をグローバルマクロ変数に割り当てます。

注: SYMPUTN を使用して割り当てられていない値を、SYMGETN を使用して取り出すのは効率的ではありません。CALL SYMPUTN を使用して作成されたマクロ変数を、&を使用して参照することも効率的ではありません。代わりに、SYMGETN を使用してください。さらに、SYMGETN および CALL SYMPUTN を、数値以外の値で使用することも効率的ではありません。

これらの要素の詳細については、“マクロの DATA ステップ CALL ルーチン” (225 ページ) と“マクロの DATA ステップ関数” (235 ページ)を参照してください。

マクロ参照の SCL による置換

マクロ機能を SCL で使用する場合に覚えておくべき重要な点は、SCL プログラム内のマクロ参照とマクロ変数参照が、アプリケーションの実行時ではなく、SCL プログラムのコンパイル時に置換されるということです。マクロとマクロ変数の割り当ておよび置換をさらに細かく制御するには、次の手法を適用します。

- SCL プログラムの実行時にマクロ変数に値を割り当て、それを取り出す場合、SCL プログラム内で CALL SYMPUT および CALL SYMPUTN を使用します。
- SCL プログラムの実行時にマクロ呼び出しまたはマクロ変数参照を置換する場合、SCL プログラム内で SYMGET および SYMGETN を使用します。

サブミットブロックのマクロ変数の参照

SCL では、マクロ変数参照は、サブミットブロックに含まれていなければ、コンパイル時に置換されます。SCL は、アンパサンド(&)の接頭語が付いた名前をサブミットブロック内で検出すると、アンパサンドの後ろの名前が SCL 変数名であるかどうかをチェックします。SCL 変数名である場合、SCL は、サブミットブロックの内のその変数参照を、対応する変数の値に置換します。アンパサンドの後ろの名前がどの SCL 変数名とも一致しない場合、その名前がそのまま(アンパサンドを含め)サブミットされるステートメントに含まれて渡されます。SAS は、ステートメントを処理するときに、その名前をマクロ変数参照として置換しようとします。

サブミットされるステートメント内の名前がマクロ変数参照として渡されることを保証するには、名前の前にアンパサンドを 2 つ付けます(たとえば、&&DSNAME)。同じ名前のマクロ変数と SCL 変数がある場合、参照に 1 つのアンパサンドを付けると SCL 変数として置換されます。強制的にマクロ変数として置換するには、アンパサンドを 2 つ(&&)付けて参照します。

SCL プログラム間でのマクロの共有の考慮事項

SCL プログラム間でマクロを共有すると役に立ちますが、構成管理上の問題が発生する可能性もあります。マクロを複数のプログラムで使用する場合、マクロを更新したときにそれらすべてを再コンパイルできるようにするために、そのマクロを使用するすべてのプログラムを把握しておく必要があります。SCL がコンパイルされるため、マクロを更新した場合、そのマクロを呼び出している各 SCL プログラムを必ず再コンパイルする必要があります。

注意:

SCL プログラムを再コンパイルしてください。 マクロを更新したときに SCL プログラムを再コンパイルしなかった場合、コンパイル済み SCL とソースが一致しなくなる危険があります。

SCL プログラムのマクロ使用例

この SCL プログラムは、BORROWED、INTEREST、および PAYMENT の各フィールドを使用するアプリケーションの例です。このプログラムは、マクロ CKAMOUNT および CKRATE を使用して、ユーザーが各フィールドに入力した値を検証します。このプログラムは、入力された金利(INTEREST)と合計金額(BORROWED)の値を使用して、支払額を計算します。

```
/* Display an error message if AMOUNT */
/* is less than zero or larger than 1000. */
%macro ckamount(amount);
if (&amount < 0) or (&amount > 1000) then
do;
erroron borrowed;
_msg_='Amount must be between $0 and $1,000.';
stop;
end;
else erroroff borrowed;
%mend ckamount;

/* Display an error message if RATE */
/* is less than 0 or greater than 1.5 */
%macro ckrate(rate);
if (&rate < 0) or (&rate > 1) then
do;
erroron interest;
_msg_='Rate must be between 0 and 1.5';
stop;
end;
else erroroff interest;
%mend ckrate;

/* Open the window with BORROWED at 0 and INTEREST at .5. */
INIT:
control error;
borrowed=0;
interest=.5;
return;

MAIN:
/* Run the macro CKAMOUNT to validate */
/* the value of BORROWED. */
%ckamount(borrowed)
/* Run the macro CKRATE to validate */
/* the value of INTEREST. */
%ckrate(interest)
/* Calculate payment. */
payment=borrowed*interest;
return;

TERM:
return;
```

SAS/CONNECT インターフェイス

%SYSRPUT を SAS/CONNECT と共に使用する

%SYSRPUT マクロステートメントは、SAS/CONNECT を使用してリモートホストにサブミットされ、リモートホストに格納されたマクロ変数の値を取り出します。%SYSRPUT ステートメントは、その取得した値をローカルホスト上にあるマクロ変数に割り当てます。%SYSRPUT ステートメントは、マクロ変数に値を割り当てるという意味では、%LET マ

クロステートメントに似ています。ただし、%SYSRPUT ステートメントは、同ステートメントが処理されるリモートホスト上の変数に対してではなく、ローカルホスト上の変数に値を割り当てます。%SYSRPUT ステートメントは、マクロ変数をローカルホストの現在のスコープ内に配置します。

注: リモートホストおよびローカルホストのマクロ変数の名前の先頭に、アンパサンドを付けることはできません。

%SYSRPUT ステートメントは、自動マクロ変数 SYSINFO の値を取得し、その値をローカルホストに渡す場合に使用すると便利です。SYSINFO には、一部の SAS プロシジャが出力したリターンコードの情報が格納されます。SAS/CONNEC の UPLOAD プロシジャと DOWNLOAD プロシジャは、どちらもマクロ変数 SYSINFO を更新できます。これらのプロシジャがエラーで終了した場合、この変数には 0 以外の値が設定されます。リモートホスト上で%SYSRPUT ステートメントを使用すると、SYSINFO マクロ変数の値をローカル SAS セッションに送り返すことができます。このようなジョブをリモートホストに対してサブミットすることにより、リモートホストまたはローカルホスト上で別のステップを開始する前に、PROC UPLOAD ステップまたは PROC DOWNLOAD ステップが正常終了したかどうかをチェックできます。

%SYSRPUT を使用するには、SAS コマンドを使用して DMR オプションをサブミットすることによって、リモートの SAS ウィンドウ環境のセッションを呼び出している必要があります。%SYSRPUT の使用の詳細については、SAS/CONNECT のドキュメントを参照してください。

リモートホスト上またはサーバ上で、新しいマクロ変数を作成したり、既存のマクロ変数の値を変更したりするには、%SYSLPUT マクロステートメントを使用します。

%SYSRPUT を使用したリモートホストのリターンコード値のチェック例

この例では、ファイルをダウンロードし、ステップが成功したことを示す情報を返す方法について説明します。リモート処理が完了すると、このジョブは RETCODE に格納されたリターンコードの値をチェックします。リモート処理が成功した場合、ローカルホスト上の処理を続行します。次の例では、PROC DOWNLOAD ステップの後に%SYSRPUT ステートメントが実行されます。SYSINFO が返す値によって、PROC DOWNLOAD ステップの実行が成功したことが示されます。

```
/* This code executes on the remote host. */
rsubmit;
proc download data=remote.mydata out=local.mydata;
run;
/* RETCODE is on the local host. */
/* SYSINFO is on the remote host. */
%sysrput retcode=&sysinfo;
endrsubmit;

/* This code executes on the local host. */
%macro checkit;
%if &retcode = 0 %then
%do;
further processing on local host
%end;
%mend checkit;

%checkit
```

リモートホスト上で実行されたステップが成功したかどうか確認するには、%SYSRPUT マクロステートメントを使用して、自動マクロ変数 SYSERR の値をチェックします。

%SYSRPUT ステートメントの詳細および構文については、“%SYSRPUT ステートメント” (326 ページ)を参照してください。

SAS/CONNECT での%SYSLPUT の使用

%SYSLPUT ステートメントは、クライアントセッションでサブミットされるマクロステートメントです。これによって、クライアントセッションにおいて使用可能な値を、サーバセッションからアクセスできるマクロ変数に割り当てます。複数のサーバセッションにサインオンしている場合、%SYSLPUT は、最後に使用されたサーバセッションにマクロ割り当てステートメントをサブミットします。サインオンしているサーバセッションが 1 つしかない場合、%SYSLPUT は、マクロ割り当てステートメントをそのサーバセッションにサブミットします。どのセッションにもサインオンしていない場合、エラーが返されます。%SYSLPUT ステートメントは、%LET ステートメントと同様にマクロ変数に値を割り当てます。%LET とは異なり、%SYSLPUT ステートメントは、そのステートメントが実行されたクライアントセッションではなく、サーバセッションの変数に値を割り当てます。%SYSLPUT ステートメントは、サーバセッションのグローバルシンボルテーブルにマクロ変数を格納します。

%SYSLPUT の使用の詳細については、SAS/CONNECT のドキュメントを参照してください。

%SYSLPUT の使用例

%SYSLPUT を使用することで、サーバセッションで実行されるマクロが使用する変数に、動的に値を割り当てることができます。ここでは、マクロステートメント%SYSLPUT は、クライアント側のマクロ変数 RUNID の値を使用して、サーバセッションのマクロ変数 REMID の作成に使用されています。変数 REMID は、サーバセッションで実行されるマクロ%DOLIB によって使用され、サーバセッションで使用されているオペレーティングシステム固有のライブラリ割り当てが検出されます。

```
%macro assignlib (runid);

signon rem &runid
%syslput remid=&runid
rsubmit rem &runid
%macro dolib;
%if (&runid eq 1) %then %do;
libname mylib 'h:';
%end;
%else %if (&runid eq 2) %then %do;
libname mylib '/afs/some/unix/path';
%end;
%mend;
%dolib;
endrsubmit;

%mend;
```

9 章

マクロの保存および再利用

マクロの保存および再利用

マクロ定義をサブミットすると、デフォルトでは、マクロプロセッサはそのマクロをコンパイルし、WORK ライブラリ内の SAS カタログに格納します。それらのマクロは、*セッションコンパイル済みマクロ*と呼ばれ、現在の SAS セッションが存続している間だけ、存在します。セッション間で頻繁に使用されるマクロを保存するには、自動呼び出しマクロ機能またはコンパイル済みマクロ機能を使用します。

自動呼び出しマクロ機能は、SAS マクロのソースを、*自動呼び出しライブラリ*と呼ばれる外部ファイルの集合に格納します。自動呼び出し機能は、さまざまなアプリケーションおよびユーザーからアクセスできる場所に、マクロを容易に管理できるプールを作成する場合に役立ちます。自動呼び出しライブラリは、まとめて連結できます。自動呼び出し機能の主なデメリットは、自動呼び出しマクロが、最初にセッションで呼ばれたときにマクロプロセッサによってコンパイルされることです。このコンパイルはオーバーヘッドになります。このオーバーヘッドは、コンパイル済みマクロ機能を使用することで回避できます。

コンパイル済みマクロ機能は、指定した SAS ライブラリの SAS カタログに、コンパイルされたマクロを格納します。コンパイル済みマクロを使用することで、プロダクションレベルのジョブにおいて、マクロをコンパイルする時間を省くことができます。ただし、これらのマクロはコンパイルされて格納されるため、マクロ定義のソースを別の場所に保存して管理する必要があります。

自動呼び出しマクロ機能とコンパイル済みマクロ機能には、それぞれメリットがあります。マクロ定義の保存方法を決定する要因には、次のものがあります。

- マクロを使用する頻度

- マクロを変更する頻度
- マクロを実行する必要のあるユーザーの数
- マクロに含まれるコンパイルされるマクロステートメントの数

新しいプログラムを開発している場合、マクロを作成し、それらを現在のセッション中にコンパイルすることを検討してください。ネームスタイルマクロを使用してプロダクションレベルのジョブを実行している場合、コンパイル済みマクロの使用を検討してください。ユーザーグループでマクロを共有している場合、自動呼び出し機能の使用を検討してください。

注: コンパイル済みマクロ機能を使用する場合、さらに効率を高めるために、ネームスタイルマクロのみを格納してください。ステートメントスタイルマクロおよびコマンドスタイルマクロは、効率が劣ります。

コンパイル済みマクロまたは自動呼び出しマクロをプログラミングする場合、そのマクロ内でのみ使用されるマクロ変数を、%LOCAL ステートメントを使用して定義することをお勧めします。そうしないと、現在のマクロの外部で定義されたマクロ変数の値は、変更される可能性があります。マクロ変数のスコープの説明については、“マクロ変数のスコープ” (45 ページ)を参照してください。

通常、SAS マクロ機能では、マクロ名と変数名の大文字小文字は区別されず、内部で大文字に変更されます。SAS マクロ機能では、値の大文字小文字は区別され、変更されません。

自動呼び出しマクロまたはコンパイル済みマクロを呼び出すと、マクロ名が大文字に変更されてカタログルーチンに渡され、その名前のメンバが開かれます。カタログルーチンはホストに依存しており、メンバを検索するときに、特定のホストのデフォルトの大文字小文字規則を使用します。マクロカタログエントリは、対象となるホストのデフォルトの大文字小文字規則を使用して作成する必要があります。各ホストのデフォルトは次のとおりです。

- UNIX のデフォルトは小文字
- z/OS のデフォルトは大文字
- Windows のデフォルトは小文字

注: UNIX では、自動呼び出しマクロが格納されるメンバの名前をすべて小文字にする必要があります。

自動呼び出しライブラリへのマクロの保存

自動呼び出しライブラリの概要

通常、自動呼び出しライブラリは、個別のファイルを含むディレクトリです。それぞれのファイルには、1 つのマクロ定義が保存されます。SAS 6.11 からは、自動呼び出しライブラリを SAS カタログにすることもできるようになりました。(SAS カタログを自動呼び出しライブラリとして使用する場合の詳細については、次のセクションを参照してください。)

動作環境の情報
*さまざまなホスト上の自動呼び出しライブラリディレクトリ*という用語は、ホストオペレーティングシステムによって管理されるファイル(またはメンバ)の集合的な保存場所のことを指します。さまざまなホストオペレーティングシステムは、さまざまな名前(ディレクトリ名、サブディレクトリ名、マクロライブラリ名、テキストライブラリ名、

区分データセット名など)を使用して集合的な保存場所を識別します。詳細については、使用しているオペレーティングシステムに関する SAS ドキュメントを参照してください。

ディレクトリを自動呼び出しライブラリとして使用する

ディレクトリを SAS 自動呼び出しライブラリとして使用するには、次の手順を実行します。

1. ライブラリメンバを作成するために、各マクロのソースコードを、ディレクトリ内の個別のファイルに保存します。ファイルの名前は、マクロ名と同じにする必要があります。たとえば、%SPLIT をサブミットすることで呼び出されるマクロを定義するステートメントは、SPLIT というファイル名で保存する必要があります。

   動作環境の情報
   *自動呼び出しライブラリのメンバ名*拡張子付きのファイル名を使用できるオペレーティングシステムの場合、自動呼び出しマクロライブラリのメンバには、特殊な拡張子(通常は`*.sas`)付きの名前を付ける必要があります。システムにある SAS が提供した自動呼び出しマクロを調べて、対象のサイトで、マクロを保存したファイルの名前に特殊な拡張子を付ける必要があるかどうかを確認してください。z/OS オペレーティングシステムの場合、マクロ名を、PDS メンバの名前として割り当てる必要があります。

2. SASAUTOS システムオプションを設定して、ディレクトリを自動呼び出しライブラリとして指定します。ほとんどのホストでは、起動時に、予約済みのファイル参照名 SASAUTOS が、SAS から提供された自動呼び出しライブラリ、またはサイトで指定した別の自動呼び出しライブラリに割り当てられます。1 つ以上の自動呼び出しライブラリを指定する場合、それらのライブラリのマクロをすべて使用できるようにするために、SAS が提供した自動呼び出しライブラリとサイトで指定した自動呼び出しライブラリを必ず連結してください。詳細については、使用しているホストのドキュメントおよび “SASAUTOS=システムオプション” (362 ページ)を参照してください。

自動呼び出しライブラリにファイルを保存する場合、次の点に注意してください。

- 自動呼び出しライブラリに配置するファイルの種類に制限はありませんが、混乱を避け、管理を容易にするために、自動呼び出しライブラリファイルのみを保存するようにしてください。
- 自動呼び出しライブラリのメンバには、複数のマクロ定義に加えて、オープンコードを含めることもできますが、通常は、どの自動呼び出しライブラリのメンバも 1 つのマクロのみを含むようにしてください。複数のマクロを同じマクロライブラリのメンバに含める必要がある場合、関連するマクロを一緒に含めます。

SAS カタログを自動呼び出しライブラリとして使用する

SAS 6.11 以降、CATALOG アクセスメソッドを使用して、自動呼び出しマクロを SAS カタログの SOURCE エントリとして格納できるようになりました。SAS カタログを使用して自動呼び出しライブラリを作成するには、次の手順に従います。

1. LIBNAME ステートメントを使用して、ライブラリ参照名を SAS ライブラリに割り当てます。
2. CATALOG 引数を付けて FILENAME ステートメントを使用し、自動呼び出しマクロを格納するカタログにファイル参照名を割り当てます。たとえば、次のコードは、

MYMACS.MYAUTOS という名前のカタログを指すファイル参照名 MYMACROS を作成しています。

```
libname mymacs 'SAS-data-library';
filename mymacros catalog 'mymacs.myautos';
```

3. 各マクロのソースコードを、SAS カタログ内の SOURCE エントリに格納します。(SOURCE はエントリタイプです。)SOURCE エントリの名前は、マクロ名と同じにする必要があります。
4. SASAUTOS システムオプションを設定して、ファイル参照名を自動呼び出しライブラリとして指定します。詳細については、“SASAUTOS=システムオプション” (362 ページ)を参照してください。

自動呼び出しマクロの呼び出し

自動呼び出しマクロを呼び出すには、システムオプション MAUTOSOURCE を設定し、SASAUTOS に値を割り当てる必要があります。MAUTOSOURCE によって自動呼び出し機能を有効にし、SASAUTOS によって自動呼び出しライブラリを指定します。詳細については、“MAUTOSOURCE システムオプション” (341 ページ) および “SASAUTOS=システムオプション” (362 ページ)を参照してください。

必要なオプションが設定されていれば、自動呼び出しマクロの呼び出しは、現在のセッション中に作成したマクロの呼び出しと同様です。ただし、呼び出すマクロをマクロプロセッサがどのように検索するかについて、理解しておくことが重要です。マクロを呼び出すと、マクロプロセッサは次のタスクを実行します。

- セッション中にコンパイルされたマクロ定義を検索します。
- MSTORED オプションが設定されている場合、SASMSTORE オプションで指定されたライブラリ内のコンパイル済みマクロ定義を検索します。
- MAUTOSOURCE オプションが設定されている場合、SASAUTOS オプションで指定された自動呼び出しライブラリ内のメンバを指定された順序で検索します。
- SAS 製品のコンパイル済みマクロ定義を、SASHELP ライブラリから検索します。

自動呼び出しライブラリ内で対象のマクロ名を持つライブラリメンバが見つかると、マクロプロセッサは次を実行します。

- そのメンバ内のあらゆるマクロ定義を含むすべてのソーステートメントをコンパイルし、その結果をセッションカタログに格納します。
- そのメンバにオープンコード(どのマクロ定義にも含まれないマクロステートメントまたは SAS ソーステートメント)があれば、それを実行します。
- 呼び出した名前の付いたマクロを実行します。

注: 自動呼び出しライブラリのメンバに複数のマクロが含まれている場合、マクロプロセッサはすべてのマクロをコンパイルしますが、実行されるのは呼び出した名前の付いたマクロのみです。

マクロと同じ自動呼び出しライブラリのメンバにオープンコードステートメントが含まれている場合、それらのステートメントは、マクロを最初に呼び出したときにのみ実行されます。その後、同じセッション内でマクロを呼び出すとコンパイル済みマクロが実行されますが、それにはコンパイル済みマクロ定義のみが含まれ、自動呼び出しマクロのソースファイルに含まれていた他のコードは含まれていません。

SAS セッション中に SASAUTOS を変更することは推奨されません。実行中の SAS セッションで SASAUTOS=指定を変更した場合、未コンパイルの自動呼び出しマクロが

呼び出され、開かれていたすべてのライブラリが閉じられ、新たに指定した開くことのできるすべてのライブラリが開かれるまで、新しい指定は格納されません。

自動呼び出しマクロのデバッグの詳細については、“マクロ機能のエラーメッセージとデバッグ” (121 ページ)を参照してください。

コンパイル済みマクロ機能を使用したマクロの保存

コンパイル済みマクロ機能の概要

コンパイル済みマクロ機能は、マクロをコンパイルして、指定されたライブラリ内の永続的なカタログに保存します。コンパイルは一度だけ実行されます。現在またはその後のセッションでコンパイル済みマクロを呼び出すと、マクロプロセッサはコンパイル済みのコードを実行します。

SAS 9.1.3 以降では、コンパイル済みマクロカタログが、最初に読み込み専用で開かれます。コンパイル済みマクロをコンパイルまたは更新しようとする場合、このカタログは即座に閉じられ、更新の目的で再び開かれます。マクロがコンパイルされてカタログの更新または変更が完了すると、カタログが再び即座に閉じられ、読み込み専用で再び開かれます。

マクロ定義のコンパイルと保存

永続的なカタログ内のマクロ定義をコンパイルするには、まず、それぞれのコンパイル済みマクロのソースを作成する必要があります。コンパイル済みマクロを格納するには、次の手順を実行します。

1. %MACRO ステートメントで STORE オプションを使用します。SOURCE オプションを使用すると、ソースコードとコンパイル済みコードを格納できます。さらに、DES= オプションを指定することによって、SAS カタログ内のマクロエントリに説明的なタイトルを割り当てることができます。例として、次の定義の%MACRO ステートメントに、STORE、SOURCE、DES=の各オプションを示します。

```
%macro myfiles / store source
des='Define filenames';
filename file1 'external-file-1';
filename file2 'external-file-2';
%mend;
```

注意:

マクロソースコードを保存しておいてください。 コンパイル済みマクロからソースステートメントを再作成することはできません。したがって、マクロを変更する場合、マクロの元のソースステートメントが保存されている必要があります。また、すべてのコンパイル済みマクロについて、マクロソースコードを文書化してください。%MACRO ステートメントで SOURCE オプションを使用して、コンパイル済みコードと共にソースコードを保存できます。あるいは、ソースを別のファイルに保存することもできます。別のファイルにソースを保存する場合、そのソースコードをコンパイル済みマクロと同じカタログ内に保存することをお勧めします。次の例では、ソースコードは次のライブラリに保存されます。

```
mylib.sasmacro.myfiles.source
```

注: コンパイル済みマクロのソースを取り出す場合、“%COPY ステートメント” (292 ページ)を参照してください。

2. MSTORED システムオプションを設定して、コンパイル済みマクロ機能を有効にします。詳細については、“MSTORED システムオプション” (360 ページ)を参照してください。
3. SASMSTORE オプションを割り当てることによって、コンパイル済み SAS マクロのカタログが含まれる、または含まれる予定の SAS ライブラリを指定します。たとえば、コンパイル済みマクロを MYLIB.SASMACR という名前の SAS カタログに格納したり、それを呼び出したりするには、次のステートメントをサブミットします。

   ```
   libname mylib 'SAS-data-library';
   options mstored sasmstore=mylib;
   ```

   詳細については、“SASMSTORE=システムオプション” (363 ページ)を参照してください。
4. コンパイルして永続的に格納するマクロごとに、ソースをサブミットします。

コンパイル済みマクロを、別のオペレーティングシステムまたは SAS の別のリリースに移動することはできません。ただし、マクロソースコードを、別のオペレーティングシステムまたは SAS の別のリリースに移動して、そこでコンパイルして格納することは可能です。詳細については、ホストのドキュメントを参照してください。

SAS 提供の自動呼び出しマクロの保存

SAS が提供する自動呼び出しライブラリに含まれるマクロを使用する場合、独自に作成したマクロに加えて、それらのマクロをコンパイルし、格納することによって、マクロをコンパイルする時間を省くことができます。SAS が提供する自動呼び出しライブラリに含まれる Base SAS ソフトウェアに関連する多くのマクロは、SAS が提供する自動呼び出しマクロ COMPSTOR を使用してコンパイルし、SASMACR という名前の SAS カタログに格納できます。詳細については、“%COMPSTOR 自動呼び出しマクロ” (179 ページ)を参照してください。

コンパイル済みマクロの呼び出し

必要なシステムオプションが設定されていれば、コンパイル済みマクロの呼び出しは、セッション中にコンパイルされたマクロの呼び出しと全く同じです。ただし、マクロプロセッサがマクロをどのように検索するかについて、理解しておくことが重要です。マクロを呼び出すと、マクロプロセッサは、次の順序でマクロ名を検索します。

1. 現在のセッション中にコンパイルされたマクロ
2. 指定されたライブラリ内の SASMACR カタログに含まれるコンパイル済みマクロ (オプション MSTORED および SASMSTORE=が有効の場合)
3. SASAUTOS オプションで指定された各自動呼び出しライブラリ(オプション SASAUTOS=および MAUTOSOURCE が有効の場合)
4. SASHELP ライブラリ内の SASMACR カタログに含まれるコンパイル済みマクロ

カタログ内のコンパイル済みマクロを含むエントリを表示できます。詳細については、“マクロ機能のエラーメッセージとデバッグ” (121 ページ)を参照してください。

10 章

マクロ機能のエラーメッセージとデバッグ

マクロのデバッグに関する一般情報

層化アプローチでのマクロの開発

マクロ機能は非常に強力なツールであるため、複雑なツールでもあり、大規模なマクロアプリケーションのデバッグに極端に時間がかかって、フラストレーションがたまるこ

とがあります。したがって、エラーを最小化するような方法でマクロアプリケーションを開発することは理にかなっています。そうすることで、発生するエラーを可能な限り簡単に検出および修正できます。その最初の手順は、発生する可能性のあるエラーの種類と、発生する状況を理解することです。次に、モジュール化された層化アプローチを使用してマクロを開発します。最後に、システムオプション、自動マクロ変数、%PUT ステートメントなど、いくつかの組み込みツールを使用してエラーを診断します。

注: 置換されなかったマクロ名およびマクロ変数に関する特定の重要な警告メッセージを表示するには、"SERROR システムオプション" (364 ページ) および "MERROR システムオプション" (346 ページ) を必ず有効にしてください。これらのシステムオプションについては、"マクロのシステムオプション" (335 ページ) を参照してください。

エラーの発生

ワードスキャナは、プログラムの処理中に&または%の形式のトークンを検出すると、マクロプロセッサを起動して、&または%の後ろに続くトークン名を調べます。マクロプロセッサは、検出したトークンに応じて、次のアクティビティのいずれかを開始します。

- マクロ変数の置換
- マクロのオープンコードの処理
- マクロのコンパイル
- マクロの実行

エラーは、これらのステージのいずれかが実行されているときに発生する可能性があります。たとえば、マクロ関数名のスペルを間違えたり、必要なセミコロンを付け忘れたりした場合、コンパイル中に*構文エラー*が発生します。構文エラーは、プログラムステートメントがマクロ言語の規則に従わない場合に発生します。また、スコープの範囲外の変数を参照した場合、*マクロ変数置換エラー*が発生します。*実行エラー*(*セマンティックエラー*とも呼ばれます)は、通常、プログラムロジック内で発生します。たとえば、マクロによって生成されたテキストに間違った論理(間違った順序または予期しない方法で実行されるステートメント)が含まれる場合、実行エラーが発生することがあります。

もちろん、マクロコードに問題がなくても、マクロコード以外の SAS コードでエラーが発生しないとも限りません。たとえば、次のようなエラーが発生する場合があります。

- ライブラリ参照名が未定義
- オープンコード(つまり、マクロ定義の外部)での構文エラー
- マクロによって生成されたコード内のタイプミス

通常、番号付きのエラーメッセージは、マクロコード以外の SAS コードに関するエラーメッセージです。マクロプロセッサによって生成されたエラーメッセージには、番号は付いていません。

バグのないマクロの開発

どの言語でプログラミングする場合でも、コードをモジュールに分けて開発することは、優れた手法です。つまり、大きな 1 つのプログラムを記述するのではなく、部品に分けて開発し、各部品を別々にテストしてから、それらを結合する手法です。この手法は、マクロアプリケーションを開発する場合、特に役立ちます。これは、SAS マクロには、マクロコードとマクロコードによって生成される SAS コードという 2 つの部分に分かれた性質があるためです。

マクロコードをサブミットする前に、よくある誤りがマクロコードに含まれていないかどうかを校正するのも良い方法です。

次に、主要なチェック項目の一部を示します。

- %MACRO ステートメントと%MEND ステートメントに含まれる名前が一致しており、各%MACRO ステートメントに対応する%MEND ステートメントが存在すること。
- %DO ステートメントの数が%END ステートメントの数と一致すること。
- 反復する%DO ステートメントに対応する、適切な%TO 値が存在すること。
- すべてのステートメントがセミコロンで終わっていること。
- コメントが正しく始まり、終了しており、一致しない一重引用符を含んでいないこと。
- マクロ変数参照が&で始まり、マクロステートメントが%で始まっていること。
- CALL SYMPUT によって作成されたマクロ変数が、それらが作成されたのと同じ DATA ステップ内で参照されていないこと。
- 即座に実行されるステートメント(%LET など)が、DATA ステップの条件付きロジックに含まれていないこと。
- マクロ変数参照の前後で一重引用符を使用していないこと(TITLE ステートメントや FILENAME ステートメントなど)。マクロ変数参照を、クォーティングされた文字内で使用する場合、二重引用符で囲まれた文字列内のマクロ変数参照のみが置換されます。
- マクロ変数値に、算術演算子として解釈できるどのキーワード、文字も含まれていないこと(そのような文字を含める場合は、該当するマクロクォーティング関数を使用してください)。
- マクロ変数、%GOTO ラベル、およびマクロ名が、SAS とホスト環境の予約済みキーワードと競合していないこと。

マクロのトラブルシューティング

よくあるマクロ問題を解決する

マクロ機能を操作したときに発生する可能性のある問題の一部を、次の表に示します。これらの問題の多くは、エラーメッセージが SAS ログに書き込まれないため、解決することが困難です。表には、問題ごとに考えられる原因と解決策を示します。

表 10.1 *よく発生するマクロ問題*

問題	原因	説明
マクロ定義のサブミット後、SAS ウィンドウ環境のセッションが応答しなくなる。コードを入力してサブミットしても、何も起こらない。	• %MEND ステートメントの構文エラー • セミコロン、かっこ、または引用符の欠損 • %MEND ステートメントの欠損 • コメントが閉じられていない	%MEND ステートメントが認識されないため、すべてのテキストがマクロ定義の一部になっています。

問題	原因	説明
マクロの呼び出し後、SAS ウィンドウ環境のセッションが応答しなくなる。	呼び出しに関するエラー。パラメータ付きで定義されたマクロを呼び出す場合の、1 つ以上のパラメータの指定もれ、かっこの不足など。	マクロ機能は、ユーザーが呼び出しを終了するまで待機します。
マクロをサブミットしてもコンパイルされない。	マクロ定義内のどこかに構文エラーがある。	コンパイルされるのは、構文的に正しいマクロのみです。
マクロを呼び出しても実行されないか、一部実行されてから停止する。	• 不正な値が(たとえばパラメータとして)マクロに渡された。 • マクロ定義内のどこかに構文エラーがある。	マクロは、受け取るパラメータの数が正しく、パラメータのタイプが正しい場合にのみ、正常に実行されます。
マクロは実行されるが、SAS コードが不正な結果を返すか、結果を何も返さない。	マクロまたは SAS コードのロジックが不正。	
コードが、オープンコードとしてサブミットされると正しく動作するが、マクロによって生成されると動作せず、不明なエラーメッセージを発行する。	• 意図したとおりにトークン化されていない。 • マクロ定義内のどこかに構文エラーがある。	まれに、マクロクォーティング関数が、渡されたテキストのトークン化を変更する場合があります。“%UNQUOTE 関数” (279 ページ)を使用してください。
%MACRO ステートメントが、"無効なステートメント"エラーを生成する。	• MACRO システムオプションを無効にしている。 • マクロ定義内のどこかに構文エラーがある。	マクロ機能が動作するには、MACRO システムオプションが有効である必要があります。SAS 構成ファイルを適宜編集してください。

一般的なマクロエラーと警告メッセージの一部を、次の表に示します。メッセージごとに問題の原因を示し、詳細情報へのリンクを示します。

表 10.2 一般的なマクロエラーメッセージと原因

エラーメッセージ	考えられる原因	詳細情報
`Apparent invocation of macro xxx not resolved.`	• マクロ名にスペルミスがある。 • MAUTOSOURCE システムオプションを無効にしている。 • MAUTOSOURCE は有効だが、SASAUTOS=システムオプションで不正なパス名を指定している。 • 自動呼び出し機能を使用しているが、マクロ名とファイル名に別の名前を与えている。 • 自動呼び出し機能を使用しているが、ファイル名に`.sas` 拡張子を付けていない。 • マクロ定義内に構文エラーがある。	• マクロ名のスペルを確認してください。 • “自動呼び出し機能の問題を解決する” (132 ページ) • “バグのないマクロの開発” (122 ページ)
`Apparent symbolic reference xxx not resolved.`	• マクロ変数を作成した CALL SYMPUT と同じ DATA ステップ内で、そのマクロ変数を置換しようとしている。 • マクロ変数名にスペルミスがある。 • 範囲外のマクロ変数を参照している。 • マクロ変数の末尾にテキストを追加するときに、ピリオドの区切り文字を付け忘れた。	• “タイミングの問題を解決する” (130 ページ) • マクロ変数名のスペルを確認してください。 • “マクロ変数のスコープの問題を解決する” (126 ページ) • “マクロ変数の置換の問題を解決する” (125 ページ) • “マクロ機能について” (3 ページ)のマクロ変数参照の接尾語の生成

マクロ変数の置換の問題を解決する

マクロプロセッサは、`&`が前に付いたトークン名を調べるときに、それに一致するマクロ変数のエントリをマクロシンボルテーブル内で検索します。マクロプロセッサは、一致するエントリを見つけると、シンボルテーブルから関連付けられたテキストを取り出して、入力スタックにある`&`***name*** をそのテキストに置き換えます。マクロ変数名がマクロプロセッサに渡されて、一致するエントリがシンボルテーブル内で検出されなかった場合、入力スタックのトークンは置換されず、次のメッセージが生成されます。

```
WARNING: Apparent symbolic reference
NAME not resolved.
```

置換されないトークンは、SAS の他の部分で使用するために、入力スタックに転送されます。

注: SERROR システムオプションを有効にした場合にのみ、警告メッセージが表示されます。

これらの問題を解決するには、マクロ変数名を正しく記述しており、適切なスコープ内で参照していることを確認します。

マクロ変数は置換されるが、正しい値に置換されない場合、いくつかの項目を確認できます。まず、変数が計算の結果である場合、正しい値を計算に渡していることを確認します。次に、グローバル変数の値を誤って変更していないことを確認します(変数のスコープの問題の詳細については、"マクロ変数のスコープの問題を解決する" (126 ページ)を参照してください)。

別のよくある問題は、マクロ変数の末尾にテキストを追加したが、マクロ変数名の末尾と追加したテキストの先頭を示す区切り文字を、挿入し忘れることです。たとえば、WEEK1、WEEK2 などへの参照を含む TITLE ステートメントを記述する場合を考えます。次に示すように、これらの文字列の前の部分(WEEK)をマクロ変数に設定し、TITLE ステートメント内で WEEK の番号を指定します。

```
%let wk=week;

title "This is data for &wk1"; /* INCORRECT */
```

これらのステートメントをコンパイルすると、マクロプロセッサは、WK ではなく WK1 という名前のマクロ変数を検索します。この問題を修正するには、次のステートメントに示すように、マクロ変数名の末尾と追加したテキストとの間にピリオド(マクロ区切り文字)を追加します。

```
%let wk=week;

title "This is data for &wk.1";
```

注意:

マクロ変数名では、AF、DMS、または SYS を接頭語として使用しないでください。 AF、DMS、および SYS という文字列は、SAS が作成するマクロ変数の接頭語として頻繁に使用されます。SAS では、AF、DMS、または SYS をマクロ変数名の接頭語として使用することは、禁止されていません。ただし、これらの文字列を接頭語として使用すると、ユーザーが指定した名前と SAS が作成したマクロ変数(将来の SAS のリリースでの自動マクロ変数を含む)の名前との間に、競合が発生する恐れがあります。名前の競合が発生した場合、競合の内容によっては、警告メッセージやエラーメッセージが発行されないことがあります。そのため、マクロ名およびマクロ変数名の文字列の先頭には、AF、DMS、または SYS という文字列を使用しないことをお勧めします。

マクロ変数のスコープの問題を解決する

マクロ変数でよく発生する間違いは、マクロ変数のスコープの外にあるローカルマクロ変数の参照に関連しています。"マクロ変数のスコープ" (45 ページ)で説明されているように、マクロ変数はグローバルまたはローカルのいずれかです。スコープ外の変数を参照すると、マクロプロセッサは、その変数参照を置換できません。たとえば、次のプログラムについて考えます。

```
%macro totinv(var);
data inv;
retain total 0;
set sasuser.houses end=final;
total=total+&var;
if final then call symput("macvar",put(total,dollar14.2));
run;
```

```
%put **** TOTAL=&macvar ****;
%mend totinv;

%totinv(price)
%put **** TOTAL=&macvar ****; /* ERROR */
```

これらのステートメントをサブミットすると、マクロ TOTINV 内の%PUT ステートメントによって、TOTAL の値がログに書き込まれます。マクロ呼び出しの後の%PUT ステートメントによって警告メッセージが生成され、テキスト **TOTAL=&macvar** が次のようにログに書き込まれます。

```
TOTAL= $1,240,800.00
WARNING: Apparent symbolic reference MACVAR not resolved.
**** TOTAL=&macvar ****
```

2 番目の%PUT ステートメントは、マクロ変数 MACVAR が TOTINV マクロに対してローカルであるため、実行に失敗します。エラーを修正するには、%GLOBAL ステートメントを使用して、マクロ変数 MACVAR を宣言する必要があります。

マクロ変数でよく発生する別の間違いは、マクロ変数名の重複に関連します。マクロ定義内で、グローバルマクロ変数と同じ名前のマクロ変数を参照した場合、グローバル変数に影響を与えます。その影響は、意図したものではない場合があります。重複しない名前をマクロ変数に与えるか、%LOCAL ステートメントを使用して、変数のスコープをローカルとして明確に定義するようにしてください。この方法の例については、“マクロ変数をローカルにする” (60 ページ) を参照してください。

オープンコードステートメントの再帰問題を解決する

*再帰*とは、自分自身を呼び出すことです。*オープンコードの再帰*は、オープンコードによって、マクロステートメントによる別のマクロステートメントの呼び出しが誤って生じた場合に発生します。この呼び出しは、再帰参照と呼ばれます。オープンコードの再帰を引き起こす最もよくあるエラーは、セミコロンの欠損です。次の例では、%LET ステートメントがセミコロンで終わっていません。

```
%let a=b /* ERROR */
%put **** &a ****;
```

マクロプロセッサは、%LET ステートメント内で%PUT ステートメントを検出すると、次のエラーメッセージを生成します。

```
ERROR: Open code statement recursion detected.
```

通常、オープンコードの再帰エラーは、マクロプロセッサがマクロステートメントを意図されたとおりに読み込まないために発生します。オープンコードの再帰エラーは、ほとんどがコードのタイプミスによって生じ、実行ロジックのエラーではないため、通常は慎重に校正することによって解決できます。

オープンコードの再帰エラーから回復するには、まず、1 つのセミコロンをサブミットしてみます。これで効果がない場合は、次の文字列をサブミットしてみます。

```
*'; *"; *); */; %mend; run;
```

次のメッセージが SAS ログに表示されるまで、この文字列のサブミットを続けます。

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

この方法で効果がない場合は、SAS セッションを閉じて SAS を再起動します。当然ながら、SAS を閉じて再起動すると、保存されていないデータはすべて失われます。マクロの開発中は必ず頻繁に保存するようにし、それらのマクロをサブミットする前に、必ず慎重に校正してください。

マクロ関数の問題を解決する

マクロ関数の問題のよくある原因としては、次のようなものがあります。

- 関数名のスペルミス
- 左かっこまたは右かっこの付け忘れ
- 引数の指定もれ、または余分な引数の指定

マクロ関数に関連するエラーが発生した場合に、他のエラーメッセージも表示されることがあります。それらのメッセージは、入力スタックに無効なトークンが残っていることにより、マクロプロセッサによって生成されます。

次に示す例について考えます。この例では、%SUBSTR 関数を使用して、マクロ変数 LINCOLN の値の一部をマクロ変数 SECONDWD に割り当てようとしています。しかし、2 番目の%LET ステートメントにタイプミスがあり、%SUBSTR が誤って%SUBSRT と記述されています。

```
%macro test;
%let lincoln=Four score and seven;
%let secondwd=%subsrt(&lincoln,6,5); /* ERROR */
%put *** &secondwd ***;
%mend test;

%test
```

この誤りのあるプログラムをサブミットすると、次のメッセージが SAS ログに表示されます。

```
WARNING: Apparent invocation of macro SUBSRT not resolved.
```

このエラーメッセージは、誤って記述された関数名をはっきりと指摘しています。

未置換のマクロの問題を解決する

マクロプロセッサにマクロ名が渡されたが、マクロプロセッサがそれに対応するマクロ定義を検出できなかった場合、次のメッセージが生成されます。

```
WARNING: Apparent invocation of macro
NAME not resolved.
```

このエラーの原因として、次のことが考えられます。

- マクロ名またはマクロ関数名のスペルミス
- マクロ定義内のエラーにより、マクロがダミーマクロとしてコンパイルされた

*ダミーマクロ*とは、マクロプロセッサによって部分的にコンパイルされるが保存されないマクロのことです。

注: MERROR システムオプションを有効にした場合にのみ、この警告メッセージが表示されます。

“ブラックホール”マクロ問題を解決する

マクロプロセッサは、マクロ定義のコンパイルを開始すると、対応する%MEND ステートメントを検出するまでトークンを読み込み、コンパイルします。%MEND ステートメントの記述が抜けている場合、または前のステートメントでセミコロンを付け忘れたことに

よって%MEND ステートメントが認識されなかった場合、マクロプロセッサはトークンのコンパイルを停止しません。サブミットしたすべてのコード行は、マクロの一部になります。

マクロ定義に%MED ステートメントを追加して再サブミットしても、エラーは修正されません。修正したマクロ定義をサブミットすると、マクロプロセッサは、そのマクロ定義を、修正前のマクロ定義内でネストされているものとして扱います。コンパイルを停止するには、マクロプロセッサによって、対応する%MEND ステートメントが検出される必要があります。

注: %MACRO ステートメントと%MEND ステートメントの対応付けが容易になるように、マクロ名を指定して%MEND ステートメントを使用することをお勧めします。

サブミットしたステートメントが SAS によって処理されていないと判断した場合、回復方法がわからなければ、一度に 1 つの%MEND ステートメントをサブミットしてみて、次のメッセージが SAS ログ表示されるまでそれを繰り返します。

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

その後、元のエラーを含むマクロ定義を再び呼び出し、%MEND ステートメントのエラーを修正してから、そのマクロ定義をサブミットしてコンパイルします。

他にも、同様の問題を引き起こす構文エラーがあります。たとえば、一致しない引用符や、閉じていないかっこなどです。多くの場合、これらの構文エラーのいずれかが他の問題を引き起こしています。次の例について考えます。

```
%macro rooms;
/* other macro statements& */
%put **** %str(John's office) ****; /* ERROR */
%mend rooms;

%rooms
```

これらのステートメントをサブミットすると、マクロプロセッサは、マクロ定義 ROOMS のコンパイルを開始します。ところが、%PUT ステートメント内の一重引用符に、パーセント記号のマークが付けられていません。そのため、コンパイル時にマクロプロセッサは、この一重引用符をリテラルトークンの先頭と解釈します。その場合、右かっこ、ステートメントの末尾のセミコロン、またはマクロ定義の最後にある%MEND ステートメントが認識されません。

このエラーから回復するには、次の文字列をサブミットする必要があります。

```
');
%mend;
```

この方法で効果がない場合は、次の文字列をサブミットしてみます。

```
*'; *"; *); */; %mend; run;
```

次のメッセージが SAS ログに表示されるまで、この文字列のサブミットを続けます。

```
ERROR: No matching %MACRO statement for this %MEND statement.
```

エラーが実際に発生する前に、それらを検出した方が、明らかに簡単です。マクロをサブミットしてコンパイルする前に、それらを入念に確認することで、細かい構文エラーを回避できます。構文のチェックリストについては、“バグのないマクロの開発” (122 ページ) を参照してください。

注: 説明の付かない予想外のマクロの動作を引き起こす別の原因として、マクロ変数名またはマクロ名として予約語を使用することが挙げられます。たとえば、SAS で

は SYS で始まる名前が予約されているため、SYS で始まる名前のマクロおよびマクロ変数を作成することはできません。ほとんどのホスト環境にも、予約語があります。たとえば、PC ベースのプラットフォームでは、コンソール入力用として、CON という予約語があります。予約されている SAS のキーワードについては、“ マクロ機能の予約語” (369 ページ) を確認してください。ホスト環境の予約語については、SAS ドキュメントを確認してください。

タイミングの問題を解決する

多くのマクロエラーは、ユーザーが意図したタイミングとは異なるタイミングでマクロ変数が置換された場合、またはユーザーが期待するタイミングでマクロステートメントが実行されなかった場合に発生します。タイミングの重要性を示す主な例として、CALL SYMPUT を使用して DATA ステップ変数をマクロ変数に書き込む場合が挙げられます。このマクロ変数は、それを定義した同じ DATA ステップ内では使用できません。その後のステップ(DATA ステップの RUN ステートメントの後)でのみ使用できます。

タイミングエラーを防ぐには、マクロプロセッサがどのように動作するかを理解することが重要です。簡単に説明すると、コンパイルと実行という 2 つの主なステップがあります。コンパイルステップは、すべてのマクロコードをコンパイル済みコードに置換します。次に、そのコードが実行されます。ほとんどのタイミングエラーは、次の理由によって発生します。

- コンパイル中に起きると期待されていることが、実際には実行されるまで起きない場合。
- 後で起きると期待されていることが、実際には即座に実行された場合。

以降では、コンパイルと実行のタイミングがなぜ重要になるかについて、理解に役立つ 2 つの例を示します。

直ちに実行するマクロステートメントの例

次のプログラムでは、%LET ステートメントと SR_CIT 変数を使用して、データセットに高齢者のデータが含まれているかどうかを示そうとしています。

```
data senior;
set census;
if age > 65 then
do;
%let sr_cit = yes; /* ERROR */
output;
end;
run;
```

しかし、得られる結果は、期待する結果とは異なります。%LET ステートメントは、DATA ステップがコンパイルされている間、つまりデータセットが読み込まれる前に、即座に実行されます。そのため、%LET ステートメントは、IF 条件の結果とは無関係に実行されます。65 よりも大きい AGE の値を持つオブザベーションがデータセットに含まれない場合でも、SR_CIT は常に **yes** になります。

これを解決するには、IF ロジックによって制御し、IF ステートメントが true の場合にのみ実行するという方法で、マクロ変数の値を設定します。この場合、次の正しいプログラムのように、CALL SYMPUT ステートメントを使用する必要があります。

```
%let sr_cit = no;
data senior;
set census;
if age > 65 then
```

```
do;
call symput ("sr_cit","yes");
output;
end;
run;
```

このプログラムをサブミットすると、65 よりも大きい AGE の値を持つオブザベーションが検出された場合にのみ、SR_CIT の値は **yes** に設定されます。なお、SR_CIT の値は **no** に初期設定されています。通常、マクロ変数を初期化しておくのは良いことです。

DATA ステップのコンパイル時のマクロ置換の問題を解決する

前の例では、DATA ステップ内のマクロ変数に条件付きで値を割り当てるには、CALL SYMPUT を使用する必要があるということを学習しました。そのため、次のプログラムをサブミットします。

```
%let sr_age = 0;
data senior;
set census;
if age > 65 then
do;
call symput("sr_age",age);
put "This data set contains data about a person";
put "who is &sr_age years old."; /* ERROR */
end;
run;
```

AGE の値が 67 である場合、次のようなログメッセージが表示されることが期待できます。

```
This data set contains data about a person
who is 67 years old.
```

しかし実際は、AGE の値に関係なく、次のメッセージがログに出力されます。

```
This data set contains data about a person
who is 0 years old.
```

DATA ステップがコンパイルされるときに、&SR_AGE がマクロ機能に渡されて置換されます。その置換結果が返されて、DATA ステップが実行されます。目的の結果を得るには、代わりに、修正された次のプログラムをサブミットします。

```
%let sr_age = 0;
data senior;
set census;
if age > 65 then
do;
call symput("sr_age",age);
stop;
end;
run;

data _null_;
put "This data set contains data about a person";
put "who is &sr_age years old.";
run;
```

注: PUT のようなステートメントでは、二重引用符を使用します。これは、一重引用符で囲むとマクロ変数が置換されないためです。

マクロ変数が作成されたのと同じステップ内で、そのマクロ変数を誤って参照している別の例を次に示します。

```
data _null_;
retain total 0;
set mydata end=final;
total=total+price;
call symput("macvar",put(total,dollar14.2));
if final then put "*** total=&macvar ***"; /* ERROR */
run;
```

これらのステートメントをサブミットすると、次の行が SAS ログに書き込まれます。

```
WARNING: Apparent symbolic reference MACVAR not resolved.

*** total=&macvar ***
```

この DATA ステップがトークン化され、コンパイル化されるときに、ワードスキャナが**&**を検出してマクロプロセッサを起動します。起動されたマクロプロセッサは、シンボルテーブル内の MACVAR エントリを検索します。そのようなエントリが存在しないため、マクロプロセッサは警告メッセージを生成します。入力スタックにトークンが残っているため、それらのトークンが DATA ステップコンパイラに転送されます。DATA ステップの実行中に、CALL SYMPUT ステートメントによってマクロ変数 MACVAR が作成され、それに値が割り当てられます。ところが、PUT ステートメントには、**&macvar** というテキストが生成されます。これは、マクロのコンパイル中に、このテキストがすでに処理されたためです。これらのステートメントを再びサブミットして、マクロが正常に動作したように見えたとしても、その MACVAR の値は、前回の DATA ステップの実行時に設定された値を反映しています。この値は、誤解を招く場合があります。

通常、**%**と**&**は、他の SAS コードのコンパイル時に、直ちに実行または置換を引き起こします。このことを覚えておいてください。

CALL SYMPUT を使用してマクロ変数を作成するその他の例および説明については、“CALL SYMPUT ルーチンを使用したスコープの特殊なケース” (65 ページ)を参照してください。

自動呼び出し機能の問題を解決する

プロダクション(デバッグ済み)マクロを保存して使用する場合、自動呼び出し機能は有効な手段になります。自動呼び出しマクロの呼び出しでエラーが発生した場合、その原因は次の 2 つのうちのいずれかです。

- 間違った自動呼び出しライブラリの指定
- 無効な自動呼び出しマクロ定義

エラーが自動呼び出しライブラリ指定にある場合、MERROR オプションが設定されていれば、SAS によって、次の警告メッセージのいずれか、またはすべてが生成されることがあります。

```
WARNING: No logical assign for filename
FILENAME.
WARNING: Source level autocall is not found or cannot be opened.
Autocall has been suspended and OPTION NOMAUTOSOURCE has
been set. To use the autocall facility again, set OPTION
MAUTOSOURCE.
WARNING: Apparent invocation of macro
MACRO-NAME not resolved.
```

エラーが自動呼び出しマクロ定義にある場合、SAS によって次のようなメッセージが生成されます。

```
NOTE: Line generated by the invoked macro
"MACRO-NAME".
```

自動呼び出しライブラリ指定の修正

自動呼び出しライブラリの指定によってエラーが発生した場合、その原因は、マクロプロセッサが、SASAUTOS システムオプションで指定された 1 つまたは複数のライブラリから、自動呼び出しマクロ定義を含むメンバを検出できなかったためです。

このエラーを修正するには、次の手順に従います。

1. 未置換のマクロ呼び出しによって無効な SAS ステートメントが作成された場合、1 つのセミコロンをサブミットして、無効なステートメントを終了させます。その後、SAS は、以降のステートメントを正しく認識できます。
2. OPTIONS プロシジャの出力を表示するか、SAS ウィンドウ環境で **OPTIONS** ウィンドウを参照することによって、SASAUTOS システムオプションの値を調べます(あるいは、SAS 構成ファイルまたは SAS AUTOEXEC ファイルを編集します)。各ファイル参照名またはディレクトリ名を確認します。エラーが見つかった場合、新しい OPTIONS ステートメントをサブミットするか、**OPTIONS** ウィンドウで SASAUTOS の設定を変更します。
3. MAUTOSOURCE システムオプションを確認します。SAS は、ライブラリを 1 つも開けなかった場合、NOMAUTOSOURCE オプションを設定します。NOMAUTOSOURCE が存在する場合、新しい OPTIONS ステートメントまたは **OPTIONS** ウィンドウを使用して、MAUTOSOURCE をリセットします。
4. ライブラリ指定が正しい場合、各ディレクトリの内容を調べて、自動呼び出しライブラリメンバが存在し、それに同じ名前のマクロ定義が含まれていることを確認します。メンバが欠損している場合は、それを追加します。
5. 新しい OPTIONS ステートメントまたは **OPTIONS** ウィンドウを使用して、MRECALL オプションを設定します。デフォルトでは、マクロプロセッサは、未定義のマクロを一度だけ検索します。このオプションを設定すると、マクロプロセッサは、指定された自動呼び出しライブラリを再検索します。
6. 自動呼び出しマクロのソースを含む自動呼び出しマクロを呼び出し、それをサブミットします。
7. NOMRECALL オプションをリセットします。

注: 一部のホスト環境では、環境変数名またはシステムレベルの論理名が SASAUTOS ライブラリに割り当てられています。SASAUTOS ライブラリ指定がホスト環境でどのように扱われているかについてには、SAS ドキュメントで詳細を確認してください。

自動呼び出しマクロ定義エラーの修正

自動呼び出し機能によって自動呼び出しライブラリメンバが検出されると、マクロプロセッサは、そのライブラリメンバに含まれるすべてのマクロをコンパイルします。コンパイルされたマクロは、コンパイル済みマクロが含まれるカタログに保存されます。SAS セッションの他の部分では、これらのマクロのいずれかを呼び出すことによって、WORK ライブラリからコンパイル済みマクロを取り出します。いかなる状況であっても、自動呼び出し機能は、同じ名前のコンパイル済みマクロがすでに自動呼び出しライブラリメンバに存在する場合、そのメンバを使用します。そのため、自動呼び出しマ

クロを呼び出したときに、その自動呼び出しマクロ定義にエラーがあることがわかった場合、今後の使用のために、自動呼び出しライブラリメンバを修正する必要があります。プログラム内またはセッション内で、修正したバージョンを直接コンパイルします。

ウィンドウ環境で自動呼び出しマクロ定義を修正するには、次の手順を実行します。

1. INCLUDE コマンドを使用して、自動呼び出しライブラリメンバを **SAS プログラムエディタ**ウィンドウに表示します。マクロがカタログの SOURCE エントリに保存されている場合、COPY コマンドを使用して、そのプログラムを**プログラムエディタ**ウィンドウに表示します。
2. エラーを修正します。
3. 修正したマクロのコピーを、そのマクロが外部ファイル内にある場合は FILE コマンドを使用し、カタログエントリ内にある場合は SAVE コマンドを使用して、自動呼び出しライブラリに保存します。
4. **プログラムエディタ**ウィンドウから、マクロ定義をサブミットします。

その後、修正したバージョンがマクロプロセッサによってコンパイルされ、エラーのあるコンパイル済みマクロと置き換えられます。これで、修正したコンパイル済みマクロを次の呼び出しで実行する準備が整いました。

自動呼び出しマクロ定義を対話的なラインモードセッションで修正するには、次の手順を実行します。

1. 自動呼び出しマクロのソースを、テキストエディタを使用して編集します。
2. エラーを修正します。
3. %INCLUDE ステートメントを使用して、修正したライブラリメンバを SAS セッションで表示します。

その後、修正したバージョンがマクロプロセッサによってコンパイルされ、エラーのあるコンパイル済みマクロと置き換えられます。これで、修正したコンパイル済みマクロを次の呼び出しで実行する準備が整いました。

自動呼び出しファイル名とマクロ名

マクロを自動呼び出しマクロとして使用する場合、そのマクロを、それと同じ名前のファイルに保存する必要があります。また、ファイル拡張子も必要です **.sas**(オペレーティングシステムがファイル拡張子を使用している場合)。自動呼び出し機能で問題が発生した場合は、マクロ名とファイル名が一致していることを必ず確認し、必要に応じてファイルの拡張子が正しいことを確認してください。

コンパイル済みマクロに関する情報の表示

コンパイル済みマクロを含むカタログ内のエントリの一覧を表示するには、**カタログ**ウィンドウまたは CATALOG プロシジャを使用します。次の PROC ステップは、ライブラリ参照名 MYSASLIB で指定した SAS ライブラリ内のマクロカタログの内容を表示します。

```
libname mysaslib
'SAS-data-library';
proc catalog catalog=mysaslib.sasmacr;
contents;
run;
quit;
```

PROC CATALOG を使用して、カタログ内の SOURCE エントリに保存された自動呼び出しライブラリのマクロに関する情報を表示することもできます。コンパイル済みマクロをコピー、削除、または名前変更するために、PROC CATALOG または**エクスプローラ**ウィンドウを使用することはできません。

MCOMPILENOTE システムオプションを使用して、マクロのコンパイルが完了した際にログにメモを出力することができます。詳細については、“MCOMPILENOTE システムオプション” (341 ページ)を参照してください。

SAS 6.11 以降では、PROC SQL を使用して、すべてのコンパイル済みマクロに関する情報を取得できます。たとえば、次のステートメントをサブミットすると、次に示すような出力が生成されます。

```
proc sql;
select * from dictionary.catalogs
where memname in ('SASMACR');
```

アウトプット 10.1 *コンパイル済みマクロを表示するための PROC SQL プログラムの出力*

```
Library Member Member Object Object Date Object
Name Name Type Name Type Object Description Modified Alias
-------------------------------------------------------------------------------------------
WORK SASMACR CATALOG FINDAUTO MACRO 05/28/96

SASDATA SASMACR CATALOG CLAUSE MACRO Count words in clause 05/24/96

SASDATA SASMACR CATALOG CMPRES MACRO CMPRES autocall macro 05/24/96

SASDATA SASMACR CATALOG DATATYP MACRO DATATYP autocall macro 05/24/96

SASDATA SASMACR CATALOG LEFT MACRO LEFT autocall macro 05/24/96
```

コンパイル済みマクロを呼び出すときに、それらの情報を表示するには、SAS システムオプションの MLOGIC、MPRINT、および SYMBOLGEN を使用します。SAS システムオプション MLOGIC を指定すると、マクロの実行中に表示される通常の情報と共に、ライブラリ参照名、およびコンパイル済みマクロがコンパイルされた日付がログに書き込まれます。

式の評価の問題を解決する

次のマクロステートメントは、%EVAL 関数を使用します。

表 10.3 *%EVAL 関数を使用するマクロステートメント*

%DO	%IF-%THEN	%SCAN
%DO %UNTIL	%QSCAN	%SYSEVALF
%DO %WHILE	%QSUBSTR	%SUBSTR

また、%EVAL 関数を使用して、式の評価を指定できます。

式の評価中における最もよくあるエラーは、数値オペランドが必要とされる場所に文字オペランドが存在するか、トークンが数値演算子なのか文字値なのかがあいまいな場

合に発生します。“マクロ式” (73 ページ) で、これらおよびその他のマクロ式に関するエラーについて説明しています。

特殊文字やキーワードが文字列に現れる場合、頻繁にエラーが発生します。次のプログラムについて考えます。

```
%macro conjunct(word= );
%if &word = and or &word = but or &word = or %then /* ERROR */
%do %put *** &word is a conjunction. ***;

%else
%do %put *** &word is not a conjunction. ***;
%mend conjunct;
```

この%IF ステートメントでは、テストされる WORD の値があいまいです。これらの値は、数値演算子 AND および OR としても解釈できます。そのため、次のエラーメッセージがログに生成されます。

```
ERROR: A character operand was found in the %EVAL function or %IF
condition where a numeric operand is required. The condition
was:word = and or &word = but or &word = or
ERROR: The macro will stop executing.
```

この問題を修正するには、次の修正済みプログラムに示すように、クォーティング関数%BQUOTE および%STR を使用します。

```
%macro conjunct(word= );
%if %bquote(&word) = %str(and) or %bquote(&word) = but or
%bquote(&word) = %str(or) %then
%do %put *** &word is a conjunction. ***;

%else
%do %put *** &word is not a conjunction. ***;
%mend conjunct;
```

この修正済みプログラムでは、%BQUOTE 関数によってマクロ変数の置換結果をクォーティングしています(一致しない引用符などの半端な値を含むワードを渡す場合)。%STR 関数は、比較する値 AND および OR を、コンパイル時にクォーティングします。そのため、これらの値はあいまいではありません。値 BUT はあいまいではない(SAS 言語にもマクロ言語にも含まれない)ため、これに対して%STR を使用する必要はありません。マクロクォーティング関数の使用の詳細については、“マクロクォーティング” (82 ページ)を参照してください。

デバッグの方法

システムオプションを使用して、問題をトラッキングする

SAS システムオプション MLOGIC、MLOGICNEST、MPRINT、MPRINTNEST、および SYMBOLGEN は、マクロコードおよびマクロが生成する SAS コードをトラッキングするのに役立ちます。これらのオプションが生成するメッセージは、それを生成したオプションの名前が先頭に付加されて、SAS ログに表示されます。

注: マクロ機能を使用する場合は、マクロのオプション MACRO、MERROR、および SERROR を使用します。SOURCE は、マクロ機能を使用する場合に役立つシステムオプションです。%INCLUDE を使用する場合、SOURCE2 システムオプションを使用することも役に立ちます。

以降のセクションでは、各システムオプションを別々に説明しますが、当然、それらを組み合わせて使用することができます。ただし、これらのオプションによって大量の出力が生成される場合があり、多すぎる情報は、少なすぎる情報と同様に混乱を招くことがあります。そのため、必要と判断したオプションのみを使用し、デバッグが完了したら無効にしてください。

MLOGIC を使用した実行フローのトレース

MLOGIC システムオプションを指定すると、パラメータの置換などのマクロの実行フロー、変数のスコープ(グローバルまたはローカル)、評価されたマクロ式の状態、ループの反復回数、各マクロの実行開始と終了がトレースされます。単純な構文エラーとは対照的に、プログラムロジックにバグがあると思われる場合、MLOGIC オプションを使用します。

注: MLOGIC によって大量の出力が生成される場合があるため、必要なときにのみこのオプションを使用し、デバッグが終了したら無効にしてください。

次の例では、マクロ FIRST によってマクロ SECOND を呼び出し、式を評価しています。

```
%macro second(param);
%let a = %eval(&param); &a
%mend second;

%macro first(exp);
%if (%second(&exp) ge 0) %then
%put **** result >= 0 ****;
%else
%put **** result < 0 ****;
%mend first;

options mlogic;
%first(1+2)
```

オプション MLOGIC を指定してこの例をサブミットすると、各マクロの実行開始時刻、渡したパラメータの値、および式の評価結果が表示されます。

```
MLOGIC(FIRST): Beginning execution.
MLOGIC(FIRST): Parameter EXP has value 1+2
MLOGIC(SECOND): Beginning execution.
MLOGIC(SECOND): Parameter PARAM has value 1+2
MLOGIC(SECOND): %LET (variable name is A)
MLOGIC(SECOND): Ending execution.
MLOGIC(FIRST): %IF condition (%second(&exp) ge 0) is TRUE
MLOGIC(FIRST): %PUT **** result >= 0 ****
MLOGIC(FIRST): Ending execution.
```

MLOGICNEST によって生成されるネスト情報

MLOGICNEST を指定すると、マクロのネスト情報を、MLOGIC の出力として SAS ログに書き込むことができます。MLOGICNEST を設定しても、MLOGIC を設定したことにはなりません。ネスト情報を含む出力を SAS ログに表示するには、MLOGIC および MLOGICNEST の両システムオプションを設定する必要があります。

詳細と例については、“MLOGICNEST システムオプション” (353 ページ)を参照してください。

MPRINT を使用した生成済み SAS ステートメントの検証

MPRINT システムオプションを指定すると、マクロが生成した各 SAS ステートメントが SAS ログに書き込まれます。コードが期待どおり生成されず、コードにバグがあると疑われる場合、MPRINT オプションを使用します。

たとえば、次のプログラムでは単純な DATA ステップを生成しています。

```
%macro second(param);
%let a = %eval(&param); &a
%mend second;

%macro first(exp);
data _null_;
var=%second(&exp);
put var=;
run;
%mend first;

options mprint;
%first(1+2)
```

オプション MPRINT を指定してこれらのステートメントをサブミットすると、次の行が SAS ログに書き込まれます。

```
MPRINT(FIRST): DATA _NULL_;
MPRINT(FIRST): VAR=
MPRINT(SECOND): 3
MPRINT(FIRST): ;
MPRINT(FIRST): PUT VAR=;
MPRINT(FIRST): RUN;

VAR=3
```

MPRINT オプションを使用することで、生成されたテキストを表示し、それを生成したマクロを特定できます。

MPRINTNEST によって生成されるネスト情報

MPRINTNEST を指定すると、マクロのネスト情報を、MPRINT の出力として SAS ログに書き込むことができます。この値は、外部ファイルに送信される MPRINT の出力には効果がありません。詳細については、“MFILE システムオプション” (348 ページ)を参照してください。

MPRINTNEST を設定しても、MPRINT を設定したことにはなりません。ネスト情報を含む出力を SAS ログに表示するには、MPRINT と MPRINTNEST の両システムオプションを設定する必要があります。

詳細および例については、“MPRINTNEST システムオプション” (357 ページ)を参照してください。

外部ファイルへの MPRINT 出力の保存

マクロの実行中にマクロ機能によって生成されたテキストを、外部ファイルに保存できます。最近の SAS セッションで生成されたテキストをテストするときに、マクロの実行

中に生成されたステートメントをファイルに出力しておくと、マクロのデバッグに役立ちます。

この機能を使用するには、MFILE システムオプションと MPRINT システムオプションの両方を有効に設定します。また、MPRINT を、マクロ機能によって生成された出力を含むファイルへのファイル参照名として割り当てるには、次のように設定します。

```
options mprint mfile;
filename mprint 'external-file';
```

MPRINT システムオプションによって作成された外部ファイルは、SAS セッションが終了するまで開かれたままになります。マクロ機能によって生成された MPRINT のテキストは、SAS セッション中はログに書き込まれ、セッションが終了すると外部ファイルに書き込まれます。このテキストは、マクロの実行中に生成され、マクロ変数参照とマクロ式が置換されたプログラムステートメントから成ります。外部ファイルには、マクロによって生成されたステートメントのみが保存されます。マクロの外部のどのステートメントも、外部ファイルには書き込まれません。各ステートメントは、ワード間のスペースを 1 つ開けて、新しい行に書き込まれます。テキストは、ログに表示されている **MPRINT(*macroname*:**という接頭語を付けないで外部ファイルに保存されます。

MPRINT がファイル参照名として割り当てられていないか、外部ファイルアクセスできない場合、警告メッセージがログに書き込まれ、MFILE が無効になります。この機能を再び使用するには、もう一度 MFILE を指定する必要があります。

デフォルトでは、MPRINT オプションと MFILE オプションは無効になっています。

次の例では、MPRINT オプションと MFILE オプションを使用して、生成されたテキストを TEMPOUT という名前の外部ファイルに保存しています。

```
options mprint mfile;
filename mprint 'TEMPOUT';

%macro temp;
data one;
%do i=1 %to 3;
x&i=&i;
%end;
run;
%mend temp;

%temp
```

マクロ機能によって次の行が SAS ログに書き込まれ、TEMPOUT という名前の外部ファイルが作成されます。

```
MPRINT(TEMP): DATA ONE;
NOTE: The macro generated output from MPRINT will also be written
to external file '/u/local/abcdef/TEMPOUT' while OPTIONS
MPRINT and MFILE are set.
MPRINT(TEMP): X1=1;
MPRINT(TEMP): X2=2;
MPRINT(TEMP): X3=3;
MPRINT(TEMP): RUN;
```

SAS セッションが終了したときの、ファイル TEMPOUT の内容は次のとおりです。

```
DATA ONE;
X1=1;
X2=2;
X3=3;
RUN;
```

注: MPRINT を使用した外部ファイルへのコードの書き込みは、デバッグ専用ツールです。デバッグ以外の目的で、MPRINT を使用して SAS コードファイルを作成しないでください。

SYMBOLGEN を使用したマクロ変数の置換の検証

SYMBOLGEN システムオプションを指定すると、各マクロ変数の置換結果が、メッセージとして SAS ログに書き込まれます。特殊文字によってマクロ変数が意図したとおりに置換されないといった、クォーティングの問題を特定する場合、特にこのオプションが役に立ちます。

例として、次のステートメントをサブミットする場合を考えます。

```
options symbolgen;

%let a1=dog;
%let b2=cat;
%let b=1;
%let c=2;
%let d=a;
%let e=b;
%put **** &&&d&b ****;
%put **** &&&e&c ****;
```

SYMBOLGEN オプションによって、次の行が SAS ログに書き込みまれます。

```
SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable D resolves to a
SYMBOLGEN: Macro variable B resolves to 1
SYMBOLGEN: Macro variable A1 resolves to dog
**** dog ****

SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable E resolves to b
SYMBOLGEN: Macro variable C resolves to 2
SYMBOLGEN: Macro variable B2 resolves to cat
**** cat ****
```

SYMBOLGEN オプションによって得られたログを読むことは、プログラムステートメントを調べて間接的に置換をトレースするよりも簡単です。SYMBOLGEN オプションを指定したことで、マクロプロセッサによるマクロ変数の置換の各ステップがトレースされていることに注目してください。置換が完了すると、%PUT ステートメントによって SAS ログに値が書き込まれます。

マクロクォーティング関数によってマスクされたマクロ変数の値を、SYMBOLGEN を使用してトレースしたときに、出力するためにクォーティングが解除されたことを示す追加メッセージが表示される場合があります。たとえば、SYMBOLGEN を有効に設定して次のステートメントをサブミットしたとします。

```
%let nickname = %str(My name%'s O%'Malley, but I%'m called Bruce);
%put *** &nickname ***;
```

これらのステートメントの実行が完了すると、次のメッセージが SAS ログに出力されます。

```
SYMBOLGEN: Macro variable NICKNAME resolves to
My name's O'Malley, but I'm called Bruce
SYMBOLGEN: Some characters in the above value which were
subject to macro quoting have been
```

```
unquoted for printing.
*** My name's O'Malley, but I'm called Bruce ***
```

このクォーティング解除メッセージは、無視することができます。

%PUT ステートメントを使用して、問題をトラッキングする

マクロの開発中およびデバッグ中に、SYMBOLGEN システムオプションを使用してマクロ変数の値を SAS ログに書き込むことに加えて、%PUT ステートメントを使用すると効果的です。マクロが完成したら、それらの%PUT ステートメントを削除するか、コメント化することができます。次の表に、%PUT ステートメントがデバッグで役立つ場合の例をいくつか示します。

表 10.4 マクロのデバッグ時に役立つ%PUT ステートメントの例

状況	例
マクロ変数の値を表示する	**%PUT ****&=*variable-name*****;**
変数の値の先頭または末尾の空白を確認する	**%PUT ***&*variable-name****;**
ループの実行中に、二重アンパサンドの置換を確認する	**%PUT ***variable-name&i = &&*variable-name****;**
条件の評価を確認する	**%PUT ***This condition was met.****;**

ご存知のように、マクロ変数はシンボルテーブルに格納されます。シンボルテーブルには、グローバルマクロ変数が格納されるグローバルシンボルテーブルと、ローカルマクロ変数が格納されるローカルシンボルテーブルがあります。デバッグ中に、時々これらのテーブルを出力してマクロ変数のグループのスコープと値を調べると、役立つ場合があります。これを実行するには、次のオプションのいずれかを指定して%PUT ステートメントを使用します。

ALL
: スコープに関係なく、現在定義されているすべてのマクロ変数を表示します。ユーザー定義のグローバル変数とローカル変数、および自動マクロ変数が含まれます。

AUTOMATIC
: すべての自動マクロ変数を表示します。スコープは、AUTOMATIC として表示されます。自動マクロ変数は、SYSPBUFF を除き、すべてグローバルです。

GLOBAL
: マクロプロセッサによって作成されていない、すべてのグローバルマクロ変数を表示します。スコープは、GLOBAL として表示されます。自動マクロ変数は表示されません。

LOCAL
: 現在実行中のマクロ内で定義された、ユーザー定義のローカルマクロ変数にを表示します。スコープは、そのマクロ変数が定義されているマクロの名前で表示されます。

USER
スコープに関係なく、すべてのユーザー定義マクロ変数を表示します。グローバルマクロ変数の場合、スコープは GLOBAL になります。ローカルマクロ変数の場合、スコープはマクロの名前になります。

次の例では、引数_USER_を指定して%PUT ステートメントを使用し、マクロ TOTINV で使用できるグローバル変数とローカル変数を調べています。%PUT ステートメントによって値がログに書き込まれるタイミングを、ユーザー定義マクロ変数 TRACE を使用して制御していることに注目してください。

```
%macro totinv(var);
%global macvar;
data inv;
retain total 0;
set sasuser.houses end=final;
total=total+&var;
if final then call symput("macvar",put(total,dollar14.2));
run;

%if &trace = ON %then
%do;
%put *** Tracing macro scopes. ***;
%put _USER_;
%end;
%mend totinv;

%let trace=ON;
%totinv(price)
%put *** TOTAL=&macvar ***;
```

これらのステートメントをサブミットすると、SAS ログには、マクロ TOTINV 内の 1 番目の%PUT ステートメントによって、トレースが有効になったことを示すメッセージが書き込まれ、次にすべてのユーザー定義マクロ変数のスコープと値が書き込まれます。

```
*** Tracing macro scopes. ***
TOTINV VAR price
GLOBAL TRACE ON
GLOBAL MACVAR $1,240,800.00
*** TOTAL= $1,240,800.00 ***
```

マクロ変数のスコープの詳細な説明については、“マクロ変数のスコープ” (45 ページ)を参照してください。

11 章

効率的なマクロとポータブルマクロの作成

効率的なマクロとポータブルマクロの作成

マクロ機能は、SAS コードの開発をさらに効率的にする、強力なツールです。ただし、マクロは、ユーザーが作成したとおりの効率性しか発揮しません。効率的なマクロを作成するには、いくつかの手法と考慮事項があります。複数のホスト環境で使用できるマクロを作成して、マクロ機能をさらに強化しようとする場合は、ポータブルマクロを作成するための追加の考慮事項があります。

全体的な視野に立った効率の維持

効率性は、理解しにくい問題です。効率性を定量化することは難しく、定義することはさらに困難です。あるアプリケーションに対して効果があることが、別のアプリケーションに対して効果があるとは限らず、あるホスト環境で有効なことが、別のホスト環境でも有効とは限りません。ただし、一般的に留意しておくべきことはあります。

通常、効率性の問題は、CPU サイクル、経過時間、入出力回数、メモリ使用量、ディスク使用量などの観点から議論されます。このセクションでは、これらの項目のベンチマークは、すべての変数に関連するため提供されません。一度しか実行されないプログラムは、何度も実行されるプログラムとは別の調整を必要とします。メインフレーム上で実行されるアプリケーションには、デスクトップ PC 上で開発されるアプリケーションとは異なるハードウェアパラメータがあります。使用している環境の視点から効率性を維持する必要があります。

節約したいリソースの種類に応じて、効率性を維持するためのさまざまな方法があります。たとえば、入出力回数よりも CPU サイクルの方が重要であったり、メモリは十分にあるがディスク容量が不足しているなどの状況があります。プログラムの調整方法を決める前に、このような状況を調べておくことをお勧めします。

SAS マクロ機能が最も影響を与える効率性の領域は、人間の作業効率、つまりプログラムの開発と保守の両方に必要な作業時間です。 自動呼び出しマクロは、その自動呼び出し機能によってコードを再利用できるため、この領域では特に重要になります。タスクを実行するマクロを一度開発したら、それを保存して、それを開発したアプリケーション内だけでなく、今後開発するアプリケーション内でも、追加作業を行わずに使用できます。再利用可能で即座に呼び出すことのできるマクロのライブラリは、すべてのアプリケーション開発チームに恩恵をもたらします。

コンパイル済みマクロ機能(“マクロの保存および再利用” (115 ページ)で説明されています)を使用すると、さまざまな SAS ジョブや SAS セッションの実行中にコンパイル済みマクロにアクセスできるため、実行時間が減る可能性があります。ただし、この機能は、プロダクションアプリケーションに対してのみ効率的なツールであり、開発中のアプリケーションに対しては効率的ではありません。そのため、選択する効率化手法は、使用するハードウェアや担当者の状況によって変わるだけでなく、アプリケーション開発プロセスの到達段階によっても変わります。

マクロコードを SAS アプリケーションに組み込んだからといって、自動的にアプリケーションが効率的になるわけではないということも、覚えておく必要があります。SAS アプリケーションを設計する際は、基本的な SAS コードを作成することに集中します。それらのコードは、マクロを使用することでさらに効率化されます。効率的な SAS コードについては、*SAS Programming Tips: A Guide to Efficient SAS Processing* など、多くの情報源があります。

効率的なマクロの作成

マクロの有効利用

マクロを使用して定数テキストのみを生成するアプリケーションは、効率的ではありません。通常、そのような状況では、%INCLUDE ステートメントの使用を検討してください。%INCLUDE ステートメントは、最初にコードをコンパイルする必要がないため(即座に実行されるため)、マクロを使用するよりも効率的です。そのコードを一度だけ実行する場合、特に効果的です。同じコードを繰り返し使用する場合は、マクロを使用し

たほうが効率的です。これは、マクロが、SAS ジョブの実行中に何度呼び出されたとしても、一度しかコンパイルされないためです。

ただし、%INCLUDE を使用する場合、物理ファイルが保存されている場所を正確に把握し、その名前をプログラム内で指定する必要があります。自動呼び出し機能で覚えておく必要があるのは(パス名ではなく)マクロ名のみであるため、作業効率の点で得られるメリットが、マクロをコンパイルしないことで得られる時間的なメリットを上回る可能性があります。また、マクロには、パラメータ、条件付きセクション、ループなどの他のプログラミング機能に加え、マクロ変数の置換を SAS ログに表示する機能も用意されています。

これらを考慮して、必要な場合にのみマクロを使用するようにしてください。また、効率性に影響を与えるさまざま要因とメリットとの間のバランス(コードの使用回数や CPU 時間に対する使い勝手の良さなど)を考えて、使用するアプリケーションに最も適した解決策を見つける必要があります。

ユーザー名スタイルマクロ

マクロの呼び出しには、ネームスタイル、コマンドスタイル、ステートメントスタイルの 3 種類があります。これら 3 つのうち、ネームスタイルが最も効率的です。これは、ネームスタイルマクロが必ず%で始まり、ワードスキャナに対して、マクロプロセッサにトークンを渡すよう即座に指示できるためです。他の 2 種類の場合、ワードスキャナは、マクロプロセッサにトークンを送信するべきかどうかを即座に知ることはできません。そのため、トークンを送信するかどうかをワードスキャナが判断している間に時間が経過します。

ネストされたマクロ定義の回避

他のマクロ内でマクロ定義ネストすることは、通常は必要なく、効率的でもありません。ネストされたマクロ定義を含むマクロを呼び出すと、マクロプロセッサによって、ネストされたマクロ定義がテキストとして生成され、入力スタックに配置されます。次に、そのマクロ定義は、ワードスキャナによってスキャンされ、マクロプロセッサによってコンパイルされます。あるマクロ定義をネストした場合、それを変更しなくても、外側のマクロが実行されるたびに、同じネストされたマクロがマクロプロセッサによってコンパイルされます。

原則としてマクロは、別々に定義する必要があります。マクロのスコープをネストする場合は、マクロ定義ではなく、単にマクロ呼び出しをネストします。

たとえば、次の例では、マクロ TITLE のネストされたマクロ定義が、マクロ STATS1 に含まれています。

```
/* Nesting a Macro Definition--INEFFICIENT */
%macro stats1(product,year);
%macro title;
title "Statistics for &product in &year";
%if &year>1929 and &year<1935 %then
%do;
title2 "Some Data Might Be Missing";
%end;
%mend title;

proc means data=products;
where product="&product" and year=&year;
%title
run;
%mend stats1;
```

```
%stats1(steel,2002)
%stats1(beef,2000)
%stats1(fiberglass,2001)
```

マクロ STATS1 が呼び出されるたび、マクロプロセッサはマクロ TITLE の定義をテキストとして生成し、マクロ定義を認識し、マクロ TITLE をコンパイルします。この場合、STATS1 は 3 回呼び出されます。つまり、TITLE マクロが 3 回コンパイルされます。このマクロのステートメントは数行しかないため、コンパイルに数マイクロ秒しかかかりませんが、ステートメントが数百行になるような大規模なマクロの場合は、かなりの時間がかかります。

TITLE は、STATS1 の定義内で呼び出されているため、PRODUCT と YEAR の値を使用できます。したがって、これらの値を TITLE のスコープで使用できるようにするために、TITLE の定義をネストする必要はありません。TITLE ステートメントの定義に含まれる値が、STATS1 の実行中に変化する値に依存していないという理由からも、TITLE の定義のネストは不要です。TITLE ステートメントの定義がそのような値に依存している場合でも、定義をネストするのではなく、グローバルマクロ変数を使用することで、変更を反映することができます。

別々に定義されたマクロを、次のプログラムに示します。

```
/* Separating Macro Definitions--EFFICIENT */
%macro stats2(product,year);
proc means data=products;
where product="&product" and year=&year;
%title
run;
%mend stats2;

%macro title;
title "Statistics for &product in &year";
%if &year>1929 and &year<1935 %then
%do;
title2 "Some Data Might Be Missing";
%end;
%mend title;

%stats2(cotton,1999)
%stats2(brick,2002)
%stats2(lamb,2001)
```

ここでは、マクロ TITLE の定義がマクロ STATS2 の定義の外にあるため、TITLE は、STATS2 が 3 回呼び出されても1回しかコンパイルされません。TITLE の呼び出しが STATS2 の定義に含まれているため、この場合も TITLE は、PRODUCT と YEAR の値を使用できます。

注: マクロを別々に定義するもう 1 つの理由は、各マクロを別々のファイルに保存して保守しやすくするためです。

マクロ変数への関数結果の割り当て

関数を評価するよりも、変数の参照を置換したほうが効率的です。したがって、頻繁に使用する関数の結果は、マクロ変数に割り当てます。

たとえば、次のマクロは、%DO %WHILE ステートメントのすべての反復でマクロ変数 THETEXT の長さを評価する必要があるため、非効率です。

```
/* INEFFICIENT MACRO */
```

```
%macro test(thetext);
%let x=1;
%do %while (&x > %length(&thetext));
.
.
.
%end;
%mend test;

%test(Four Score and Seven Years Ago)
```

より効率的な方法は、THETEXT の長さを一度評価して、その値を別のマクロ変数に割り当てることです。その後、次のプログラムに示すように、そのマクロ変数を%DO %WHILE ステートメントで使用します。

```
/* MORE EFFICIENT MACRO */
%macro test2(thetext);
%let x=1;
%let length=%length(&thetext);
%do %while (&x > &length);
.
.
.
%end;
%mend test2;

%test(Four Score and Seven Years Ago)
```

別の例として、%SUBSTR 関数を使用して、SYSDATE の値から年を取り出すとします。コード内で繰り返し%SUBSTR を使用する代わりに、%SUBSTR(&SYSDATE, 6) の値をマクロ変数に割り当て、年が必要なときには、その変数を使用します。

システムオプションの無効化(適切な場合)

MPRINT や MLOGIC などのデバッグ用のシステムオプションは、場合によっては非常に効果的ですが、この種のシステムオプションを有効に設定して、プロダクション(デバッグ済み)マクロを実行することは効率的ではありません。プロダクションマクロの場合は、NOMLOGIC、NOMPRINT、NOMRECALL、および NOSYMBOLGEN を設定してジョブを実行してください。

ジョブにエラーがない場合にも、これらのオプションを有効にしてジョブを実行すると、オプションが必要とするオーバーヘッドが発生します。これらのオプションを無効にすることで、プログラムの実行がより効率的になります。

注: MPRINT と NOMPRINT の使い分けを判断する別の方法は、このオプションの設定を SOURCE オプションの設定に合わせることです。つまり、プログラムで SOURCE オプションを使用する場合は、MPRINT も使用する必要があります。同様に、プログラムで NOSOURCE を使用する場合は、NOMPRINT を設定して実行します。

注: 自動呼び出しマクロを使用しない場合は、NOMAUTOSOURCE システムオプションを使用してください。コンパイル済みマクロを使用しない場合は、NOMSTORED システムオプションを使用してください。

コンパイル済みマクロ機能の使用

コンパイル済みマクロ機能は、以前の SAS ジョブまたは SAS セッションでコンパイルされたマクロを、それ以降の SAS ジョブおよび SAS セッションでアクセス可能にすることで、実行時間を削減します。つまり、これらのマクロを再コンパイルする必要がありません。コンパイル済みマクロ機能は、プロダクション(デバッグ済み)マクロにのみ使用してください。この機能をマクロアプリケーションを開発するときに使用することは、効率的ではありません。

注意:

ソースコードを保存してください。 コンパイル済みコードからマクロのソースコードを再作成することはできません。そのため、何らかの理由でコンパイル済みコードが破損する場合に備えて、ソースコードのコピーを安全な場所に保管する必要があります。後でマクロを変更しようとする場合にも、ソースのコピーを保持しておく必要があります。

コンパイル済みマクロ機能の詳細については、“マクロの保存および再利用” (115 ページ) を参照してください。

注: コンパイル済みマクロ機能が生成するコンパイル済みコードは、ポータブルではありません。マクロを別のホスト環境に移動する必要がある場合は、ソースコードを新しいホストに移動して再コンパイルし、保存する必要があります。

自動呼び出しマクロの一元保存

自動呼び出し機能を使用する場合、入出力の点から見て最も効率的なのは、すべての自動呼び出しマクロを 1 つのライブラリに保存し、そのライブラリ名を SASAUTOS システムオプション指定の先頭に追加することです。もちろん、任意の数のライブラリに自動呼び出しマクロを保存できますが、マクロを呼び出すたびに、そのマクロが検出されるまで各ライブラリが順次検索されます。開いて検索するライブラリを 1 つに限定することで、マクロの検索時間を減らすことができます。

ただし、多数の自動呼び出しマクロが存在する場合、目的、プロダクションのレベル、サポート担当者などに応じて、それらのマクロを論理的に分割することは理にかなっています。この場合も、入出力が減少することに対する、使い勝手と保守性の悪化について、バランスを考える必要があります。

リスト内で連結されたすべての自動呼び出しライブラリが開かれ、SAS ジョブまたは SAS セッションが実行されている間は開かれたままになります。自動呼び出しマクロを最初に呼び出すと、1 回目で開かれなかったライブラリが再びテストされます。自動呼び出しマクロが使用されるたびに、これが繰り返されます。そのため、SASAUTOS システムオプション指定に無効なパス名が存在すると、極めて非効率的になります。この SAS の一部の無駄な処理に関する警告メッセージは、ライブラリを 1 つも開けない場合を除き、表示されません。

自動呼び出し機能の効率に関して、次の 2 つのヒントがあります。

- マクロ以外のコードを自動呼び出しライブラリファイルに保存しないでください。
- 各自動呼び出しライブラリファイルには、複数のマクロを保存しないでください。

これらのヒントに従わなくてもライブラリファイルは使用されて動作しますが、コードの保守作業が非常に増大し、その結果、効率が下がります。

その他の有用な効率のヒント

試すことのできるその他の効率化手法を次に示します。

- 参照する予定のないマクロ変数があれば、それを null にリセットします。
- 三重アンパサンドを使用して、長い値を持つマクロ変数の追加スキャンを強制的に行います(該当する場合)。詳細については、“長いマクロ変数値のコピーを 1 つだけ保存する” (149 ページ) を参照してください。
- 状況に合わせて、“MSYMTABMAX=システムオプション” (360 ページ) および “MVARSIZE=システムオプション” (361 ページ) の値を調整します。通常、ディスク容量が不足している場合はこれらの値を増やし、メモリが不足している場合は減らします。MSYMTABMAX は、マクロ変数シンボルテーブルを格納できる領域に影響を与え、MVARSIZE は、個々のマクロ変数の値を格納できる領域に影響を与えます。

長いマクロ変数値のコピーを 1 つだけ保存する

マクロ変数には非常に長い値を格納できるため、マクロ変数を格納する方法がプログラムの効率に影響を与える場合があります。3 つのアンパサンドを使用して間接的に参照することで、格納される長い値のコピーの数を減らすことができます。

たとえば、次に示すように、SAS プログラムのセクションを表す長いマクロ変数値がプログラムに含まれているとします。

```
%let pgm=%str(data flights;
set schedule;
totmiles=sum(of miles1-miles20);
proc print;
var flightid totmiles;);
```

次のマクロによって、SAS プログラムを RUN ステートメントで終わるようにします。

```
%macro check(val);
/* first version */ &val
%if %index(&val,%str(run;))=0 %then %str(run;);
%mend check;
```

最初に、マクロ CHECK が、パラメータ VAL (%MACRO ステートメントで定義され、マクロ呼び出しから渡されるマクロ変数)に格納されたプログラムステートメントを生成します。次に%INDEX 関数が、VAL の値に対して文字列 **run** を検索します(%STR 関数を使用することで、セミコロンをテキストとして扱っています)。この文字列が存在しない場合、%INDEX 関数は 0 を返します。%IF 条件が true になり、マクロプロセッサは RUN ステートメントを生成します。

マクロ CHECK を変数 PGM に対して使用するには、次のように、マクロ呼び出しでパラメータ VAL に PGM の値を割り当てます。

```
%check(&pgm)
```

その結果、SAS はこれらのステートメントを次のように解釈します。

```
data flights;
set schedule;
totmiles=sum(of miles1-miles20);

proc print;
var flightid totmiles;
run;
```

マクロ CHECK は、正常に動作します。ただし、マクロプロセッサは、CHECK を実行する際に、PGM の値を VAL の値として割り当てます。そのためマクロプロセッサは、CHECK を実行する間、2 つの長い値(PGM と VAL の値)を格納する必要があります。

プログラムを効率化するには、PGM の値を VAL にコピーしないで使用するように、マクロを記述します。

```
%macro check2(val); /* more efficient macro */ &&&val
%if %index(&&&val,%str(run;))=0 %then %str(run;);
%mend check2;

%check2(pgm)
```

次のマクロ CHECK2 は、マクロ CHECK と同じ結果を生成します。

```
data flights;
set schedule;
totmiles=sum(of miles1-miles20);

proc print;
var flightid totmiles;
run;
```

ただし、マクロ CHEKC2 では、VAL には、PGM の値ではなく、単に **PGM** という名前が割り当てらます。マクロプロセッサは、&&&VAL を&PGM に置換し、次にマクロ変数 PGM に格納されている SAS ステートメントに置換します。そのため、長い値は一度だけ格納されます。

ポータブルマクロの作成

%SYSFUNC でポータブル SAS 言語関数を使用する

コードを 2 つの異なる環境で実行できるようにすると、原則的に、開発作業の価値は 2 倍になります。ただし、ポータブルアプリケーションを開発する場合、前もって計画が必要になります。SAS のホスト固有の機能の詳細については、使用しているホスト環境に関する SAS ドキュメントを参照してください。

%SYSFUNC マクロ関数を使用して SAS 言語関数にアクセスし、ファイルを開いたり削除したりするなどの、ほとんどのホスト固有の操作を実行できます。詳細については、“%SYSFUNC 関数と%QSYSFUNC 関数” (273 ページ)を参照してください。

%SYSFUNC を使用してポータブル SAS 言語関数にアクセスすると、多くのマクロコードを省くことができます。これによってポータブルになるだけでなく、効率も向上します。次の表に、一般的なホスト固有のタスクと、それらのタスクを実行する関数を示します。

表 11.1 *ポータブル SAS 言語関数とその用途*

タスク	SAS 言語関数
ファイル参照名と物理ファイルの割り当ておよび存在の確認	FILENAME、FILEREF、PATHNAME
ファイルを開く	FOPEN、MOPEN
ファイルの存在の確認	FEXIST、FILEEXIST

タスク	SAS 言語関数
ファイルに関する情報の取得	FINFO、FOPTNAME、FOPTNUM
ファイルへのデータの書き込み	FAPPEND、FWRITE
ファイルの読み込み	FPOINT、FREAD、FREWIND、FRLEN
ファイルを閉じる	FCLOSE
ファイルの削除	FDELETE
ディレクトリを開く	DOPEN
ディレクトリに関する情報を返す	DINFO、DNUM、DOPTNAME、DOPTNUM、DREAD
ディレクトリを閉じる	DCLOSE
ホスト固有のオプションの読み込み	GETOPTION
ファイルデータバッファ(FDB)の操作	FCOL、FGET、FNOTE、FPOS、FPUT、FSEP
ライブラリ参照名の割り当ておよび確認	LIBNAME、LIBREF、PATHNAME
実行されたホスト環境のコマンドに関する情報の取得	SYSRC

注: もちろん、%SYSFUNC を使用して、ABS、MAX、TRANWRD などの他の関数を使用することもできます。ただし、いくつかの SAS 言語関数は、%SYSFUNC では使用できません。詳細については、"%SYSFUNC 関数と%QSYSFUNC 関数" (273 ページ) を参照してください。

%SYSFUNC の使用例

次のプログラムでは、ファイル参照名 MYFILE で指定されたファイルを削除しています。

```
%macro testfile(filrf);
%let
rc=%sysfunc(filename(filrf,physical-filename));
%if &rc = 0 and %sysfunc(fexist(&filrf)) %then
%let rc=%sysfunc(fdelete(&filrf));
%let rc=%sysfunc(filename(filrf));
%mend testfile;
```

```
%testfile(myfile)
```

ホスト固有の値を持つ自動変数の使用

タスク別のマクロ変数

すべてのホスト環境で自動マクロ変数を使用できますが、それらの値は各ホストによって決められます。次の表に、タスク別のマクロ変数を示します。"タイプ"列は、変数が変更可能(読み込み/書き込み)か、それとも参照可能(読み込み専用)かを示しています。

表 11.2 ホスト固有の値を持つ自動マクロ変数

タスク	自動マクロ変数	タイプ
DEVICE=で設定した現在のグラフィックデバイスの名前を表示します。	SYSDEVIC	読み込み/書き込み
実行モード(FORE または BACK)を表示します。一部のホスト環境では、1 つのモード(FORE)のみが可能です。	SYSENV	読み込み専用
現実行しているバッチジョブの名前、ユーザー ID、またはプロセス ID を表示します。たとえば、UNIX の場合、SYSJOBID の値はプロセス ID になります。	SYSJOBID	読み込み専用
ホスト環境によって最後に生成されたリターンコードを表示します。この値は、オープンコード内の X ステートメント、SAS ウィンドウ環境での X コマンド、または%SYSEXEC(あるいは%TSO や%CMS)マクロステートメントを使用して実行されたコマンドのリターンコードです。 デフォルト値は、0 です。	SYSRC	読み込み/書き込み
使用しているホスト環境の省略形を表示します。	SYSSCP	読み込み専用
使用しているホスト環境の詳細な省略形を表示します。	SYSSCPL	読み込み専用
SYSPARM=システムオプションによって SAS に渡された文字列を取得します。	SYSPARM	読み込み/書き込み

SYSSCP と SYSSCPL の使用例

次に示すマクロ DELFILE は、SYSSCP の値を使用して、SAS を実行しているプラットフォームを決定し、TMP ファイルを削除します。FILEREF は、ファイル名が格納されたマクロパラメータです。ファイル名は、ホスト固有です。そのため、ファイル名をマクロパラメータにすることにより、ホスト環境に必要なファイル名構文はすべて、マクロが使用できるになります。

```
%macro delfile(fileref);
```

```
/* Unix */
%if &sysscp=HP 800 or &sysscp=HP 300 %then %do;
X "rm &fileref..TMP";
%end;

/* DOS-LIKE platforms */
%else %if &sysscp=OS2 or &sysscp=WIN %then %do;
X "DEL &fileref..TMP";
%end;
/* CMS */
%else %if &sysscp=CMS %then %do;
X "ERASE &fileref TMP A";
%end;
%mend delfile;
```

PC 環境でのマクロ DELFILE の呼び出しを、次に示します。ここでは、C:\SAS \SASUSER\DOC1.TMP という名前のファイルを削除しています。

```
%delfile(c:\sas\sasuser\doc1)
```

このプログラムでは、ポータブルな%SYSEXEC ステートメントを使用して、ホスト固有のオペレーティングシステムのコマンドを実行していることに注意してください。

ここで、いずれかのバージョンの Microsoft Windows 上でマクロアプリケーションが実行されるということが、わかっているとします。SYSSCPL 自動マクロ変数は、SYSSCP 自動マクロ変数と同様に、ホスト環境の名前に関する情報を提供します。ただし、SYSSCPL のほうが詳細な情報を提供するため、それによってマクロコードを細かく調整できます。

SYSPARM の使用例

SYSPARM=システムオプションに都市名を設定する場合を考えます。つまり、SYSPARM 自動変数に、その都市名が設定されます。この値を使用してデータセットをサブセット化し、この値に固有のコードを生成できます。SAS を呼び出すコマンド(または SAS 構成ファイル)に対してわずかな変更を行うだけで、SAS ジョブは別のタスクを実行します。

```
/* Create a data set, based on the value of the */
/* SYSPARM automatic variable. */
/* An example data set name could be MYLIB.BOSTON. */
data mylib.&sysparm;
set mylib.alltowns;
/* Use the SYSPARM SAS language function to */
/* compare the value (city name) */
/* of SYSPARM to a data set variable. */
if town=sysparm();
run;
```

このプログラムを実行すると、対象となる都市のデータのみを含むデータセットが得られます。生成するデータセットを変更し、その後、SAS ジョブ開始できます。

ここで、やはり SYSPARM の値を使用して、ジョブが使用するプロシジャを制御したいとします。次のマクロは、それを実行しています。

```
%macro select;
%if %upcase(&sysparm) eq BOSTON %then
%do;
proc report ... more SAS code;
title "Report on &sysparm";
```

```
run;
%end;

%if %upcase(&sysparm) eq CHICAGO %then
%do;
proc chart ... more SAS code;
title "Growth Values for &sysparm";
run;
%end;
.
. /* more macro code */
.
%mend select;
```

SYSPARM の詳細

SYSPARM 自動マクロ変数の値は、SYSPARM=システムオプションの値と同じであり、SAS 言語関数 SYSPARM の戻り値と等価です。デフォルト値は、null です。SAS の起動時に SYSPARM=システムオプションを使用できるため、SYSPARM 自動マクロ変数の値を設定してから SAS セッションを開始できます。

SYSRC の詳細

SYSRC 自動マクロ変数には、ホスト環境によって生成された最後のリターンコードが格納されます。返されるコードは、オープンコード内の X ステートメント、ウィンドウ環境の X コマンド、または%SYSEXEC マクロステートメント(あるいは、非ポータブルの%TSO および%CMS マクロステートメント)を使用して実行したコマンドに基づきます。ホスト環境のコマンドが成功したかどうかをテストする場合、この SYSRC 自動マクロ変数を使用します。

注: SAS ログにエラーメッセージは生成されませんが、SYSRC 自動マクロ変数をすべてのホスト環境で使用できるわけではありません。たとえば、一部のホスト環境では、ホスト環境のコマンドが成功したかどうかにかかわらず、この変数の値は常に 99 になります。ホスト環境で SYSRC 自動マクロ変数を使用できるかどうかを確認するには、使用しているホスト環境に関する SAS ドキュメントを調べてください。

システム依存のマクロ言語要素

次に示すような、いくつかのマクロ言語要素は、ホスト固有です。

並べ替えシーケンスに依存する言語要素
そのような式の例には、%DO、%DO %UNTIL、%DO %WHILE、%IF-%THEN、%EVAL などがあります。

たとえば、次のプログラムについて考えます。

```
%macro testsort(var);
%if &var < a %then %put *** &var is less than a ***;
%else %put *** &var is greater than a ***;
%mend testsort;
%testsort(1)
/* Invoke the macro with the number 1 as the parameter. */
```

z/OS などの EBCDIC システムや VSE では、このプログラムによって、次のメッセージが SAS ログに書き込まれます。

```
*** 1 is greater than a ***
```

ところが、UNIX や Windows などの ASCII システムでは、次のメッセージが SAS ログに書き込まれます。

```
*** 1 is less than a ***
```

MSYMTABMAX=
このシステムオプションは、マクロ変数のシンボルテーブルで使用可能な最大メモリ量を指定します。シンボルテーブルは、この値を超えるとディスク上の WORK ファイルに保存されます。

MVARSIZE=
このシステムオプションは、メモリに格納される任意のマクロ変数の最大バイト数を指定します。マクロ変数は、この値を超えるとディスク上の WORK ファイルに保存されます。

%SCAN と%QSCAN
%SCAN 関数と%QSCAN 関数によって文字列内のワードの検索に使用されるデフォルトの区切り文字は、ASCII システムと EBCDIC システムとでは異なります。デフォルトの区切り文字は次のとおりです。

ASCII システム
空白 . < (+ & ! $ *) ; ^ − / , % |

EBCDIC システム
空白 . < (+ | & ! $ *) ; ¬ − / , % ¦ ¢

%SYSEXEC、%TSO、および%CMS
%SYSEXEC、%TSO、%CMS のいずれかのマクロステートメントを使用して、ホスト環境のコマンドを実行できます。

%SYSGET
一部のホスト環境では、%SYSGET 関数によって、ホスト環境変数の値およびシンボルが返されます。

SYSPARM=
このシステムオプションによって、SAS の起動時に SYSPARM 自動マクロ変数の値を指定できます。これは、プロダクションジョブのカスタマイズに役立ちます。たとえば、非対話型の実行の一部として、都市に基いてタイトルを作成するために、プロダクションプログラムで使用する SYSPARM=システムオプションを、SAS 構成ファイルまたは SAS の起動コマンドに含めることができます。SYSPARM=システムオプションと SYSPARM 自動マクロ変数を併用する例については、“SYSPARM の詳細” (154 ページ) を参照してください。

SASMSTORE=
このシステムオプションは、コンパイル済みマクロの場所を指定します。

SASAUTOS=
このシステムオプションは、自動呼び出しマクロの場所を指定します。

ホスト固有のマクロ変数

一部のホスト環境では、固有のマクロ変数が作成されます。それらのマクロ変数は、自動マクロ変数ではありません。次の表に、一般的に使用されるホスト固有のマクロ変数の一部を示します。今後のリリースで、使用可能なホスト固有のマクロ変数が追加される可能性があります。詳細については、SAS ドキュメントを参照してください。

表 11.3 z/OS のホスト固有のマクロ変数

変数名	説明
SYS99ERR	SVC99 のエラー理由コード
SYS99INF	SVC99 の情報理由コード
SYS99MSG	SVC のエラー理由コードまたは情報理由コードに対応する YSC99 のテキストメッセージ
SYS99R15	SVC99 のリターンコード
SYSJCTID	JCT 制御ブロック内の JCTUSER フィールドの値
SYSJMRID	JCT 制御ブロック内の JMRUSEID フィールドの値
SYSUID	SAS セッションに関連付けられた TSO ユーザー ID

自動呼び出し機能で使用するマクロと外部ファイルに名前を付ける

自動呼び出しライブラリに保存するマクロに名前を付ける場合、使用しているホスト環境によって異なる制限があります。それらの制限の一部を次に示します。

- すべてのホスト環境には、ファイル命名規則があります。ホスト環境でファイル拡張子が使用されている場合、マクロファイルの拡張子として、`.sas` を使用します。
- SAS 名にはアンダースコアを含めることができますが、一部のホスト環境では、外部ファイルの名前にアンダースコアを含めることができません。アンダースコアが使用されない一部のホスト環境ではシャープ記号(#)が使用されますが、マクロを使用するときに#が自動的にアンダースコアに置換される場合があります。
- 一部のホスト環境には、CON や NULL などの予約語があります。自動呼び出しマクロや外部ファイルに名前を付けるときに、予約語を使用しないでください。
- 一部のホストには、ホスト固有の自動呼び出しマクロがあります。これらの自動呼び出しマクロと同じ名前でマクロを定義しないでください。
- マクロカタログは、ポータブルではありません。マクロソースコードを、忘れずに、必ず安全な場所に保存するようにしてください。
- UNIX システムの場合、自動呼び出しマクロを保存するファイルの名前を、すべて小文字にする必要があります。

12 章

マクロ言語要素

マクロ言語要素

SAS マクロ言語は、ステートメント、関数、および自動マクロ変数で構成されてます。このセクションでは、これらの要素を定義し、一覧を示します。

- “マクロステートメント ” (158 ページ)
- “マクロ関数 ” (160 ページ)
- “自動マクロ変数” (166 ページ)

また、Base SAS ソフトウェア、SQL プロシジャ、および SAS コンポーネント言語が提供するマクロ機能とのインターフェイスについて説明する他、自動呼び出しマクロとマクロのシステムオプションについても説明します。

マクロステートメント

マクロステートメントの使用

マクロ言語ステートメントは、マクロプロセッサに特定の操作を実行するよう命令します。マクロ言語ステートメントは、キーワードの文字列、SAS 名、および特殊文字と演算子から成り、セミコロンで終わります。一部のマクロ言語ステートメントは、マクロ定義の内部でのみ使用できます。それ以外のマクロステートメントは、SAS セッションや SAS ジョブの任意の場所で、マクロ定義の内外にかかわらず使用できます。SAS セッションや SAS ジョブにおけるマクロ定義の外側のことをオープンコードと呼びます。マクロ定義とオープンコードの両方で使用可能なマクロ言語ステートメントを、次の表に示します。

表 12.1 マクロ定義とオープンコードで使用されるマクロ言語ステートメント

ステートメント	説明
%* コメント	コメントテキストを指定します。
%COPY	SAS ライブラリから、指定された項目をコピーします。
%DISPLAY	マクロウィンドウを表示します。
%GLOBAL	SAS セッションの実行中に、SAS セッション全体で使用可能なマクロ変数を作成します。
%INPUT	マクロの実行中に、マクロ変数に値を入力します。
%LET	マクロ変数を作成し、それに値を割り当てます。
%MACRO	マクロ定義を開始します。
%PUT	テキストまたはマクロ変数の値を、SAS ログに書き込みます。
%SYMDEL	引数で指定されたマクロ変数を削除します。
%SYSCALL	SAS CALL ルーチンを呼び出します。
%SYSEXEC	オペレーティングステムのコマンドを実行します。
%SYSLPUT	リモートホスト上またはリモートサーバー上で新しいマクロ変数を定義したり、既存のマクロ変数の値を変更したりします。
%SYSMACDELETE	WORK.SASMACR カタログからマクロ定義を削除します。
%SYSMSTORECLEAR	コンパイル済みマクロを終了し、SASMSTORE=ライブラリをクリアします。

ステートメント	説明
%SYSRPUT	リモートホスト上のマクロ変数の値を、ローカルホスト上のマクロ変数に割り当てます。
%WINDOW	カスタマイズされたウィンドウを定義します。

マクロ定義内でのみ使用可能なマクロ言語ステートメントを、次の表に示します。

表 12.2 *マクロ定義内でのみ使用されるマクロ言語ステートメント*

ステートメント	説明
%ABORT	現在の DATA ステップ、SAS ジョブ、または SAS セッションで実行されているマクロを停止します。
%DO	%DO グループを開始します。
%DO (反復)	インデックス変数の値に基いて、ステートメントを反復して実行します。
%DO %UNTIL	条件が true になるまで、ステートメントを反復して実行します。
%DO %WHILE	条件が true である間、ステートメントを反復して実行します。
%END	%DO グループを終了します。
%GOTO	指定されたラベルにマクロ処理を分岐します。
%IF-%THEN/%ELSE	マクロの一部を条件付きで処理します。
%ラベル	%GOTO ステートメントの分岐先を指定します。
%LOCAL	マクロ変数を作成します。このマクロ変数は、それが定義されているマクロの実行中にのみ使用可能です。
%MEND	マクロ定義を終了します。
%RETURN	実行中のマクロを正常終了します。

自動評価を実行するマクロステートメント

一部のマクロステートメントは、算術演算式または論理式の評価に基いて、演算を実行します。評価は、%EVAL 関数を自動的に呼び出すことによって実行されます。マクロで%EVAL 以外のステートメントを使用しているときに、%EVAL の問題を示すエラーメッセージが表示された場合、次のいずれかのステートメントを確認します。これらのマクロステートメントは、自動評価を実行します。

- %DO *macro-variable=expression* %TO *expression* <%BY *expression*>;
- %DO %UNTIL(*expression*);

- %DO %WHILE(*expression*);
- %IF *expression* %THEN *action*;

式のオペランドと演算子の詳細については、“マクロ式” (73 ページ)を参照してください。

マクロ関数

マクロ関数の使用

各マクロ言語関数は、1 つ以上の引数を処理することで結果を生成します。すべてのマクロ関数を、マクロ定義とオープンコードの両方で使用できます。マクロ関数には、文字関数、評価関数、クォーティング関数などがあります。マクロ言語関数を次の表に示します。

***表 12.3** マクロ関数*

関数	説明
%BQUOTE、%NRBQUOTE	マクロの実行時に、置換された値に含まれる特殊文字とニーモニック演算子をマスクします。
%EVAL	整数演算を使用して、算術演算式と論理式を評価します。
%INDEX	文字列の最初の文字の位置を返します。
%LENGTH	文字列の長さを返します。
%QUOTE、%NRQUOTE	マクロの実行時に、置換された値に含まれる特殊文字とニーモニック演算子をマスクします。一致しない引用符(“ ”)とかっこ(())には、文字の前に%を挿入してマークを付ける必要があります。
%SCAN、%QSCAN	番号で指定されたワードを検索します。%QSCAN は、結果に含まれる特殊文字とニーモニック演算子をマスクします。
%STR、%NRSTR	マクロのコンパイル時に、定数テキストに含まれる特殊文字とニーモニック演算子をマスクします。一致しない引用符(“ ”)とかっこ(())には、文字の前に%を挿入してマークを付ける必要があります。
%SUBSTR、%QSUBSTR	文字列の部分文字列を生成します。%QSUBSTR は、結果に含まれる特殊文字とニーモニック演算子をマスクします。
%SUPERQ	マクロの実行時に、すべての特殊文字とニーモニック演算子をマスクして、値の置換が行われないようにします。
%SYMEXIST	指定されたマクロ変数が存在するかどうかを示す値を返します。
%SYMGLOBL	指定されたマクロ変数のスコープがグローバルかどうかを示す値を返します。

関数	説明
%SYMLOCAL	指定されたマクロ変数のスコープがローカルかどうかを示す値を返します。
%SYSEVALF	浮動小数点演算を使用して、算術演算式と論理式を評価します。
%SYSFUNC、%QSYSFUNC	SAS 関数またはユーザー作成関数を実行します。%QSYSFUNC は、結果に含まれる特殊文字とニーモニック演算子をマスクします。
%SYSGET	指定されたホスト環境変数の値を返します。
%SYSMACEXEC	マクロが現在実行中かどうかを示します。
%SYSMACEXIST	WORK.SASMACR カタログにマクロ定義があるかどうかを示します。
%SYSMEXECDEPTH	呼び出し点からのネストの深さを返します。
%SYSMEXECNAME	ネストレベルで実行しているマクロの名前を返します。
%SYSPROD	SAS ソフトウェアプロダクトがサイトでライセンスされているかどうかをレポートします。
%UNQUOTE	値に含まれるすべての特殊文字とニーモニック演算子のマスクを解除します。
%UPCASE、%QUPCASE	文字列を大文字に変換します。%QUPCASE は、結果に含まれる特殊文字とニーモニック演算子をマスクします。

マクロ文字関数

文字関数は、文字列を変更したり、文字列に関する情報を提供したりします。マクロ文字関数を次の表に示します。

表 12.4 *マクロ文字関数*

関数	説明
%INDEX	文字列の最初の文字の位置を返します。
%LENGTH	文字列の長さを返します。
%SCAN、%QSCAN	番号で指定されたワードを検索します。%QSCAN は、結果に含まれる特殊文字とニーモニック演算子をマスクします。
%SUBSTR、%QSUBSTR	文字列の部分文字列を生成します。%QSUBSTR は、結果に含まれる特殊文字とニーモニック演算子をマスクします。

関数	説明
%UPCASE、%QUPCASE	文字列を大文字に変換します。%QUPCASE は、結果に含まれる特殊文字とニーモニック演算子をマスクします。

名前が Q で始まる場合と始まらない場合の 2 つがあるマクロ文字関数(たとえば、%QSCAN と%SCAN)の場合、Q で始まる関数が結果に含まれる特殊文字とニーモニック演算子をマスクするということ以外、それら 2 つの関数は同じ動作をします。引数がマクロクォーティング関数を使用してすでにマスクされている場合、またはマスクされた結果が必要な場合(たとえば、一致しない引用符やかっこが結果に含まれる可能性がある場合)、名前が Q で始まる関数を使用します。詳細については、“マクロクォーティング” (82 ページ)を参照してください。

多くのマクロ文字関数の名前は SAS 文字関数に対応しており(たとえば、%SUBSTR と SUBSTR)、それらは同じようなタスクを実行します。ただし、マクロ関数は、DATA ステップが実行される前に動作します。次の DATA ステップについて考えてみます。

```
data out.%substr(&sysday,1,3); /* macro function */
set in.weekly (keep=name code sales);
length location $4;
location=substr(code,1,4); /* SAS function */
run;
```

このプログラムを月曜に実行すると、次のように、OUT.MON というデータセット名が作成されます。

```
data out.MON; /* macro function */
set in.weekly (keep=name code sales);
length location $4;
location=substr(code,1,4); /* SAS function */
run;
```

IN.WEEKLY の変数 CODE に、cary18593 および apex19624 という値が含まれているとします。SAS 関数 SUBSTR は、DATA ステップの実行時に動作して、**cary** および **apex** という値を変数 LOCATION に割り当てます。

マクロ評価関数

評価関数は、算術演算式と論理式を評価します。これらの関数は、引数に含まれるオペランドを一時的に数値に変換します。次に、これらの関数は、オペランドによって指定された演算を実行し、その結果を文字値に変換します。マクロプロセッサは、評価関数を使用して、次のことを実行します。

- 文字の比較
- 論理(ブール)式の評価
- 数値プロパティ(関数の引数に含まれる整数など)のトークンへの割り当て

詳細については、“マクロ式” (73 ページ)を参照してください。次の表に、マクロ評価関数を示します。

表 12.5 *マクロ評価関数*

関数	説明
%EVAL	整数演算を使用して、算術演算式と論理式を評価します。

関数	説明
%SYSEVALF	浮動小数点演算を使用して、算術演算式と論理式を評価します。

%EVAL は、次の関数の評価を実行するステートメントで、マクロプロセッサによって自動的に呼び出され、引数に含まれる式を評価します。

- %QSCAN(*argument*, *n*<, *delimiters*>)
- %QSUBSTR(*argument*, *position*<, *length*>)
- %SCAN(*argument*, *n*<, *delimiters*>)
- %SUBSTR(*argument*, *position*<, *length*>)

マクロクォーティング関数

マクロクォーティング関数は、マクロプロセッサが特殊文字とニーモニック演算子をマクロ言語の要素としてではなくテキストとして解釈できるようにするために、それらの文字をマスクします。

次の表では、マクロクォーティング関数を示し、それらがマスクする特殊文字と、それらが動作するタイミングについて説明します(%QSCAN、%QSUBSTR、および%QUPCASE は、結果に含まれる特殊文字とニーモニック演算子をマスクしますが、これらの関数はクォーティング関数とは見なされません。これは、これらの関数の目的が文字値を処理することにあり、単に値をクォーティングすることではないためです)。詳細については、“マクロクォーティング” (82 ページ)を参照してください。

表 12.6 *マクロクォーティング関数*

関数	説明
%BQUOTE、%NRBQUOTE	マクロの実行時に、置換された値に含まれる特殊文字とニーモニック演算子をマスクします。%BQUOTE と%NRBQUOTE は、一致しない引用符(“ ”)とかっこ(())にマークを付ける必要がないため、実行時に値をマスクする最も強力な関数です。
%QUOTE、%NRQUOTE	マクロの実行時に、置換された値に含まれる特殊文字とニーモニック演算子をマスクします。一致しない引用符(“ ”)とかっこ(())には、文字の前に%を挿入してマークを付ける必要があります。
%STR、%NRSTR	マクロのコンパイル時に、定数テキストに含まれる特殊文字とニーモニック演算子をマスクします。一致しない引用符(“ ”)とかっこ(())には、文字の前に%を挿入してマークを付ける必要があります。
%SUPERQ	マクロの実行時に、すべての特殊文字とニーモニック演算子をマスクして、値の置換が行われないようにします。
%UNQUOTE	値に含まれるすべての特殊文字とニーモニック演算子のマスクを解除します。

コンパイルクォーティング関数

%STR と%NRSTR は、マクロ定義またはオープンコード内のマクロ言語ステートメントのコンパイル時に、値に含まれる特殊文字とニーモニック演算子をマスクします。たとえば、次の%STR 関数は、%LET ステートメントが誤って終了しないようにしています。この関数は、PROC PRINT ステートメントに含まれるセミコロンが、%LET ステートメントのセミコロンとして解釈されないようにしています。

```
%let printit=%str(proc print; run;);
```

マクロクォーティング関数の実行

%BQUOTE、%NRBQUOTE、%QUOTE、%NRQUOTE、および%SUPERQ は、マクロまたはオープンコード内のマクロ言語ステートメントの実行時に、値に含まれる特殊文字とニーモニック演算子をマスクします。%SUPERQ を除くこれらの関数は、マクロ式を可能な限り置換するようにマクロプロセッサに指示し、その結果をマスクし、置換できなかったマクロ変数参照またはマクロ呼び出しに対しては、警告メッセージを発行します。%SUPERQ は、それ以上置換が行われないように、マクロ変数の値を保護します。

実行時に値を置換するクォーティング関数のうち、%BQUOTE と%NRBQUOTE が最も柔軟性を持ちます。たとえば、次の%BQUOTE 関数は、マクロ変数 STATE が OR(Oregon の略称)に置換された場合に、%IF ステートメントでエラーが発生しないようにしています。%BQUOTE を使用しないと、マクロプロセッサは、Oregon の略称を論理演算子 OR として解釈します。

```
%if %bquote(&state)=nc %then %put North Carolina Dept. of
Revenue;
```

%SUPERQ は、マクロシンボルテーブルからマクロ変数の値を取得して、それを即座にマスクし、置換されたその値のどの部分もマクロプロセッサによって置換されないようにします。たとえば、次の%LET ステートメントでは、%SUPERQ を使用して、このステートメントがアンパサンドを含む値(**Smith&Jones** など)に置換された場合にエラーが発生しないようにしています。%SUPERQ を使用しないと、マクロプロセッサは **&Jones** を置換しようとします。

```
%let testvar=%superq(corpname);
/* No ampersand in argument to %superq. */
```

(%SUPERQ は、引数として、アンパサンドを含まないマクロ変数名、またはマクロ変数名を生成するテキスト式のいずれかを受け取ります。)

一致しない引用符とかっこ

%STR、%NRSTR、%QUOTE、および%NRQUOTE の引数に、一致しない引用符またはかっこが含まれる場合、構文エラーが発生します。これらのエラーを回避するには、一致しない引用符とかっこの前にパーセント記号を挿入してマークを付けます。たとえば、**値 345)** をマクロ変数 B に格納するには、次のように記述します。

```
%let b=%str(345%));
```

%STR、%NRSTR、%QUOTE、または%NRQUOTE の引数に、前にパーセント記号の付いた引用符またはかっこを含める場合は、引数のパーセント記号が引用符またはかっこのためのマークではないことを指定するために、2 つのパーセント記号(%%)を使用します。たとえば、値 **TITLE "20%";**をマクロ変数 P に格納するには、次のように記述します。

```
%let p=%str(TITLE "20%%";);
```

これらの関数のいずれかの引数に、コメントシンボル(/*および-->)を含む文字列を格納する場合、各文字に対して%STR 関数を使用します。たとえば、次のステートメントを考えます。

```
%let instruct=Comments can start with %str(/)%str(*).;
%put &instruct;
```

これによって、次の行が SAS ログに書き込まれます。

```
Comments can start with /*
```

注: クォーティング関数を使用してコメントシンボルをクォーティングしなかった場合、予期しない結果が生じる恐れがあります。

マクロクォーティングの詳細については、“マクロクォーティング” (82 ページ)を参照してください。

DBCS (ダブルバイト文字セット)用のマクロ関数

東アジア言語には数千の文字があるため、各文字を表現するには、ダブル(2)バイトの情報が必要です。各東アジア言語には、通常、複数の DBCS エンコード体系があります。SAS は、主要な東アジア言語に固有の DBCS エンコード情報を処理します。DBCS をサポートするマクロ関数を、次の表で定義します。

表 12.7 *DBCS 用のマクロ関数*

関数	説明
%KCMPRES	複数の空白を圧縮し、先頭と末尾の空白を削除します。
%KINDEX	文字列の最初の文字の位置を返します。
%KLEFT および%QKLEFT	先頭の空白を削除することによって、引数を左に揃えます。
%KLENGTH	文字列の長さを返します。
%KSCAN および%QKSCAN	位置で指定されたワードを文字列から検索します。
%KSUBSTR および%QKSUBSTR	%KSUBSTR と%QKSUBSTR は、文字列の部分文字列を生成します。
%KUPCASE および%QKUPCASE	値を大文字に変換します。

詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*の“Macro Functions for NLS”を参照してください。

その他のマクロ関数

その他の 7 つのマクロ関数は、これまで述べたカテゴリには当てはまりませんが、重要な情報を提供します。それらの関数を、次の表に示します。

表 12.8 その他のマクロ関数

関数	説明
%SYMEXIST	指定されたマクロ変数が存在するかどうかを示す値を返します。
%SYMGLOBL	指定されたマクロ変数のスコープがグローバルかどうかを示す値を返します。
%SYMLOCAL	指定されたマクロ変数のスコープがローカルかどうかを示す値を返します。
%SYSFUNC、%QSYSFUNC	SAS 言語関数またはユーザー作成の関数をマクロ機能内で実行します。
%SYSGET	指定されたホスト環境変数の値を返します。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。
%SYSPROD	SAS ソフトウェアプロダクトがサイトでライセンスされているかどうかをレポートします。

%SYSFUNC 関数と%QSYSFUNC 関数を使用すると、Base SAS ソフトウェアのほとんどの関数、SAS/TOOLKIT ソフトウェアを使用して作成した関数、または FCMP プロシジャを使用して作成した関数が、マクロ機能で利用できるようになります。次に例を示します。

```
• /* in a DATA step or SCL program */
dsid=open("sasuser.houses","i");
• /* in the macro facility */
%let dsid = %sysfunc(open(sasuser.houses,i));
```

詳細については、“%SYSFUNC 関数と%QSYSFUNC 関数” (273 ページ)を参照してください。

自動マクロ変数

自動マクロ変数は、マクロプロセッサによって作成され、さまざまな情報を提供します。これらは、プログラム内でコードを実行する前に、条件のステータスをチェックする場合に役立ちます。&SYSLAST や&SYSJOBID などの自動マクロ変数を使用する場合、ユーザーが作成したマクロ変数と同じ方法で、それらを参照できます。

注意:

名前が SYS で始まるマクロ変数を作成しないでください。 3 文字の接頭語 SYS は、SAS が自動マクロ変数に使用するため、予約されています。マクロ言語の予約語の完全な一覧については、“ マクロ機能の予約語” (369 ページ)を参照してください。

たとえば、現在の SAS セッションが起動された曜日と日付を含める場合を考えます。自動マクロ変数 SYSDAY と SYSDATE9 を参照するには、次のように FOOTNOTE ステートメントを記述します。

```
footnote "Report for &sysday, &sysdate9";
```

現在の SAS セッションが 2007 年 6 月 13 日に起動された場合、マクロ変数が置換されることによって、このステートメントは次のように解釈されます。

```
FOOTNOTE "Report for Friday, 13JUN2007";
```

SYSPBUFF を除くすべての自動マクロ変数は、グローバルであり、SAS の起動時に作成されます。次の表に、自動マクロ変数の一覧と、それらの読み込み/書き込みステータスを示します。

表 12.9 *自動マクロ変数*

変数	読み込み/書き込みステータス
SYSADDRBITS	読み込み専用
SYSBUFFR	読み込み/書き込み
SYSCC	読み込み/書き込み
SYSCHARWIDTH	読み込み専用
SYSCMD	読み込み/書き込み
SYSDATE	読み込み専用
SYSDATE9	読み込み専用
SYSDAY	読み込み専用
SYSDEVIC	読み込み/書き込み
SYSDMG	読み込み/書き込み
SYSDSN	読み込み/書き込み
SYSENCODING	読み込み専用
SYSENDIAN	読み込み専用
SYSENV	読み込み専用
SYSERR	読み込み専用
SYSERRORTEXT	読み込み専用
SYSFILRC	読み込み/書き込み
SYSHOSTNAME	読み込み専用
SYSINDEX	読み込み専用
SYSINFO	読み込み専用
SYSJOBID	読み込み専用

変数	読み込み/書き込みステータス
SYSLAST	読み込み/書き込み
SYSLCKRC	読み込み/書き込み
SYSLIBRC	読み込み/書き込み
SYSLOGAPPLNAME	読み込み専用
SYSMACRONAME	読み込み専用
SYSMENV	読み込み専用
SYSMSG	読み込み/書き込み
SYSNCPU	読み込み専用
SYSNOBS	読み込み専用
SYSODSESCAPECHAR	読み込み専用
SYSODSPATH	読み込み専用
SYSPARM	読み込み/書き込み
SYSPBUFF	読み込み/書き込み
SYSPROCESSID	読み込み専用
SYSPROCESSNAME	読み込み専用
SYSPROCNAME	読み込み専用
SYSRC	読み込み/書き込み
SYSSCP	読み込み専用
SYSSCPL	読み込み専用
SYSSITE	読み込み専用
SYSSIZEOFLONG	読み込み専用
SYSSIZEOFPTR	読み込み専用
SYSSIZEOFUNICODE	読み込み専用
SYSSTARTID	読み込み専用
SYSSTARTNAME	読み込み専用

変数	読み込み/書き込みステータス
SYSTCPIPHOSTNAME	読み込み専用
SYSTIME	読み込み専用
SYSUSERID	読み込み専用
SYSVER	読み込み専用
SYSVLONG	読み込み専用
SYSVLONG4	読み込み専用
SYSWARNINGTEXT	読み込み専用

マクロ機能とのインターフェイス

DATA ステップ、SAS コンポーネント言語、および SQL プロシジャは、マクロ機能とのインターフェイスを提供します。次の表に、SAS マクロ機能を操作する要素を示します。

DATA ステップには、DATA ステップの実行中にプログラムからマクロ機能を操作できるようにする要素が用意されています。

表 12.10 *DATA ステップとのインターフェイス*

要素	説明
EXECUTE ルーチン	引数を置換し、置換した値を次のステップ境界で実行します。
RESOLVE 関数	DATA ステップの実行中に、テキスト式の値を置換します。
SYMDEL ルーチン	引数で指定されたマクロ変数を削除します。
SYMEXIST 関数	指定されたマクロ変数が存在するかどうかを示す値を返します。
SYMGET 関数	DATA ステップの実行時に、マクロ変数の値を DATA ステップに返します。
SYMGLOBL 関数	指定されたマクロ変数のスコープがグローバルかどうかを示す値を返します。
SYMLOCAL 関数	指定されたマクロ変数のスコープがローカルかどうかを示す値を返します。
SYMPUT/SYMPUTX ルーチン	DATA ステップで生成された値をマクロ変数に割り当てます。

SAS コンポーネント言語(SCL)には、SAS マクロ機能を使用して SCL プログラムのマクロとマクロ変数を定義するための、2 つの要素が用意されています。

表 12.11 SAS コンポーネント言語とのインターフェイス

要素	説明
SYMGETN	グローバルマクロ変数の値を数値として返します。
SYMPUTN	数値をグローバルマクロ変数に割り当てます。

SQL プロシジャには、SQL プロシジャが生成した値を使用してマクロ変数を作成および更新する機能が備わっています。

表 12.12 SQL プロシジャとのインターフェイス

要素	説明
INTO	計算の結果、またはデータ列の値を割り当てます。

詳細については、“マクロ機能とのインターフェイス” (103 ページ)を参照してください。

SAS が提供する自動呼び出しマクロ

提供される自動呼び出しマクロの概要

SAS は、自動呼び出しマクロのライブラリを各 SAS サイトに提供します。提供されるライブラリは、サイトでライセンスを取得した SAS プロダクトによって異なります。自動呼び出しマクロは、プログラムで定義したり含めたりしなくても使用できます。

SAS をインストールすると、自動呼び出しライブラリは、システム構成ファイル内の SASAUTOS システムオプションの値に含められます。自動呼び出しマクロは、個々のメンバとして保存され、それらにはマクロ定義が含まれています。各メンバには、それに含まれているマクロ定義と同じ名前が付けられています。

SAS が提供する自動呼び出しライブラリで利用できるマクロは、動作するユーティリティプログラムですが、それらをユーザー独自のルーチンのモデルとして使用することもできます。さらに、それらのマクロを、ユーザーが作成したマクロ内で呼び出すこともできます。

それらのマクロ定義を調べるには、各メンバの先頭にあるコメント化されたセクションを参照してください。自動呼び出しライブラリの場所を見つけるには、SAS システムオプション SASAUTOS の設定を参照してください。SASAUTOS の値を表示するには、次のいずれかを使用します。

- OPTIONS ウィンドウを開くための SAS ウィンドウ環境の OPTIONS コマンド
- OPTIONS プロシジャ
- VERBOSE システムオプション
- OPLIST システムオプション

これらのオプションの詳細については、*SAS システムオプション: リファレンス*の“SAS System Options”を参照してください。

提供される自動呼び出しマクロを、次の表に示します。

表 12.13 *提供される自動呼び出しマクロ*

マクロ	説明
CMPRES および QCMPRES	複数の空白を圧縮し、先頭と末尾の空白を削除します。QCMPRES は、結果をマスクして、マクロ機能によって特殊文字とニーモニック演算子が解釈されずに、テキストとして扱われるようにします。
COMPSTOR	マクロをコンパイルし、それらを永続的な SAS ライブラリ内のカタログに格納します。
DATATYP	値のデータタイプを返します。
LEFT および QLEFT	先頭の空白を削除することによって、引数を左に揃えます。QLEFT は、結果をマスクして、マクロ機能によって特殊文字とニーモニック演算子が解釈されずに、テキストとして扱われるようにします。
SYSRC	エラー状態に対応する値を返します。
TRIM および QTRIM	末尾の空白を削除します。QTRIM は、結果をマスクして、マクロ機能によって特殊文字とニーモニック演算子が解釈されずに、テキストとして扱われるようにします。
VERIFY	式に固有の最初の文字の位置を返します。

自動呼び出しマクロの必須システムオプション

自動呼び出しマクロを使用するには、次の 2 つの SAS システムオプションを設定する必要があります。

MAUTOSOURCE
: 自動呼び出し機能を有効にします。NOMAUTOSOURCE は、自動呼び出し機能を無効にします。

SASAUTOS=*library-specification* | (*library-specification-1*..., *library-specification-n*)
: 1 つ以上の自動呼び出しライブラリを指定します。詳細については、使用しているオペレーティングシステムの SAS ドキュメントを参照してください。

SAS が提供する自動呼び出しライブラリがサイトにインストールされており、SAS が提供する SAS ソフトウェアの標準構成を使用している場合、自動呼び出しマクロの使用を開始するには、SAS システムオプション MAUTOSOURCE が有効になっていることを確認するだけですみます。

MAUTOLOCDISPLAY システムオプションは必須ではありませんが、これを設定しておくと、自動呼び出しマクロを呼び出したときに、自動呼び出しマクロのソースの場所が SAS ログに表示されます。詳細については、“MAUTOLOCDISPLAY システムオプション” (339 ページ)を参照してください。

自動呼び出しマクロの使用

自動呼び出しマクロを使用するには、*%macro-name* というステートメントを使用して、プログラム内でそれを呼び出します。マクロプロセッサは、その名前を持つコンパイル済みマクロ定義について、まず、WORK ライブラリ内を検索します。マクロプロセッサは、コンパイル済みマクロを検出できなかった場合、MAUTOSOURCE が有効であれば、その名前を持つメンバについて、SASAUTOS オプションで指定されたライブラリ内を検索します。マクロプロセッサは、メンバを検出すると、次を実行します。

1. そのメンバ内のすべてのソースステートメントを、すべてのマクロ定義を含めてコンパイルします。
2. そのメンバにオープンコード(どのマクロ定義にも含まれないマクロステートメントまたは SAS ソースステートメント)があれば、それを実行します。
3. 呼び出した名前の付いたマクロを実行します。

マクロは、コンパイルが完了すると WORK.SASMACR カタログに保存され、再コンパイルを必要とせずに SAS セッション内で使用できるようになります。

独自の自動呼び出しマクロを作成し、それらを簡単に実行するために、ライブラリに保存することもできます。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

DBCS (ダブルバイト文字セット)用の自動呼び出しマクロ

東アジア言語には数千の文字があるため、各文字を表現するには、ダブル(2)バイトの情報が必要です。各東アジア言語には、通常、複数の DBCS エンコード体系があります。SAS は、主要な東アジア言語に固有の DBCS エンコード情報を処理します。次の表に、DBCS をサポートする自動呼び出しマクロの定義を示します。

表 12.14 *DBCS 用の自動呼び出しマクロ*

自動呼び出しマクロ	説明
%KLOWCASE および%QKLOWCAS	大文字を小文字に変更します。
%KTRIM および%QKTRIM	末尾の空白を削除します。
%KVERIFY	式に固有の最初の文字の位置を返します。

詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*の“Autocall Macros for NLS”を参照してください。

マクロ機能に使用されるシステムオプション

マクロ機能に適用される SAS システムオプションを、次の表に示します。

表 12.15 マクロ機能で使用されるシステムオプション

オプション	説明
CMDMAC	コマンドスタイルマクロ呼び出しを制御します。
IMPLMAC	ステートメントスタイルマクロ呼び出しを制御します。
MACRO	SAS マクロ言語を使用可能にするかどうかを制御します。
MAUTOCOMPLOC	自動呼び出しマクロのコンパイル時に、自動呼び出しマクロのソースの場所を SAS ログに表示します。
MAUTOLOCDISPLAY	自動呼び出しマクロが呼び出されたときに、自動呼び出しマクロのソースの場所を SAS ログに表示します。
MAUTOLOCINDES	マクロプロセッサが自動呼び出しソースファイルのフルパス名を、WORK.SASMACR カタログのコンパイル済み自動呼び出しマクロ定義のカタログエントリの説明フィールドに追加するかどうかを指定します。
MAUTOSOURCE	マクロ自動呼び出し機能を使用可能にするかどうかを制御します。
MCOMPILE	新しいマクロの定義を可能にします。
MCOMPILENOTE	マクロのコンパイルの完了時に、SAS ログに NOTE を出力します。
MCOVERAGE	カバレッジ分析データの生成を可能にします。
MCOVERAGELOC	カバレッジ分析データファイルの場所を指定します。
MERROR	マクロ形式の名前(*%name*)がコンパイル済みマクロと一致しない場合に、マクロプロセッサによって警告メッセージを発行するかどうかを制御します。
MEXECNOTE	マクロの呼び出し時に、マクロの実行情報を SAS ログに表示します。
MEXECSIZE	メモリ内で実行できるマクロの最大サイズを指定します。
MFILE	MPRINT 出力を外部ファイルに送信するかどうかを指定します。
MINDELIMITER	マクロの IN 演算子のデリミタとして使用される文字を指定します。
MINOPERATOR	マクロプロセッサで IN(#)論理演算子を認識するかどうかを制御します。
MLOGIC	デバッグのためにマクロの実行をトレースするかどうかを制御します。

オプション	説明
MLOGICNEST	マクロのネスト情報を、SAS ログの MLOGIC 出力に表示できるようにします。
MPRINT	マクロの実行によって生成された SAS ステートメントを、デバッグのためにトレースするかどうかを制御します。
MPRINTNEST	マクロのネスト情報を、SAS ログの MPRINT 出力に表示できるようにします。
MRECALL	マクロプロセッサによって、前の検索で検出できなかったメンバを、自動呼び出しライブラリから検索するかどうかを制御します。
MREPLACE	既存のマクロの再定義を可能にします。
MSTORED	コンパイル済みマクロを使用可能にするかどうかを制御します。
MSYMTABMAX	マクロ変数シンボルテーブルで使用可能な最大メモリ量を指定します。
MVARSIZE	メモリ内のマクロ変数値の最大サイズを指定します。
SASAUTOS	1 つ以上の自動呼び出しライブラリを指定します。
SASMSTORE	コンパイル済み SAS マクロのカタログを含む SAS ライブラリの、ライブラリ参照名を指定します。
SERROR	マクロ変数参照がマクロ変数と一致しない場合に、マクロプロセッサによって警告メッセージを発行するかどうかを制御します。
SYMBOLGEN	マクロ変数参照の置換結果を、デバッグのために表示するかどうかを制御します。
SYSPARM	SAS プログラムに渡すことのできる文字列を指定します。

2 部

マクロ言語リファレンス

13 章

自動呼び出しマクロ

自動呼び出しマクロ

SAS は、自動呼び出しマクロのライブラリを各 SAS サイトに提供します。提供されるライブラリは、サイトでライセンスを取得した SAS プロダクトによって異なります。自動呼び出しマクロは、プログラムで定義したり含めたりしなくても使用できます。

ディクショナリ

%CMPRES 自動呼び出しマクロと%QCMPRES 自動呼び出しマクロ

複数の空白を圧縮し、先頭と末尾の空白を削除します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%CMPRES (*text* | *text expression*)

%QCMPRES (*text* | *text expression*)

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

CMPRES マクロおよび QCMPRES マクロは、複数の空白を圧縮し、先頭と末尾の空白を削除します。次に示す特殊文字またはニーモニック演算子が引数に含まれる可能性がある場合は、%QCMPRES を使用してください。

CMPRES は、引数がクォーティングされている場合でも、クォーティング解除された結果を返します。QCMPRE は、次の特殊文字およびニーモニック演算子をマスクして結果を生成します。そのためマクロプロセッサは、それらをマクロ言語要素ではなく、テキストとして解釈します。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

例

例 1: %CMPRES を使用した不必要な空白の削除

```
%macro createft;
%let footnote="The result of &x &op &y is %eval(&x &op &y).";
footnote1 &footnote;
footnote2 %cmpres(&footnote);
%mend createft;
data _null_;
x=5;
y=10;
call symput('x',x); /* Uses BEST12. format */
call symput('y',y); /* Uses BEST12. format */
call symput('op','+'); /* Uses $1. format */
run;
%createft
```

CREATEFT マクロは、2 つのフットノートステートメントを生成します。

```
FOOTNOTE1 "The result of 5 + _________10 is _________15.";
FOOTNOTE2 "The result of 5 + 10 is 15.";
```

例 2: %QCMPRES と%CMPRES の比較

```
%let x=5;
%let y=10;
%let a=%nrstr(%eval(&x + &y));
%put QCMPRES: %qcmpres(&a);
%put CMPRES: %cmpres(&a);
```

%PUT ステートメントによって、次の行がログに書き込まれます。

```
QCMPRES: %eval(&x + &y)
CMPRES: 15
```

%COMPSTOR 自動呼び出しマクロ

マクロをコンパイルし、それらを永続的な SAS ライブラリ内のカタログに格納します。

種類: 自動マクロ

要件 MAUTOSOURCE システムオプション

構文

%COMPSTOR (PATHNAME=*SAS library*)

必須引数

SAS-data-library
ホストシステム上の SAS ライブラリの物理名。COMPSTOR マクロは、この値を使用してライブラリ参照名を自動的に割り当てます。*SAS library* を引用符で囲まないでください。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

COMPSTOR マクロは、永続的な SAS ライブラリ内の SASMACR という SAS カタログに含まれる、次に示す自動呼び出しマクロをコンパイルします。SAS セッションで初めてこれらのマクロを呼び出すときの、コンパイルによるオーバーヘッドが省かれます。COMPSTOR マクロは、コンパイル済みマクロの作成方法の例として使用できます。SAS が提供する自動呼び出しマクロや、コンパイル済みマクロの使用の詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%CMPRES	%QLEFT
%DATATYP	%QTRIM
%LEFT	%TRIM
%QCMPRES	%VERIFY

%DATATYP 自動呼び出しマクロ

値のデータタイプを返します。

種類: 自動呼び出しマクロ

制限事項: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。

要件 MAUTOSOURCE システムオプション

構文

%DATATYP (*text* | *text expression*)

詳細

DATATYP マクロは、引数が 10 進数、先頭のプラス記号またはマイナス記号、小数、指数または浮動小数点指数(大文字または小文字の E または D)で構成されている場合、**NUMERIC** の値を返します。そうでない場合、**CHAR の値を返します。**

注: %DATATYP は、16 進数を識別しません。

例: 値のデータタイプの確認

```
%macro add(a,b);
%if (%datatyp(&a)=NUMERIC and %datatyp(&b)=NUMERIC) %then %do;
%put The result is %sysevalf(&a+&b).;
%end;
%else %do;
%put Error: Addition requires numbers.;
%end;
%mend add;
```

ADD マクロを、次のように呼び出すことができます。

```
%add(5.1E2,225)
```

このマクロは、次のメッセージを SAS ログに書き込みます。

```
The result is 735.
```

同様に、次のように ADD マクロを呼び出すことができます。

```
%add(0c1x, 12)
```

このマクロは、次のメッセージを SAS ログに書き込みます。

```
Error: Addition requires numbers.
```

%KVERIFY 自動呼び出しマクロ

式に固有の最初の文字の位置を返します。

カテゴリ: DBCS

種類: NLS 用の自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%KVERIFY(*source*, *excerpt*)

必須引数

source
テキストまたはテキスト式。これは、excerpt に存在しない文字を調べる対象となるテキストです。

excerpt
テキストまたはテキスト式。このテキストは、source を調べるために%KVERIFY が使用する一連の文字を定義します。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。

%KVERIFY は、excerpt に存在しない、source 内の最初の文字の位置を返します。source のすべての文字が excerpt に存在する場合、%KVERIFY は 0 の値を返します。

%LEFT 自動呼び出しマクロと%QLEFT 自動呼び出しマクロ

先頭の空白を削除することによって、引数を左に揃えます。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%LEFT(*text* | *text expression*)

%QLEFT(*text* | *text expression*)

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

LEFT マクロと QLEFT マクロは、どちらも先頭の空白を削除することによって引数を左に揃えます。引数に、次に示す特殊文字またはニーモニック演算子が含まれる場合は、%QLEFT を使用してください。

%LEFT は、引数がクォーティングされている場合でも、クォーティング解除された結果を返します。%QLEFT は、次の特殊文字およびニーモニック演算子をマスクして結果を生成します。そのためマクロプロセッサは、それらをマクロ言語要素ではなく、テキストとして解釈します。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

例: %LEFT と%QLEFT の比較

次の例では、LEFT マクロと QLEFT マクロは、どちらも先頭の空白を削除しています。ただし、QLEFT マクロは、マクロ変数 SYSDAY の先頭の&を、置換されないように保護します。

```
%let d=%nrstr( &sysday );
%put *&d* *%qleft(&d)* *%left(&d)*;
```

%PUT ステートメントは、次の行を SAS ログに書き込みます。

```
* &sysday * *&sysday * *Tuesday *
```

%LOWCASE 自動呼び出しマクロと%QLOWCASE 自動呼び出しマクロ

大文字を小文字に変更します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%LOWCASE (*text* | *text expression*)

%QLOWCASE (*text* | *text expression*)

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%LOWCASE マクロと%QLOWCASE マクロは、大文字のアルファベットを、それらと等価な小文字に変更します。次に示す特殊文字またはニーモニック演算子が引数が含まれる可能性がある場合は、%QLOWCASE を使用してください。

%LOWCASE は、引数に引用符が含まれている場合でも、引用符を除いた結果を返します。%QLOWCASE は、次の特殊文字およびニーモニック演算子をマスクして結果を生成します。そのためマクロプロセッサは、それらをマクロ言語要素ではなく、テキストとして解釈します。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

例: 頭文字を大文字にしたタイトルの作成

```
%macro initcaps(title);
%global newtitle;
%let newtitle=;
%let lastchar=;
%do i=1 %to %length(&title);
%let char=%qsubstr(&title,&i,1);
%if (&lastchar=%str( ) or &i=1) %then %let char=%qupcase(&char);
```

```
%else %let char=%qlowcase(&char);
%let newtitle=&newtitle&char;
%let lastchar=&char;
%end;
TITLE "&newtitle";
%mend;
%initcaps(%str(sales: COMMAND REFERENCE, VERSION 2, SECOND EDITION))
```

この例をサブミットすると、次のステートメントが生成されます。

```
TITLE "Sales: Command Reference, Version 2, Second Edition";
```

%QCMPRES 自動呼び出しマクロ

複数の空白を圧縮し、先頭と末尾の空白を削除し、特殊文字とニーモニック演算子をマスクした結果を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%QCMPRES (*text* | *text expression*)

引数なし

“%CMPRES 自動呼び出しマクロと%QCMPRES 自動呼び出しマクロ” (177 ページ)を参照してください。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%QLEFT 自動呼び出しマクロ

先頭の空白を削除することによって引数を左に揃え、特殊文字とニーモニック演算子をマスクした結果を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%QLEFT (*text* | *text expression*)

引数なし

“%LEFT 自動呼び出しマクロと%QLEFT 自動呼び出しマクロ” (181 ページ)を参照してください。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%QLOWCASE 自動呼び出しマクロ

大文字を小文字に変更し、特殊文字とニーモニック演算子をマスクした結果を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%QLOWCASE(*text* | *text expression*)

引数なし

“%LOWCASE 自動呼び出しマクロと%QLOWCASE 自動呼び出しマクロ” (182 ページ)を参照してください。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%QTRIM 自動呼び出しマクロ

末尾の空白を除去し、特殊文字とニーモニック演算子をマスクした結果を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%QTRIM (*text* | *text expression*)

引数なし

“%TRIM 自動呼び出しマクロと%QTRIM 自動呼び出しマクロ” (190 ページ)を参照してください。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%SYSRC 自動呼び出しマクロ

エラー条件に対応する値を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%SYSRC(*character-string*)

必須引数

character-string
表 13.1 (186 ページ)に示されたニーモニック値のいずれか、またはニーモニック値を生成するテキスト式。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

SYSRC マクロを使用すると、SCL 関数、MODIFY ステートメント、および KEY=オプション付きの SET ステートメントによって生成されたリターンコードをテストできます。SYSRC 自動呼び出しマクロは、エラー条件に関連付けられた数値ではなく、ニーモニック文字列を使用してエラー条件をテストします。

ニーモニック文字列を指定して SYSRC マクロを呼び出すと、SAS のリターンコードが生成されます。ニーモニックは、直感的でなく変更される場合のある数値に比べて、読みやすくなっています。

SCL 関数が返す値と、特定のエラーに対応するニーモニックを指定された SYSRC マクロが返す値を比較することによって、SCL 関数のエラーをテストできます。最後に実行された MODIFY ステートメントまたは KEY=オプション付きの SET ステートメントのエラーをテストするには、_IORC_自動変数の値と、該当するニーモニックの値を指定して呼び出した SYSRC マクロが返す値を比較します。

次の表に、SYSRC 関数で指定するニーモニック値と、それに対応するエラーの説明を示します。

表 13.1 警告条件とエラー条件のニーモニック

ニーモニック	説明
ライブラリの割り当てまたは割り当て解除メッセージ	
_SEDUPLB	ライブラリ参照名が、別のライブラリ参照名と同じ物理ライブラリを参照しています。
_SEIBASN	指定したライブラリ参照名が割り当てられていません。
_SEINUSE	ライブラリまたはメンバを使用できません。
_SEINVLB	ライブラリが、アクセスメソッドに対して有効な形式ではありません。
_SEINVLN	ライブラリ参照名が無効です。
_SELBACC	ライブラリに対して必要なアクセス権限レベルを持っていないため、要求されたアクションを実行できません。
_SELBUSE	ライブラリが使用中です。
_SELGASN	指定したライブラリ参照名が割り当てられていません。
_SENOASN	ライブラリ参照名が割り当てられていません。
_SENOLNM	ライブラリ参照名を使用できません。
_SESEQLB	ライブラリが順次(テープ)形式です。
_SWDUPLB	ライブラリ参照名が、別のライブラリ参照名と同じ物理ファイルを参照しています。
_SWNOLIB	ライブラリが存在しません。
ファイル参照名メッセージ	
_SELOGNM	ファイル参照名が無効なファイルに割り当てられています。
_SWLNASN	ファイル参照名が割り当てられていません。
SAS データセットメッセージ	
_DSENMR	TRANSACTION データセットのオブザベーションが、MASTER データセットに存在しません。
_DSEMTR	複数の TRANSACTION データセットのオブザベーションが、MASTER データセットに存在しません。
_DSENOM	一致するオブザベーションが、MASTER データセットに見つかりませんでした。

ニーモニック	説明
_SEBAUTH	このデータセットはパスワード付きです。
_SEBDIND	インデックス名が無効な SAS 名です。
_SEDSMOD	データセットが、指定した操作に対して正しいモードで開かれていません。
_SEDTLEN	データ長が無効です。
_SEINDCF	新しい名前がインデックス名と競合しています。
_SEINVMD	オープンモードが無効です。
_SEINVPN	物理名が無効です。
_SEMBACC	要求したモードでデータセットを開くために必要なアクセス権限レベルを持っていません。
_SENOLCK	レコードレベルのロックを使用できません。
_SENOMAC	データセットへのメンバレベルのアクセスが拒否されました。
_SENOSAS	ファイルが SAS データセットではありません。
_SEVARCF	新しい名前が既存の変数名と競合しています。
_SWBOF	先頭のオブザベーションを指しているときに、前のオブザベーションを読み込もうとしました。
_SWNOWHR	レコードが WHERE 句を満たしていません。
_SWSEQ	タスクではランダムな順序でオブザベーションを読み込む必要がありますが、使用しているエンジンではシーケンシャルアクセスのみが可能です。
_SWWAUG	WHERE 句が追加されました。
_SWWCLR	WHERE 句がクリアされました。
_SWWREP	WHERE 句が置き換えられました。
SAS ファイルのオープンおよび更新メッセージ	
_SEBDSNM	ファイル名が無効な SAS 名です。
_SEDLREC	レコードがファイルから削除されました。
_SEFOPEN	ファイルが現在開かれています。
_SEINVON	オプション名が無効です。

ニーモニック	説明
_SEINVOV	オプション値が無効です。
_SEINVPS	ファイルデータバッファポインタの値が無効です。
_SELOCK	ファイルが別のユーザーによってロックされています。
_SENOACC	要求したモードでファイルを開くために必要なアクセス権限レベルを持っていません。
_SENOALL	このリリースでは、ファイル名の一部に_ALL_を使用できません。
_SENOCHN	重複が許されないインデックスの値に重複が発生するため、レコードを変更できませんでした。
_SENODEL	このファイルからレコードを削除できません。
_SENODLT	ファイルを削除できませんでした。
_SENOERT	ファイルが書き込み用として開かれていません。
_SENOOAC	要求したオープンモードに対する権限がありません。
_SENOOPN	ファイルまたはディレクトリが開かれていません。
_SENOPF	物理ファイルが存在しません。
_SENORD	ファイルが読み込み用として開かれていません。
_SENORDX	このファイルは基数でアクセスできません。
_SENOTRD	レコードがまだファイルから読み込まれていません。
_SENOUPD	エンジンが読み込み専用のため、ファイルを更新用として開けません。
_SENOWRT	メンバに対する書き込み権限がありません。
_SEOBJLK	ファイルまたはディレクトリが、別のユーザーによって排他的に使用されています。
_SERECRD	レコードが入力ファイルから読み込まれていません。
_SWACMEM	ディレクトリへのアクセスは、一度に 1 つのメンバに提供されます。
_SWDLREC	レコードがファイルから削除されました。
_SWEOF	ファイルの終端。
_SWNOFLE	ファイルが存在しません。

ニーモニック	説明
_SWNOPF	ファイルまたはディレクトリが存在しません。
_SWNOREP	NOREPLACE オプションが指定されているため、ファイルは置き換えられませんでした。
_SWNOTFL	示された項目は存在しますが、ファイルではありません。
_SWNOUPD	このレコードは、この時点では更新できません。
ライブラリ/メンバ/エントリメッセージ	
_SEBDMT	メンバタイプ指定が無効です。
_SEDLT	メンバが削除されませんでした。
_SELKUSR	ライブラリまたはライブラリメンバが、別のユーザーによってロックされています。
_SEMLEN	メンバ名が、このシステム用としては長すぎます。
_SENOLKH	ライブラリまたはライブラリメンバが、現在ロックされていません。
_SENOMEM	メンバが存在しません。
_SWKNXL	まだ存在していないライブラリ、メンバ、またはエントリをロックしました。
_SWLKUSR	ライブラリまたはライブラリメンバが、別のユーザーによってロックされています。
_SWLKYOU	ライブラリまたはライブラリメンバをすでにロックしています。
_SWNOLKH	ライブラリまたはライブラリメンバが、現在ロックされていません。
その他の操作	
_SEDEVOF	デバイスがオフラインであるか、使用できない状態にあります。
_SEDSKFL	ディスクまたはテープの容量に空きがありません。
_SEINVDV	デバイスタイプが無効です。
_SENORNG	書き込み用に開かれたテープに書き込みリングがありません。
_SOK	関数が正常に実行されました。
_SWINVCC	キャリッジコントロール文字が無効です。

ニーモニック	説明
_SWNODSK	デバイスがディスクではありません。
_SWPAUAC	入出力の一時停止。ここまで累積したデータを処理してください。
_SWPAUSL	入出力の一時停止。データウィンドウを手前に表示して、ここまで累積したデータを処理してください。
_SWPAUU1	入出力の一時停止。追加ユーザーコントロールポイント 1 です。
_SWPAUU2	入出力の一時停止。追加ユーザーコントロールポイント 2 です。

比較

SYSRC 自動呼び出しマクロと SYSRC 自動マクロ変数は同じではありません。詳細については、“SYSRC 自動マクロ変数” (216 ページ)を参照してください。

例: _IORC_の値の検査

次の DATA ステップは、自動呼び出しマクロ SYSRC と自動変数_IORC_を使用して SAS ログへのメッセージの書き込みを制御する例を示しています。

```
data big;
modify big trans;
by id;
if _iorc_=%sysrc(_dsenmr) then put 'WARNING: Check ID=' id;
run;
```

%TRIM 自動呼び出しマクロと%QTRIM 自動呼び出しマクロ

末尾の空白を除去します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%TRIM(*text* | *text expression*)

%QTRIM(*text* | *text expression*)

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせ

てください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

TRIM マクロと QTRIM マクロは、どちらも末尾の空白を除去します。次に示す特殊文字またはニーモニック演算子が引数に含まれる場合は、%QTRIM を使用してください。

%QTRIM は、次の特殊文字およびニーモニック演算子をマスクして結果を生成します。そのためマクロプロセッサは、それらをマクロ言語要素ではなく、テキストとして解釈します。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

例

例 1: 末尾の空白の削除

この例では、TRIM 自動呼び出しマクロによって、SAS ログに書き込まれるメッセージから末尾の空白を削除しています。

```
%macro numobs(dsn);
%local num;
data _null_;
set &dsn nobs=count;
call symput('num', left(put(count,8.)));
stop;
run;
%if &num eq 0 %then
%put There were NO observations in %upcase(&dsn).;
%else
%put There were %trim(&num) observations in %upcase(&dsn).;
%mend numobs;
%numobs(sample)
```

NUMOBS マクロを呼び出すと、次のステートメントが生成されます。

```
DATA _NULL_;
SET SAMPLE NOBS=COUNT;
CALL SYMPUT('num', LEFT(PUT(COUNT,8.)));
STOP;
RUN;
```

データセット SAMPLE に 6 つのオブザベーションが含まれている場合、%PUT ステートメントによって次の行が SAS ログに書き込まれます。

```
There were 6 observations in SAMPLE.
```

例 2: %TRIM と%QTRIM の比較

次のステートメントが 1999 年 1 月 28 日に実行されたとします。

```
%let date=%nrstr( &sysdate );
%put *&date* *%qtrim(&date)* *%trim(&date)*;
```

%PUT ステートメントによって次の行が SAS ログに書き込まれます。

```
* &sysdate * * &sysdate* * 28JAN99*
```

%VERIFY 自動呼び出しマクロ

式に固有の最初の文字の位置を返します。

種類: 自動呼び出しマクロ

要件 MAUTOSOURCE システムオプション

構文

%VERIFY(*source*, *excerpt*)

必須引数

source
excerpt に存在しない文字を調べる対象となるテキストまたはテキスト式。

excerpt
テキストまたはテキスト式。このテキストは、*source* を調べるために%VERIFY が使用する一連の文字を定義します。

詳細

注: 自動呼び出しマクロは、SAS が提供するライブラリに含まれています。このライブラリは、サイトにインストールされていないか、サイト固有のバージョンである場合があります。このマクロにアクセスできない場合、またはマクロがサイト固有のバージョンかどうか知りたい場合は、オンサイトの SAS サポート担当者に問い合わせてください。詳細については、“マクロの保存および再利用” (115 ページ)を参照してください。

%VERIFY は、*excerpt* に存在しない、*source* 内の最初の文字の位置を返します。*source* のすべての文字が *excerpt* に存在する場合、%VERIFY は 0 を返します。

例: 有効なファイル参照名のテスト

ISNAME マクロは、文字列をチェックして、それが有効なファイル参照名かどうかを検証し、文字列が有効または無効である理由を説明するメッセージを SAS ログに出力します。

```
%macro isname(name);
%let name=%upcase(&name);
%if %length(&name)>8 %then
%put &name: The fileref must be 8 characters or less.;
%else %do;
%let first=ABCDEFGHIJKLMNOPQRSTUVWXYZ_;
%let all=&first.1234567890;
%let chk_1st=%verify(%substr(&name,1,1),&first);
%let chk_rest=%verify(&name,&all);
%if &chk_rest>0 %then
%put &name: The fileref cannot contain
"%substr(&name,&chk_rest,1)".;
%if &chk_1st>0 %then
%put &name: The first character cannot be
"%substr(&name,1,1)".;
%if (&chk_1st or &chk_rest)=0 %then
```

```
%put &name is a valid fileref.;
%end;
%mend isname;
%isname(file1)
%isname(1file)
%isname(filename1)
%isname(file$)
```

このプログラムが実行されると、次のメッセージが SAS ログに書き込まれます。

```
FILE1 is a valid fileref.
1FILE: The first character cannot be "1".
FILENAME1: The fileref must be 8 characters or less.
FILE$: The fileref cannot contain "$".
```

14 章

自動マクロ変数

自動マクロ変数

自動マクロ変数は、マクロプロセッサによって作成され、さまざまな情報を提供します。これらは、プログラム内でコードを実行する前に、条件のステータスをチェックする場合に役立ちます。

ディクショナリ

SYSADDRBITS 自動マクロ変数

アドレスのビット数が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSADDRBITS 自動マクロ変数には、アドレスに必要なビット数が格納されます。

SYSBUFFR 自動マクロ変数

対応するマクロ変数が存在しない場合に、%INPUT ステートメントに応答して入力されたテキストが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

%INPUT ステートメントが最初に実行されるまで、SYSBUFFR の値は null です。しかし、SYSBUFFR は、%INPUT ステートメントが実行されるたびに、新しい値を受け取ります。その値は、対応するマクロ変数が存在しない場合に%INPUT ステートメントに応答して入力されたテキスト、または null 値のいずれかです。%INPUT ステートメントにマクロ変数名が含まれていない場合、入力された文字はすべて SYSBUFFR に割り当てられます。

例: SYSBUFFR へのテキストの割り当て

次の%INPUT ステートメントは、2 つのマクロ変数、WATRFALL と RIVER の値を受け取ります。

```
%input watrfall river;
```

次のテキストを入力した場合、2 つの変数名とテキストは 1 対 1 で対応しません。

```
Angel Tributary of Caroni
```

たとえば、次のステートメントをサブミットできます。

```
%put WATRFALL contains: *&watrfall*;
%put RIVER contains: *&river*;
%put SYSBUFFR contains: *&sysbuffr*;
```

実行が終わると、次のメッセージが SAS ログに出力されます。

```
WATRFALL contains: *Angel*
RIVER contains: *Tributary*
SYSBUFFR contains: * of Caroni*
```

SAS ログが示すように、SYSBUFFR に格納されたテキストには、先頭の空白と文字間の空白が含まれています。

SYSCC 自動マクロ変数

SAS によって動作環境に返された現在の条件コード(動作環境の条件コード)が格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSCC は、ジョブの条件コードをリセットし、以降のステップの実行を妨げている状態から回復できるようにする、読み込みおよび書き込み用の自動マクロ変数です。

SAS 内部では、正常終了を示す値は 0 です。この内部の値は、各動作環境のホストごとのホストコードによって、意味のある条件コードに変換できます。SAS の終了時における&SYSCC の値 0 は、動作環境のリターンコードの正常値に対応します。

次に、正常条件コードの例を示します。

表 14.1 *SYSCC の動作環境と値*

動作環境	値
z/OS	RC 0
OpenVMS	$STATUS = 1

動作環境のリターンコードを確認する方法は、ホストによって異なります。

SAS の警告条件コードによって&SYSCC は 4 に設定されます。

注: SAS の ERRORCHECK=システムオプションを NORMAL に設定すると、LIBNAME ステートメントや FILENAME ステートメント、あるいは SAS/SHARE ソフトウェアの LOCK ステートメントにエラーがあっても、SYSCC の値は 0 になりま

す。ファイルが存在しないために%INCLUDE ステートメントが失敗した場合でも、SYSCC の値は 0 になります。詳細については、“ERRORCHECK=システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

SYSCHARWIDTH 自動マクロ変数

文字の幅の値が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

文字の幅の値は、1 (半角)または 2 (全角)のいずれかです。

SYSCMD 自動マクロ変数

マクロウィンドウのコマンドラインに入力された、認識されない最後のコマンドが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

%DISPLAY ステートメントが実行されるまでは、SYSCMD の値は null です。マクロウィンドウのコマンドラインに単語または語句を入力し、ウィンドウ環境がそれをコマンドとして認識しなかった場合、SYSCMD は、値としてその単語または語句を受け取ります。この方法は、SYSCMD の値を変更する唯一の方法です。これ以外の場合、SYSCMD は読み込み専用の変数になります。ユーザー作成によるウィンドウコマンドのように動作する値をコマンドラインに入力するには、SYSCMD を使用します。

例: マクロウィンドウに入力したコマンドの処理

マクロ定義 START は、コマンドラインを使用して任意のウィンドウコマンドを入力できるウィンドウを作成します。無効なコマンドを入力すると、そのコマンドが認識されなかったことを示すメッセージが表示されます。コマンドラインに QUIT を入力すると、ウィンドウが閉じてマクロが終了します。

```
%macro start;
%window start
#5 @28 'Welcome to the SAS System'
#10 @28 'Type QUIT to exit';
%let exit = 0;
%do %until (&exit=1);
%display start;
%if &syscmd ne %then %do;
%if %upcase(&syscmd)=QUIT %then %let exit=1;
%else %let sysmsg=&syscmd not recognized;
%end;
%end;
%mend start;
```

SYSDATE 自動マクロ変数

SAS ジョブまたは SAS セッションの実行が開始された日付が格納されます。

種類: 自動マクロ変数(読み込み専用)

参照項目: “SYSDATE9 自動マクロ変数” (199 ページ)

詳細

SYSDATE には、SAS 日付値が DATE7.出力形式で格納されます。この形式は、2 桁の日付、月の名前の最初の 3 文字、および 2 桁の年を表示します。個々のジョブまたはセッションが存続する間、この日付は変わりません。たとえば、あるコードをその月の特定の日に実行したい場合、それを実行する前にプログラムで SYSDATE を使用して日付をチェックできます。

例: SYSDATE の値のフォーマット

次のマクロ FDATE は、指定した出力形式を SYSDATE の値に割り当てています。

```
%macro fdate(fmt);
%global fdate;
data _null_;
call symput("fdate",left(put("&sysdate"d,&fmt)));
run;
%mend fdate;
%fdate(worddate.)
title "Tests for &fdate";
```

このマクロを 1998 年 7 月 28 日に実行した場合、SAS は各ステートメントを次のように解釈します。

```
DATA _NULL_;
CALL SYMPUT("FDATE",LEFT(PUT("28JUL98"D,WORDDATE.)));
RUN;
TITLE "Tests for July 28, 1998";
```

現在の日付をフォーマットする別の方法については、%SYSFUNC 関数および%QSYSFUNC 関数を参照してください。

SYSDATE9 自動マクロ変数

SAS ジョブまたは SAS セッションの実行が開始された日付が格納されます。

種類: 自動マクロ変数(読み込み専用)

参照項目: “SYSDATE 自動マクロ変数” (199 ページ)

詳細

SYSDATE9 には、SAS 日付値が DATE9.出力形式で格納されます。この出力形式は、2 桁の日付、月の名前の最初の 3 文字、および 4 桁の年を表示します。個々のジョブまたはセッションが存続する間、この日付は変わりません。たとえば、あるコードを

その月の特定の日に実行したい場合、それを実行する前にプログラムで SYSDATE9 を使用して日付をチェックできます。

例: SYSDATE9 の値のフォーマット

次のマクロ FDATE は、指定した出力形式を SYSDATE9 の値に割り当てています。

```
%macro fdate(fmt);
b %global fdate;
data _null_;
call symput("fdate",left(put("&sysdate9"d,&fmt)));
run;
%mend fdate;
%fdate(worddate.)
title "Tests for &fdate";
```

このマクロを 2008 年 7 月 28 日に実行した場合、SAS は各ステートメントを次のように解釈します。

```
DATA _NULL_;
CALL SYMPUT("FDATE",LEFT(PUT("28JUL2008"D,WORDDATE.)));
RUN;
TITLE "Tests for July 28, 2008";
```

現在の日付をフォーマットする別の方法については、%SYSFUNC 関数および%QSYSFUNC 関数を参照してください。

SYSDAY 自動マクロ変数

SAS ジョブまたは SAS セッションの実行が開始された曜日が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

あるコードを特定の曜日に実行したい場合、それを実行する前に、SYSDAY を使用して現在の曜日をチェックできます。ただし、SAS セッションをその日に初期化していることが前提です。

例: SAS セッションが開始された曜日の識別

次のステートメントは、SAS セッションの実行が開始されたときの曜日と日付を識別しています。

```
%put This SAS session started running on: &sysday, &sysdate9.;
```

2007 年 12 月 17 日月曜に実行が開始された SAS セッションで、2007 年 12 月 19 日水曜にこのステートメントを実行した場合、次の行が SAS ログに書き込まれます。

```
This SAS session started running on: Monday, 17DEC2007
```

SYSDEVIC 自動マクロ変数

現在のグラフィックデバイスの名前が格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

現在のグラフィックデバイスは、SAS の呼び出しで指定したデバイスです。グラフィックデバイスは、SAS/GRAPH を使用するプロダクトを使用するときに、コマンドラインからプロンプトに入力して指定できます。構成ファイルでグラフィックデバイスを指定することもできます。現在のグラフィックデバイスの名前は、SAS システムオプション DEVICE=の値でもあります。

詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

注: マクロプロセッサは、SYSDEVIC の値を必ずクォーティング解除して格納します。置換された SYSDEVIC の値をクォーティングするには、%SUPERQ マクロクォーティング関数を使用します。

比較

SYSDEVIC への値の割り当て方法は、DEVICE=システムオプションの値を指定する場合と同じです。

SYSDMG 自動マクロ変数

破損したデータセットに対して実行されたアクションを反映するリターンコードが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

デフォルト: 0

詳細

SYSDMG の値を、さらに実行するアクションを決定する条件として使用できます。

SYSDMG には、次の値を格納できます。

表 14.2 *SYSDMG の値と説明*

値	説明
0	このセッションでは、破損したデータセットの修復は発生していません。(デフォルト)
1	破損したデータセットの自動修復が 1 回以上発生しました。
2	ユーザーの要求による破損したデータセットの修復が 1 回以上発生しました。
3	ファイルが破損していたため、ファイルを開くことに 1 回以上失敗しました。
4	データセットが破損していたため、1 つ以上の SAS タスクが終了しました。

値	説明
5	破損したデータセットの自動修復が 1 回以上発生し、最後に修復されたデータセットのインデックスファイルが要求に従って削除されました。
6	ユーザーの要求による修復が 1 回以上発生しました。最後に修復されたデータセットのインデックスファイルが、要求に従って削除されました。

SYSDSN 自動マクロ変数

最後に作成された SAS データセットのライブラリ参照名と名前が格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

参照項目: “SYSLAST 自動マクロ変数” (208 ページ)

詳細

ライブラリ参照名とデータセット名が、2 つの左揃えのフィールドに表示されます。現在のプログラムで SAS データセットが作成されていない場合、SYSDSN は、8 つの空白の後に_NULL_を加え、さらに 2 つの空白を加えた値を返します。

注: マクロプロセッサは、SYSDSN の値を必ずクォーティング解除して格納します。置換された SYSDSN の値をクォーティングするには、%SUPERQ マクロクォーティング関数を使用します。

比較

- SYSDSN への値の割り当て方法は、_LAST_=システムオプションの値を指定する場合と同じです。
- SYSLAST の値は、データセット名の代わりに、その値の参照を直接 SAS コードに挿入できる形式でフォーマットされているため、多くの場合 SYSDSN よりも役立ちます。

例: SYSDSN と SYSLAST によって生成された値の比較

データセット WORK.TEST を作成してから、次のステートメントを入力します。

```
%put Sysdsn produces: *&sysdsn*;
%put Syslast produces: *&syslast*;
```

これらのステートメントを実行すると、次の行が SAS ログに書き込まれます。

```
Sysdsn produces: *WORK TEST *
Syslast produces: *WORK.TEST *
```

ライブラリ参照名またはデータセット名が 8 文字よりも少ない場合、SYSDSN は残りの文字数の空白を追加します。SYSDSN は、ライブラリ参照名フィールドとデータセット名フィールドの間にピリオドを表示しません。

SYSENCODING 自動マクロ変数

SAS セッションエンコーディングの名前が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSENCODING は、最大 12 バイトの長さの名前を表示します。

例: SYSENCODING を使用した SAS セッションエンコーディングの表示

次のステートメントでは、SAS セッションのエンコーディングを表示します。

```
%put The encoding for this SAS session is: &sysencoding;
```

このステートメントを実行すると、次のコメントが SAS ログに書き込まれます。

```
The encoding for this SAS session is: wlatin1
```

SYSENDIAN 自動マクロ変数

現在のセッションのバイトオーダーを示す値が格納されます。取りうる値は LITTLE または BIG のいずれかです。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSENDIAN 自動マクロ変数は、現在の SAS セッションのバイトオーダーを示します。取りうる値 to は LITTLE または BIG のいずれかです。

SYSENV 自動マクロ変数

SAS が対話的に実行されているかどうかをレポートします。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSENV の値は、入力のソースとは無関係です。SYSENV の値は次のとおりです。

FORE
: SAS システムオプション TERMINAL が有効な場合。たとえば、ウィンドウ環境を介して対話的に SAS を実行している場合、この値は FORE になります。

BACK
: SAS システムオプション NOTERMINAL が有効な場合。たとえば、SAS ジョブをバッチモードでサブミットした場合、この値は BACK になります。

対話処理が必要なコードをサブミットする前に、SYSENV を使用して実行モードを確認できます。%INPUT ステートメントを使用するには、SYSENV の値が FORE である必要があります。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

動作環境の情報
一部の動作環境では、バッチモードでのジョブのサブミットはサポートされていません。その場合、SYSENV の値は常に FORE になります。詳細については、使用している動作環境の SAS ドキュメントを参照してください。

SYSERR 自動マクロ変数

一部の SAS プロシジャと DATA ステップによって設定されたリターンコードのステータスが格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSERR の値を条件として使用して、さらにアクションを実行するかどうかを判定したり、実行する SAS プログラムの部分を決定したりできます。SYSERR は、一部のプロシジャや DATA ステップで使用された場合、メモリ不足やコンポーネントシステムの障害などの重大なシステムエラーを検出するために使用されます。SYSERR 自動マクロ変数は、各ステップ境界でリセットされます。完了したジョブのリターンコードについては、“SYSCC 自動マクロ変数” (197 ページ)を参照してください。

SYSERR には、次の値が格納される可能性があります。

表 14.3 *SYSERR の値*

値	説明
0	実行が正常に完了し、警告メッセージもありませんでした。
1	ユーザーによって RUN CANCEL ステートメントが使用され、実行がキャンセルされました。
2	ユーザーによって ATTN コマンドまたは BREAK コマンドが使用され、実行がキャンセルされました。
3	バッチモードまたは非対話型モードで実行されたプログラムのエラーによって、SAS が構文チェックモードになりました。
4	実行は正常に完了しましたが、警告メッセージが発生しました。
5	ユーザーによって ABORT CANCEL ステートメントが使用され、実行がキャンセルされました。
6	ユーザーによって ABORT CANCEL FILE ステートメントが使用され、実行がキャンセルされました。
>6	エラーが発生しました。返される値は、プロシジャによって異なります。

次の表に、警告リターンコードを示します。これらのコードは、具体的な問題を示しません。これらのコードは、問題の性質を識別するためのガイドラインを提供します。

表 14.4 SYSERR の警告コード

警告コード	説明
108	1 つ以上の BY グループでの問題
112	1 つ以上の BY グループでのエラー
116	1 つ以上の BY グループでのメモリの問題
120	1 つ以上の BY グループでの入出力の問題

次の表に、エラーリターンコードを示します。これらのコードは、具体的な問題を示しません。これらのコードは、問題の性質を識別するためのガイドラインを提供します。

表 14.5 SYSERR のエラーコード

エラーコード	説明
1008	一般的なデータの問題
1012	一般的なエラー状態
1016	メモリ不足状態
1020	入出力の問題
2000	セマンティックアクションの問題
2001	属性処理の問題
3000	構文エラー
4000	無効なプロシジャ
9999	プロシジャのバグ
20000	ステップが停止したか、ABORT ステートメントが発行されました。
20001	ABORT RETURN ステートメントが発行されました。
20002	ABORT ABEND ステートメントが発行されました。
25000	重大なシステムエラー。システムを初期化または続行できません。

例: SYSERR の使用

次の例では、エラーメッセージを作成し、%PUT &SYSERR を使用して、リターンコード番号(1012)を SAS ログに書き込んでいます。

```
data NULL;
set doesnotexist;
run;
%put &syserr;
```

次の SAS ログ出力には、リターンコード番号が示されています。

```
75 data NULL;
76 set doesnotexist;
ERROR: File WORK.DOESNOTEXIST.DATA does not exist.
77
78 run;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.NULL might be incomplete. When this step was stopped
there were 0 observations and 0 variables.
WARNING: Data set WORK.NULL was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
real time 0.00 seconds
cpu time 0.00 seconds
79
80 %put &syserr;
1012
```

リターンコード番号の代わりに、エラーと警告のテキストを取得するには、“SYSERRORTEXT 自動マクロ変数” (206 ページ)および“SYSWARNINGTEXT 自動マクロ変数” (223 ページ)を参照してください。

SYSERRORTEXT 自動マクロ変数

SAS ログでの表示用にフォーマットされた最後のエラーメッセージのテキストが格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSERRORTEXT の値は、SAS ログに生成された最終エラーメッセージのテキストです。SYSERR の警告とエラーの一覧については、“SYSERR 自動マクロ変数” (204 ページ)を参照してください。

注: 生成された最終エラーメッセージのテキストに、&または%が含まれていて、%PUT ステートメントを使用する場合、%SUPERQ マクロクォーティング関数を使用して特殊文字をマスクし、値の置換がそれ以上行われないようにする必要があります。次の例では、%PUT ステートメントと%SUPERQ マクロクォーティング関数を使用しています。

```
%put %superq(syserrortext);
```

詳細については、“%SUPERQ 関数” (263 ページ)を参照してください。

例: SYSERRORTEXT の使用

次の例では、エラーメッセージを作成しています。

```
data NULL;
set doesnotexist;
run;
%put &syserrortext;
```

これらのステートメントを実行すると、次のレコードが SAS ログに書き込まれます。

```
1 data NULL;
2 set doesnotexist;
ERROR: File WORK.DOESNOTEXIST.DATA does not exist.
3 run;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.NULL might be incomplete. When this step was
stopped there were 0 observations and 0 variables.
NOTE: DATA statement used (Total process time):
real time 11.16 seconds
cpu time 0.07 seconds
4 %put &syserrortext;
File WORK.DOESNOTEXIST.DATA does not exist.
```

SYSFILRC 自動マクロ変数

最後の FILENAME ステートメントからのリターンコードが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSFILRC は、最後の FILENAME ステートメントによって参照されたファイルまたはストレージの場所が、存在するかどうかをチェックします。外部ファイルへのアクセスを試みる前に、SYSFILRC を使用して、ファイルまたはストレージの場所が割り当てられていることを確認できます。

SYSFILRC の値は次のとおりです。

表 14.6 *SYSFILRC の値と説明*

値	説明
0	最後の FILENAME ステートメントは、正常に実行されました。
≠0	最後の FILENAME ステートメントは、正常に実行されませんでした。

SYSHOSTNAME 自動マクロ変数

コンピュータのホスト名が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSHOSTNAME には、単一の TCP/IP スタックを実行しているシステムのホスト名が格納されます。TCP/IP スタックの詳細については、使用しているホストの SAS ドキュメントを参照してください。

SYSINDEX 自動マクロ変数

現在の SAS ジョブまたは SAS セッションで実行が開始されたマクロの数が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

マクロを使用するプログラム内で、各マクロの呼び出し後に変化する固有の番号が必要な場合、SYSINDEX を使用できます。

SYSINFO 自動マクロ変数

一部の SAS プロシジャによって生成されたリターンコードが格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSINFO の値は、これを使用するプロシジャで生成されています。SYSINFO の値を条件として使用して、さらにアクションを実行するかどうかを判定したり、実行する SAS プログラムの部分を決定したりできます。

たとえば、2 つのデータセットを比較する PROC COMPARE は、比較結果に関する情報を提供する値を格納するために、SYSINFO を使用します。

SYSJOBID 自動マクロ変数

現在のバッチジョブの名前またはユーザー ID が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSJOBID に格納される値は、SAS の実行に使用している動作環境によって異なります。SYSJOBID を使用して、特定の処理を制限するために現在ジョブを実行しているユーザーを確認したり、あるユーザーに固有のコマンドを発行したりできます。

SYSLAST 自動マクロ変数

最後に作成された SAS データファイルの名前が格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

参照項目: “SYSDSN 自動マクロ変数” (202 ページ)

詳細

この名前は、*libref.dataset* の形式で格納されます。データセット名の代わりに、SYSLAST への参照を直接 SAS コードに挿入できます。現在のプログラムで SAS データセットが作成されていない場合、SYSLAST の値は、前後に空白を含まない _NULL_になります。

注: マクロプロセッサは、SYSLAST の値を常にクォーティング解除して格納します。置換された SYSLAST の値をクォーティングするには、%SUPERQ マクロクォーティング関数を使用します。

比較

- SYSLAST に値を割り当てる方法は、_LAST_=システムオプションの値を指定する場合と同じです。
- SYSLAST の値は、データセット名の代わりに、その値の参照を直接 SAS コードに挿入できる形式でフォーマットされているため、多くの場合 SYSDSN よりも役立ちます。

例: SYSLAST と SYSDSN によって生成された値の比較

データセット FIRSTLIB.SALESRPT を作成してから、次のステートメントを入力します。

```
%put Sysdsn produces: *&sysdsn*;
%put Syslast produces: *&syslast*;
```

これらのステートメントを実行すると、次のメッセージが SAS ログに書き込まれます。

```
Sysdsn produces: *FIRSTLIBSALESRPT*
Syslast produces: *FIRSTLIB.SALESRPT*
```

SYSLAST に格納される名前には、ライブラリ参照名とデータセット名の間にピリオドが含まれます。

SYSLCKRC 自動マクロ変数

最後の LOCK ステートメントのリターンコードが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

LOCK ステートメントは、SAS/SHARE ソフトウェアを介してアクセスされるデータライブラリ内のデータオブジェクトに対する、排他ロックの獲得と解放に使用される Base SAS ソフトウェアのステートメントです。SYSLCKRC の値は次のとおりです。

表 14.7 *LCKRC の値と説明*

値	説明
0	最後の LOCK ステートメントは正常に実行されました。

値	説明
>0	最後の LOCK ステートメントは正常に実行されませんでした。
<0	最後の LOCK ステートメントは実行されましたが、WARNING または NOTE が SAS ログに書き込まれました。

詳細については、SAS/SHARE ソフトウェアのドキュメントを参照してください。

SYSLIBRC 自動マクロ変数

最後の LIBNAME ステートメントのリターンコードが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

このコードは、最後の LIBNAME ステートメントが正常に実行されたかどうかをレポートします。SYSLIBRC は、最後の LIBNAME ステートメントによって参照された SAS ライブラリが存在するかどうかをチェックします。たとえば、保存データセットにアクセスする前に、SYSLIBRC を使用して、ライブラリ参照名が割り当てられていることを確認できます。

SYSLIBRC の値は次のとおりです。

***表 14.8** SYSLIBRC の値と説明*

値	説明
0	最後の LIBNAME ステートメントは、正常に実行されました。
≠0	最後の LIBNAME ステートメントは、正常に実行されませんでした。

SYSLOGAPPLNAME 自動マクロ変数

LOGAPPLNAME=システムオプションの値が格納されます。

種類: 自動マクロ変数(読み込み専用)

デフォルト: null

詳細

現在の SAS セッションで次のコードをサブミットすると、現在の SAS セッションの LOGAPPLNAME がログに書き込まれます。

```
%put &syslogapplname;
```

SYSMACRONAME 自動マクロ変数

実行中のマクロの名前を返します。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSMACRONAME は、実行中のマクロの外部で参照されると、ヌル文字列を返します。

SYSMENV 自動マクロ変数

実行中のマクロの起動ステータスが格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSMENV の値は次のとおりです。

表 14.9 *SMENV の値と説明*

値	説明
`S`	実行中のマクロは、SAS プログラムの一部として呼び出されました。
`D`	実行中のマクロは、SAS ウィンドウのコマンドラインから呼び出されました。

SYSMSG 自動マクロ変数

マクロウィンドウのメッセージ領域に表示されるテキストが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSMSG に割り当てる値には、引用符は必要ありません。SYSMSG の値は、%DISPLAY ステートメントのそれぞれの実行後に null に設定されます。

例: %DISPLAY ステートメント

次の例は、SYSMSGT に割り当てられたテキストが、%DISPLAY ステートメントの実行後にクリアされることを示しています。

```
%let sysmsg=Press ENTER to continue.;
```

```
%window start
#5 @28 'Welcome to SAS';
%display start;
%put Sysmsg is: *&sysmsg*;
```

このプログラムを実行すると、次のメッセージが SAS ログに書き込まれます。

```
Sysmsg is: **
```

SYSNCPU 自動マクロ変数

計算に使用できる現在のプロセッサの数が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSNCPU は、CPUCOUNT オプションの現在の値を提供する自動マクロ変数です。詳細については、“CPUCOUNT=システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

比較

次の例では、CPUCOUNT オプションに 265 を設定しています。

```
options cpucount=265;
%put &sysncpu;
```

前述の例の出力は、265 になります。

SYSNOBS 自動マクロ変数

前のプロシジャまたは DATA ステップによって閉じられた最後のデータセットから読み込まれた、オブザベーションの数が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSNOBS には、前のプロシジャまたは DATA ステップによって閉じられた最後のデータセットから読み込まれたオブザベーションの数が格納されます。

注: データセットのオブザベーションの数が、前のプロシジャまたは DATA ステップによって計算されていない場合、SYSNOBS の値は-1 に設定されます。

SYSODSESCAPECHAR 自動マクロ変数

プログラム内の ODS ESCAPECHAR=の値を表示します。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSODSESCAPECHAR 自動マクロ変数には、現在の ODS のエスケープ文字が格納されます。

SYSODSPATH 自動マクロ変数

現在の Output Delivery System (ODS)のパス名が格納されます。

種類: 自動マクロ変数(読み込み専用)

制限事項: SYSODSPATH 自動マクロ変数は、ODS または PROC TEMPLATE ステートメントが起動された場合にのみ存在します。

詳細

SYSODSPATH 自動マクロ変数には、現在の ODS のパス名が格納されます。

SYSPARM 自動マクロ変数

動作環境から SAS プログラムステップに渡すことのできる文字列が格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSPARM を使うと、動作環境から SAS プログラムステップに文字列を渡すことができます。これによって、プログラムの実行中に文字列にアクセスしたり、文字列を使用したりする手段が提供されます。たとえば、プログラムで処理されるタイトルステートメントまたは値を、SYSPARM を使用して動作環境から渡すことができます。SAS プログラム内で、SYSPARM の値を設定することもできます。SYSPARM は、SAS プログラム内の任意の場所で使用できます。SYSPARM のデフォルト値は、null (値 0 の文字)です。

SYSPARM は、SAS の起動時に指定した場合に最も役立ちます。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

注: マクロプロセッサは、SYSPARM の値を常にクォーティング解除して格納します。置換された SYSPARM の値をクォーティングするには、%SUPERQ マクロクォーティング関数を使用します。

比較

- SYSPARM に値を割り当てる方法は、SYSPARM=システムオプションの値を指定する場合と同じです。
- SYSPARM の値を取得する方法は、SYSPARM() SAS 関数を使用する場合と同じです。

例: プロシジャに値を渡す

この例では、UNIX 動作環境で次のようなコマンドを使用して、2011 年 9 月 20 日に SAS を起動します(ライブラリ参照名 DEPT および TEST は、config.sas ファイル内で定義されています)。

```
sas program-name -sysparm dept.projects -config /myid/config.sas
```

次に示すように、マクロ変数 SYSPARM によって PROC REPORT のデータセット名を指定します。

```
proc report data=&sysparm
report=test.resorces.priority.rept;
title "%sysfunc(date(),worddate.)";
title2;
title3 'Active Projects By Priority';
run;
```

このマクロを実行すると、次の SAS ステートメントが生成されます。

```
proc report data=dept.projects
report=test.resorces.priority.rept;
title "September 20, 2011";
title2;
title3 'Active Projects By Priority';
run;
```

SYSPBUFF 自動マクロ変数

マクロパラメータ値として指定されたテキストが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSPBUFF は、PARMBUFF オプションを使用して定義されたマクロの呼び出しにおいて、パラメータ値で指定されたテキストに置換されます。ネームスタイル呼び出しの場合、このテキストには、かっことカンマが含まれます。PARMBUFF オプションと SYSPBUFF を使用して、呼び出しごとに個数が変わるパラメータを受け取るマクロを定義できます。

マクロ定義に一連のパラメータと PARMBUFF オプションの両方が含まれている場合、このマクロを呼び出すと、値がパラメータで受け取られ、値の呼び出しリスト全体が SYSPBUFF に割り当てられます。

例: SYSPBUFF を使用したマクロパラメータ値の表示

マクロ PRINTZ は、PARMBUFF オプションを使用して個数が変わるパラメータを定義し、SYSPBUFF を使用して呼び出し時に指定されたパラメータを表示します。

```
%macro printz/parmbuff;
%put Syspbuff contains: &syspbuff;
%let num=1;
%let dsname=%scan(&syspbuff,&num);
%do %while(&dsname ne);
proc print data=&dsname;
run;
%let num=%eval(&num+1);
%let dsname=%scan(&syspbuff,&num);
%end;
%mend printz;
%printz(purple,red,blue,teal)
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
Syspbuff contains: (purple,red,blue,teal)
```

SYSPROCESSID 自動マクロ変数

現在の SAS プロセスのプロセス ID が格納されます。

種類: 自動マクロ変数(読み込み専用)

デフォルト: null

詳細

プロセス ID は、32 文字の 16 進文字列です。デフォルト値は null です。

例: SYSPROCESSID を使用した SAS の現在のプロセス ID の表示

次のコードでは、SAS の現在のプロセス ID を SAS ログに書き込んでいます。

```
%put &sysprocessid;
```

次に示すようなプロセス ID が SAS ログに書き込まれます。

```
41D1B269F86C7C5F4010000000000000
```

SYSPROCESSNAME 自動マクロ変数

現在の SAS プロセスのプロセス名が格納されます。

種類: 自動マクロ変数(読み込み専用)

例: SYSPROCESSNAME を使用して、現在の SAS プロセス名を表示する

次のステートメントでは、現在の SAS プロセス名を SAS ログに書き込んでいます。

```
%put &sysprocessname;
```

2つ目の SAS セッションの SAS ウィンドウ環境でこのステートメントをサブミットすると、次の行が SAS ログに書き込まれます。

```
DMS Process (2)
```

SYSPROCNAME 自動マクロ変数

SAS 言語プロセッサによって現在処理されているプロシジャ(または DATA ステップの DATASTEP)の名前が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSPROCNAME には、ステップ境界に達するまでにユーザーによって PROC ステートメントで指定されたプロシジャ名が格納されます。

SYSRC 自動マクロ変数

オペレーティングシステムによって最後に生成されたリターンコードが格納されます。

種類: 自動マクロ変数(読み込みおよび書き込み)

詳細

SYSRC から返されるコードは、オープンコード内の X ステートメント、ウィンドウ環境の X コマンド、または%SYSEXEC、%TSO、%CMS のいずれかのマクロステートメントを使用して実行したコマンドに基づきます。リターンコードは整数です。SYSRC のデフォルト値は 0 です。

ジョブを続行する前に、SYSRC を使用してシステムコマンドのリターンコードを確認できます。リターンコードの例については、使用している動作環境の SAS ドキュメントを参照してください。

SYSSCP 自動マクロ変数と SYSSCPL 自動マクロ変数

動作環境の ID が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSSCP と SYSSCPL は、使用している動作環境の名前の省略形に置換されます。SYSSCPL は、SYSSCP よりも詳細な値を提供する場合があります。適切なシステムコマンドを実行するために、SYSSCP と SYSSCPL を使用して動作環境を確認できます。

次の表に、SYSSCP と SYSSCPL の値を示します。

表 14.10 *SAS 9.2 以上を実行しているプラットフォームの SYSSCP と SYSSCPL の値*

プラットフォーム	SYSSCP の値	SYSSCPL の値
z/OS	OS	z/OS
Itanium 上の VMI または OpenVMS(Foundation SAS でのみサポートされます)	VMS ITAN	OpenVMS
	UNIX	
HP-UX PA-RISC または H64	HP 64	HP-UX

プラットフォーム	SYSSCP の値	SYSSCPL の値
Itanium 上の H61、HP-UX IPF、または HP-UX	HP IPF	HP-UX
X64(x86-64)上の LAX または LINUX	LIN X64	LINUX
LNX、LINUX、または LINUX 32 ビット(x86)	LINUX	LINUX
POWER 上の R64、AIX64、または AIX	AIX 64	AIX
SPARC 上の S64、SUN64、または Solaris	SUN 64	SUNOS または SunOS
X64(x86-64)上の SAX または Solaris 10	SUN X64	SUNOS
	Windows	
Windows XP Pro	WIN	XP_PRO
Windows Server 2003	WIN	NET_SRV
Windows Enterprise Server 2003	WIN	NET_ASRV
Windows Data Center Server 2003	WIN	NET_DSRV
Windows XP Pro x64	WIN	X64_PRO
Windows Server 2003 x64	WIN	X64_SRV
Windows Enterprise Server 2003 x64	WIN	X64_ESRV
Windows Data Center Server 2003 x64	WIN	X64_DSRV
Windows Vista Business	WIN	W32_VSPRO
Windows Server 2008	WIN	W32_SRV08
Windows Enterprise Server 2008	WIN	W32_ESRV08
Windows Data Center Server 2008	WIN	W32_DSRV08
Windows Vista Business x64	WIN	X64_VSPRO

プラットフォーム	SYSSCP の値	SYSSCPL の値
Windows Server 2008 x64	WIN	X64_SRV08
Windows Enterprise Server 2008 x64	WIN	X64_ESRV08
Windows Data Center Server 2008 x64	WIN	X64_DSRV08
Windows Server 2008 Itanium	WIN	W64_ESRV08
Windows Itanium Enterprise Server 2003 または W64_ASRV	WIN	W64_ASRV
Windows Itanium Data Center Server 2003 または W64_DSRV	WIN	W64_DSRV
Windows Itanium Server 2003	WIN	W64_SRV

例: SAS の実行プラットフォームの一時ファイルの削除

マクロ DELFILE は、SAS を実行しているプラットフォームを検出し、TMP ファイルを削除します。FILEREF は、TMP ファイルのファイル参照名が格納されたグローバルマクロ変数です。

```
%macro delfile;
%if /* HP Unix */&sysscp=HP 800 or &sysscp=HP 300
%then
%do;
X "rm &fileref..TMP";
%end;
%else %if /* DOS-LIKE PLATFORMS */&sysscp=OS2 or &sysscp=WIN
%then
%do;
X "DEL &fileref..TMP";
%end;
%else %if /* CMS */&sysscp=CMS
%then
%do;
X "ERASE &fileref TEMP A";
%end;
%mend delfile;
```

SYSSCPL 自動マクロ変数

動作環境の名前が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

“SYSSCP 自動マクロ変数と SYSSCPL 自動マクロ変数” (216 ページ)を参照してください。

SYSSITE 自動マクロ変数

サイトに割り当てられた番号が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SAS ソフトウェアのライセンスを取得した各サイトには、SAS によってサイト番号が割り当てられます。この番号は、SAS ログに表示されます。

SYSSIZEOFLONG 自動マクロ変数

現在のセッションでのロング整数の長さ(バイト単位)が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSSIZEOFLONG 自動マクロ変数には、現在の SAS セッションでのロング整数の長さが格納されます。

SYSSIZEOFPTR 自動マクロ変数

ポインタのサイズ(バイト単位)が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSSIZEOFPTR 自動マクロ変数には、ポインタのサイズ(バイト単位)が格納されます。

SYSSIZEOFUNICODE 自動マクロ変数

現在のセッションでのユニコード文字の長さ(バイト単位)が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSSIZEOFUNICODE 自動マクロ変数には、現在の SAS セッションでのユニコード文字の長さが格納されます。

SYSSTARTID 自動マクロ変数

最後の STARTSAS ステートメントから生成された ID が格納されます(評価版)。

種類: 自動マクロ変数(読み込み専用)

デフォルト: null

注: STARTSAS ステートメントは、SAS システムの評価版の機能です。

詳細

この ID は、WAITSAS ステートメントまたは ENDSAS ステートメントに渡すことができる 32 文字の 16 進文字列です。デフォルト値は null です。

例: SYSSTARTID を使用した最後の STARTSAS ステートメントの SAS プロセス ID の表示(評価版)

最後の STARTSAS ステートメントをサブミットした SAS プロセスから、次のコードをサブミットして、SYSSTARTID 変数の値を SAS ログに書き込みます。

```
%put &sysstartid
```

次のようなプロセス ID の値が SAS ログに書き込まれます。

```
41D20425B89FCED94036000000000000
```

SYSSTARTNAME 自動マクロ変数

最後の STARTSAS ステートメントから生成されたプロセス名が格納されます(評価版)。

種類: 自動マクロ変数(読み込み専用)

デフォルト: null

注: STARTSAS ステートメントは、SAS システムの評価版の機能です。

例: SYSSTARTNAME を使用して、最新の STARTSAS ステートメントから SAS プロセス名を表示する(評価版)

最後の STARTSAS ステートメントをサブミットした SAS プロセスから、次のコードをサブミットして、SYSSTARTNAME 変数の値を SAS ログに書き込みます。

```
%put &sysstartname;
```

次に示す例のようなプロセス名が SAS ログに表示されます。

```
DMS Process (2)
```

SYSTCPIPHOSTNAME 自動マクロ変数

複数の TCP/IP スタックがサポートされている場合、ローカルコンピュータとリモートコンピュータのホスト名が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSTCPIPHOSTNAME には、複数の TCP/IP スタックを実行しているシステムのホスト名が格納されます。TCP/IP スタックの詳細については、使用しているホストの SAS ドキュメントを参照してください。

SYSTIME 自動マクロ変数

SAS ジョブまたは SAS セッションの実行が開始された時刻が格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

この値は、TIME5.形式で表示され、個々のジョブまたはセッションが実行されている間は変わりません。

例: SYSTIME を使用した SAS セッションの開始時刻の表示

次のステートメントは、SAS セッションの開始時刻を表示します。

```
%put This SAS session started running at: &systime;
```

SAS セッションの実行が午前 9 時 30 分に開始されている場合に、このステートメントを午後 3 時に実行すると、次のコメントが SAS ログに書き込まれます。

```
This SAS session started running at: 09:30
```

SYSUSERID 自動マクロ変数

現在の SAS プロセスのユーザー ID またはログインが格納されます。

種類: 自動マクロ変数(読み込み専用)

例: SYSUSERID を使用して、現在の SAS プロセスのユーザー ID を表示する

現在の SAS プロセスで次のコードをサブミットすると、現在の SAS プロセスのユーザー ID またはログインが SAS ログに書き込まれます。

```
%put &sysuserid;
```

次のようなユーザー ID が SAS ログに書き込まれます。

MyUserid

SYSVER 自動マクロ変数

実行中の SAS ソフトウェアのリリース番号が格納されます。

種類: 自動マクロ変数(読み込み専用)

参照項目: “SYSVLONG 自動マクロ変数” (222 ページ)および“SYSVLONG4 自動マクロ変数” (223 ページ)

比較

SYSVER は、実行中の SAS ソフトウェアのリリース番号を提供します。新しい機能でジョブを実行する前に、SYSVER を使用して SAS のリリースを確認できます。

例: SAS ソフトウェアのリリースの識別

次のステートメントは、ユーザーの SAS ソフトウェアのリリース番号を表示します。

```
%put I am using release: &sysver;
```

このステートメントをサブミットすると、SAS 9.2 のユーザーの場合、次の出力が SAS ログに書き込まれます。

```
I am using release: 9.2
```

SYSVLONG 自動マクロ変数

実行中の SAS ソフトウェアのリリース番号とメンテナンスレベルが格納されます。

種類: 自動マクロ変数(読み込み専用)

参照項目: “SYSVER 自動マクロ変数” (222 ページ)および“SYSVLONG4 自動マクロ変数” (223 ページ)

比較

SYSVLONG は、SAS ソフトウェアのリリース番号とメンテナンスレベルを提供します。

例: SAS のメンテナンスリリースの識別

次のステートメントは、使用中の SAS リリースを識別する情報を表示します。

```
%put I am using release: &sysvlong;
```

このステートメントをサブミットすると、SAS 9.2 のユーザーの場合、次の出力が SAS ログに書き込まれます。

```
I am using release: 9.02.02M2D071609
```

SYSVLONG4 自動マクロ変数

実行中の SAS ソフトウェアのリリース番号とメンテナンスレベル、および 4 桁の年が格納されます。

種類: 自動マクロ変数(読み込み専用)

参照項目: “SYSVER 自動マクロ変数” (222 ページ)および“SYSVLONG 自動マクロ変数” (222 ページ)

比較

SYSVLONG4 は、4 桁の年、および SAS ソフトウェアのリリース番号とメンテナンスレベルを提供します。SYSVLONG4 は、4 桁の年を含むこと以外、SYSVLONG と同じです。

例: SYSVLONG4 自動マクロ変数の使用

次のステートメントは、使用中の SAS リリースを識別する情報を表示します。

```
%put I am using maintenance release: &sysvlong4;
```

このステートメントをサブミットすると、SAS 9.2 のユーザーの場合、次のコメントが SAS ログに書き込まれます。

```
I am using maintenance release: 9.02.01B0D09112007
```

SYSWARNINGTEXT 自動マクロ変数

SAS ログでの表示用にフォーマットされた最終警告メッセージのテキストが格納されます。

種類: 自動マクロ変数(読み込み専用)

詳細

SYSWARNINGTEXT の値は、SAS ログに生成された最終警告メッセージのテキストです。SYSERR の警告とエラーの一覧については、“SYSERR 自動マクロ変数” (204 ページ)を参照してください。

注: 生成された最終警告メッセージのテキストに、&または%が含まれていて、%PUT ステートメントを使用する場合、%SUPERQ マクロクォーティング関数を使用して特殊文字をマスクし、値の置換がそれ以上行われないようにする必要があります。次の例では、%PUT ステートメントと%SUPERQ マクロクォーティング関数を使用しています。

```
%put %superq(syswarningtext);
```

詳細については、“%SUPERQ 関数” (263 ページ)を参照してください。

例: SYSWARNINGTEXT の使用

次の例では、警告メッセージを作成します。

```
data NULL;
set doesnotexist;
```

```
run;
%put &syswarningtext;
```

これらのステートメントを実行すると、次のコメントが SAS ログに書き込まれます。

```
 1 data NULL;
2 set doesnotexist;
ERROR: File WORK.DOESNOTEXIST.DATA does not exist.
3 run;
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.NULL might be incomplete. When this step
was stopped there were 0 observations and 0 variables.
NOTE: DATA statement used (Total process time):
real time 11.16 seconds
cpu time 0.07 seconds
4 %put &syswarningtext;
The data set WORK.NULL might be incomplete. When this step was
stopped there were 0 observations and 0 variables.
```

15 章

マクロの DATA ステップ CALL ルーチン

マクロの DATA ステップ CALL ルーチン

DATA ステップ CALL ルーチンを使用して、マクロ機能を操作できます。

ディクショナリ

CALL EXECUTE ルーチン

引数を置換し、次のステップ境界で実行するための置換した値を発行します。

種類: DATA ステップ CALL ルーチン

構文

CALL EXECUTE (*argument*);

必須引数

argument
次のいずれかを指定できます。

- 引用符で囲んだ文字列。*一重引用符で囲んだ引数*は、プログラムの実行時に置換されます。*二重引用符で囲んだ引数*は、DATA ステップが作成される際に置換されます。たとえば、マクロ SALES を呼び出すには、次のコードを使用します。

```
call execute('%sales');
```

- 生成されるテキスト式または SAS ステートメントの値を持つ、DATA ステップ文字変数の名前。DATA ステップ変数の名前を引用符で囲まないでください。たとえば、SAS ステートメントまたはテキスト式が格納された DATA ステップ変数 FINDOBS の値を使用するには、次のコードを使用します。

  ```
  call execute(findobs);
  ```

- DATA ステップによってマクロテキスト式または SAS ステートメントに置換される文字式。たとえば、変数 MONTH の値をパラメータとして渡すマクロ呼び出しを生成するには、次のコードを使用します。

  ```
  call execute('%sales('||month||')');
  ```

詳細

EXECUTE ルーチンの引数がマクロ呼び出しまたはマクロ呼び出しへの置換である場合、即座にマクロが実行されます。マクロの実行によって生成された SAS ステートメントは、ステップ境界に達するまで実行されません。マクロ変数参照などの SAS マクロステートメントは、即座に実行されます。

注: ステップ境界に達するまで SAS ステートメントが実行されないため、SAS マクロステートメント内の、SAS ステートメントによって作成または更新されるマクロ変数への参照は、正しく置換されません。

注: マクロ参照は即座に実行されますが、SAS ステートメントはステップ境界に達するまで実行されません。そのため、あるマクロにマクロ変数の参照が含まれており、そのマクロ変数が同じマクロ内で CALL SYMPUT によって作成されたものである場合、CALL EXECUTE を使用してそのマクロを呼び出すことはできません。これを回避する方法については、次のヒントを参照してください。

ヒント 次の例では、%NRSTR マクロクォーティング関数を使用してマクロステートメントをマスクしています。この関数は、マクロステートメントの実行を、ステップ境界に達するまで遅らせます。

```
call execute('%nrstr(%sales('||month||'))');
```

比較

マクロ機能の他の要素とは異なり、CALL EXECUTE ステートメントは、SAS システムオプション MACRO または NOMACRO の設定とは無関係に使用できます。どちらに設定しても、EXECUTE は、引数の値をプログラムスタックに配置します。ただし、NOMACRO に設定すると、引数に含まれるマクロ呼び出しまたはマクロ関数は置換されません。

例

例 1: マクロの条件付き実行

次の DATA ステップでは、CALL EXECUTE を使用してマクロを実行しています。このマクロは、DATA ステップが少なくとも 1 つのオブザベーションを一時データセットに書き込む場合にのみ実行されます。

```
%macro overdue;
proc print data=late;
title "Overdue Accounts As of &sysdate";
run;
%mend overdue;
data late;
set sasuser.billed end=final;
```

```
if datedue<=today()-30 then
do;
n+1;
output;
end;
if final and n then call execute('%overdue');
run;
```

例 2: パラメータリストに DATA ステップ値を渡す

CALL EXECUTE を使用して、DATES データセットの DATE 変数の値をマクロ REPT の DAT パラメータに渡し、REPTDATA データセットの VAR1 変数の値を A パラメータに渡し、REPTDATA を DSN パラメータに渡しています。DATA _NULL_ステップの終了後、DATES データセット内の 3 つの日付に対応する、3 つの PROC GCHART ステートメントがサブミットされます。

```
data dates;
input date $;
datalines;
10nov11
11nov11
12nov11
;
data reptdata;
input date $ var1 var2;
datalines;
10nov11 25 10
10nov11 50 11
11nov11 23 10
11nov11 30 29
12nov11 33 44
12nov11 75 86
;
%macro rept(dat,a,dsn);
proc chart data=&dsn;
title "Chart for &dat";
where(date="&dat");
vbar &a;
run;
%mend rept;
data _null_;
set dates;
call execute('%rept('||date||','||'var1,reptdata)');
run;
```

CALL SYMDEL ルーチン

指定された変数を、マクロのグローバルシンボルテーブルから削除します。

種類: DATA ステップ CALL ルーチン

構文

CALL SYMDEL(*macro-variable*<, *option*>);

必須引数

macro-variable
次のいずれかを指定できます。

- 引用符で囲み、アンパサンドを付けないマクロ変数の名前。マクロ変数値に別のマクロ変数参照が含まれる場合、SYMDEL はその参照の置換を試みません。
- DATA ステップ文字変数の名前。引用符を付けずに指定し、マクロ変数名を格納します。この値が有効な SAS 名でない場合、またはマクロプロセッサがこの名前のマクロ変数を検出できなかった場合、警告メッセージがログに書き込まれます。
- マクロ変数名を作成する文字式。

option(s)

NOWARN
存在しないマクロ変数を削除しようとした場合に、警告メッセージが発行されないようにします。NOWARN を引用符で囲む必要があります。

詳細

CALL SYMDEL は、存在しないマクロ変数を削除しようとした場合、警告メッセージを発行します。このような警告メッセージが発行されないようにするには、NOWARN オプションを指定します。

CALL SYMPUT ルーチン

DATA ステップで生成された値を、マクロ変数に割り当てます。

種類: DATA ステップ CALL ルーチン

参照項目: “SYMGET 関数” (238 ページ)および“CALL SYMPUTX ルーチン” (233 ページ)

構文

CALL SYMPUT(*macro-variable*, *value*);

必須引数

macro-variable
次のいずれかの項目を指定できます。

- SAS 名を表す、引用符で囲んだ文字列。たとえば、文字列 **testing** をマクロ変数 NEW に割り当てるには、次のステートメントをサブミットします。

```
call symput('new','testing');
```

- SAS 名の値を持つ文字変数の名前。たとえば、次の DATA ステップでは、3 つのマクロ変数 SHORTSTP、PITCHER、および FRSTBASE を作成し、それらに、ANN、TOM、および BILL という値をそれぞれ割り当てています。

```
data team1;
input position : $8. player : $12.;
call symput(position,player);
datalines;
shortstp Ann
pitcher Tom
```

```
frstbase Bill
;
```

- マクロ変数名を生成する文字式。この形式は、一連のマクロ変数を作成する場合に役立ちます。たとえば、次の CALL SYMPUT ステートメントでは、文字列 POS と左に揃えられた_N_の値を結合して一連のマクロ変数名を作成し、作成したマクロ変数 POS1、POS2、および POS3 に値を割り当てています。

```
data team2;
input position : $12. player $12.;
call symput('POS'||left(_n_), position);
datalines;
shortstp Ann
pitcher Tom
frstbase Bill
;
```

value
割り当てる値。次のいずれかを指定できます。

- 引用符で囲んだ文字列。たとえば、次のステートメントでは、文字列 **testing** をマクロ変数 NEW に割り当てています。

```
call symput('new','testing');
```

- 数値変数または文字変数の名前。この変数の現在の値が、マクロ変数の値として割り当てられます。この変数が数値の場合、自動的に数値が文字列に変換されて、ログにメッセージが書き込まれます。この後のセクションで、DATA ステップ変数の文字値と数値をマクロ変数に割り当てる際に SYMPUT が従うフォーマット規則について説明します。

  注: この形式は、*macro-variable* が SAS 変数の名前、または SAS 変数を含む文字式でもある場合に最も役立ちます。データセット TEAM1 を作成する前述の例で示したように、各オブザベーションから固有のマクロ変数名と値を作成できます。

  macro-variable が文字列の場合、SYMPUT によって 1 つのマクロ変数のみが作成され、その値はプログラム内での反復ごとに変化します。プログラムの実行終了後に残る値は、最後の反復で割り当てられた値のみです。

- DATA ステップの式。現在のオブザベーションにおいて式が返す値が、*macro-variable* の値として割り当てられます。次の例では、HOLDATE というマクロ変数に **July 4,1997** という値が割り当てられています。

```
data c;
input holiday mmddyy.;
call symput('holdate',trim(left(put(holiday,worddate.))));
datalines;
070497
;
run;
```

  この式が数値の場合、自動的に数値が文字列に変換されて、ログにメッセージが書き込まれます。この後のセクションで、式の文字値と数値をマクロ変数に割り当てる際に SYMPUT が従うフォーマット規則について説明します。

詳細

macro-variable が存在しない場合、SYMPUT によってその変数が生成されます。SYMPUT は、プログラムの実行時にマクロ変数を割り当てます。

SYMPUT は、SCL プログラムなどのすべての SAS 言語プログラムで使用できます。SYMPUT は、マクロの実行時ではなくプログラムの実行時に変数を置換するため、DATA ステップビュー、SQL ビュー、および SCL プログラムのマクロ変数値の割り当てに使用する必要があります。

SYMPUT を使用して作成された変数のスコープ

SYMPUT は、マクロ変数を、最もローカルな空でないシンボルテーブルに格納します。シンボルテーブルは、次のものを含んでいる場合、空ではありません。

- 値
- 計算される%GOTO(計算される%GOTO は%または&を含み、ラベルに置換されます)
- マクロの呼び出し時に作成されるマクロ変数&SYSPBUFF

ただし、ローカルシンボルテーブルが空の場合でも、SYMPUT によってそのテーブルに変数が作成される、次の 3 つのケースがあります。

- SAS バージョン 8 以降、PROC SQL の後で SYMPUT を使用すると、ローカルシンボルテーブルに変数が作成されます。
- 実行中のマクロに計算される%GOTO ステートメントが含まれる場合、SYMPUT を使用してマクロ変数を作成すると、ローカルシンボルテーブルに変数が作成されます。
- 実行中のマクロが&SYSPBUFF と SYMPUT を使用してマクロ変数を作成すると、ローカルシンボルテーブルにマクロ変数が作成されます。

SYMPUT を使用した変数の作成の詳細については、“マクロ変数のスコープ” (45 ページ)を参照してください。

SYMPUT によって割り当てられる値が使用可能になる前に、それを参照しようとする場合の問題

SYMPUT を使用する場合に最もよく発生する問題の 1 つは、SYMPUT によって割り当てられるマクロ変数値を、その変数が作成される前に参照しようとすることです。通常、この問題は、変数の値を割り当てる CALL SYMPUT ステートメントの実行が開始される前に、マクロ変数を参照しているステートメントがコンパイルされるために発生します。SYMPUT を使用する場合に覚えておくべき最も重要なことは、マクロ変数の値がプログラムの実行時に割り当てられるということです。マクロ変数参照は、ステップ、ステップの外部で使用されるグローバルステートメント、または SCL プログラムのコンパイル時に置換されます。その結果、次のようになります。

- プログラム内で SYMPUT を使用してマクロ変数を作成し、それに値を割り当てた場合、同じプログラム(またはステップ)内でマクロ変数参照を使用してそのマクロ変数の値を取得することはできません。
- プログラムの後ろのグローバルステートメント(たとえば、TITLE ステートメント)で値を参照する前に、ステップ境界ステートメントを指定して、強制的に DATA ステップを実行する必要があります。指定できる境界は、RUN ステートメント、あるいは別の DATA ステートメントまたは PROC ステートメントです。次に例を示します。

```
data x;
x='December';
call symput('var',x);
proc print;
title "Report for &var";
run;
```

マクロ処理 (35 ページ) では、コンパイルと実行の詳細について説明されています。

文字値の割り当てのフォーマット規則

value が文字変数の場合、SYMPUT は$*w*.形式を使用して値を書き込みます。ここで、*w* は変数の長さです。そのため、値がプログラム変数の長さよりも短い場合、末尾に空白が挿入されて書き込まれます。たとえば、次に示す DATA ステップでは、変数 C の長さはデフォルトで 8 になります。したがって、SYMPUT は$8.形式を使用し、文字 **x** の後ろに 7 つの空白を加え、それを CHAR1 の値として割り当てます。これらの空白を除去するには、2 番目の SYMPUT ステートメントに示すように、TRIM 関数を使用します。

```
data char1;
input c $;
call symput('char1',c);
call symput('char2',trim(c));
datalines;
x
;
run;
%put char1 = ***&char1***;
%put char2 = ***&char2***;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
char1 = ***x ***
char2 = ***x***
```

数値の割り当てのフォーマット規則

*値*が数値変数の場合、SYMPUT は、BEST12.形式を使用してそれを書き込みます。その結果、右に揃えられた 12 バイトの文字列の値が得られます。たとえば、次に示す DATA ステップでは、数値変数 X の値をマクロ変数 NUM1 と NUM2 に割り当てています。最後の CALL SYMPUT ステートメントでは、LEFT 関数を使用して値を左に揃え、不要な末尾の空白を削除してから、SYMPUT ルーチンによってその値を NUM3 に割り当てています。

```
data _null_;
x=1;
call symput('num1',x);
call symput('num2',left(x));
call symput('num3',trim(left(put(x,8.)))); /*preferred technique*/
run;
%put num1 = ***&num1***;
%put num2 = ***&num2***;
%put num3 = ***&num3***;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
num1 = *** 1***
num2 = ***1 ***
num3 = ***1***
```

比較

- SYMPUT は、プログラムの実行時に、DATA ステップで生成された値をマクロ変数に割り当てます。一方、SYMGET 関数は、プログラムの実行時に、マクロ変数の値をプログラムに返します。
- SYMPUT は、DATA ステップ内および SCL プログラム内で使用できます。一方、SYMPUTN は、SCL プログラム内でのみ使用できます。
- SYMPUT は文字値を割り当てます。一方、SYMPUTN は数値を割り当てます。

例: マクロ変数を作成してデータセットの値を割り当てる

```
data dusty;
input dept $ name $ salary @@;
datalines;
bedding Watlee 18000 bedding Ives 16000
bedding Parker 9000 bedding George 8000
bedding Joiner 8000 carpet Keller 20000
carpet Ray 12000 carpet Jones 9000
gifts Johnston 8000 gifts Matthew 19000
kitchen White 8000 kitchen Banks 14000
kitchen Marks 9000 kitchen Cannon 15000
tv Jones 9000 tv Smith 8000
tv Rogers 15000 tv Morse 16000
;
proc means noprint;
class dept;
var salary;
output out=stats sum=s_sal;
run;
data _null_;
set stats;
if _n_=1 then call symput('s_tot',trim(left(s_sal)));
else call symput('s'||dept,trim(left(s_sal)));
run;
%put _user_;
```

このプログラムを実行すると、次に示す変数のリストが SAS ログに書き込まれます。

```
GLOBAL SCARPET 41000
GLOBAL SKITCHEN 46000
GLOBAL STV 48000
GLOBAL SGIFTS 27000
GLOBAL SBEDDING 59000
GLOBAL S_TOT 221000
```

CALL SYMPUTN ルーチン

SCL プログラムにおいて、数値をグローバルマクロ変数に割り当てます。

種類: SCL CALL ルーチン

参照項目: “SYMGET 関数” (238 ページ)、“SYMGETN 関数” (241 ページ)および“CALL SYMPUT ルーチン” (228 ページ)

構文

CALL SYMPUTN('*macro-variable*', *value*);

必須引数

macro-variable
アンパサンドを付けないグローバルマクロ変数名。一重引用符を付けていることに注意してください。あるいは、グローバルマクロ変数名が格納された SCL 変数の名前です。

value
割り当てる数値。数値または数値 SCL 変数の名前を指定できます。

詳細

SYMPUTN ルーチンは、数値を SAS グローバルマクロ変数に割り当てます。SYMPUTN は、SCL プログラムの実行時に値を割り当てます。SYMPUTN を使用して、SCL 変数に名前が格納されたマクロ変数に値を割り当てることもできます。たとえば、SCL 変数 UNITNUM の値を、'UNIT'が格納された SCL 変数 UNITVAR に割り当てるには、次のステートメントをサブミットします。

```
call symputn(unitvar,unitnum)
```

SYMPUTN は、CALL ステートメントで使用する必要があります。

注: CALL SYMPUTN を使用して作成したマクロ変数を、アンパサンド(&)を使用して参照することは効率的ではありません。代わりに、SYMGETN を使用してください。CALL SYMPUTN を使用して、数値を含まない変数を格納することも効率的ではありません。

比較

- SYMPUTN は数値を割り当てます。一方、SYMPUT は文字値を割り当てます。
- SYMPUTN は、SCL プログラム内でのみ使用できます。一方、SYMPUT は、DATA ステッププログラム内および SCL プログラム内で使用できます。
- SYMPUTN は数値を割り当てます。一方、SYMGETN は数値を取得します。

例: SCL プログラムの実行時のマクロ変数 UNIT への値 1000 の格納

次のステートメントは、SCL プログラムの実行時に、マクロ変数 UNIT に値 1000 を格納します。

```
call symputn('unit',1000);
```

CALL SYMPUTX ルーチン

マクロ変数に値を割り当て、値の先頭と末尾の空白を削除します。

カテゴリ: マクロ

参照項目: の“CALL SYMPUTX Routine” *SAS 関数と CALL ルーチン: リファレンス*

構文

CALL SYMPUTX(*macro-variable*, *value* <,*symbol-table*>);

16 章

マクロの DATA ステップ関数

マクロの DATA ステップ関数

DATA ステップ関数を使用して、マクロ機能を操作できます。

ディクショナリ

RESOLVE 関数

DATA ステップの実行時にテキスト式の値を置換します。

種類: DATA ステップ関数

構文

RESOLVE(*argument*)

必須引数

argument
次のいずれかの項目を指定できます。

- 一重引用符(DATA ステップの作成中にマクロプロセッサによって引数が置換されないようにするため)で囲んだテキスト式。マクロ変数値にマクロ変数参照が含まれている場合、RESOLVE はその参照を置換しようとします。*argument* が存在しないマクロ変数を参照している場合、RESOLVE は、未置換の参照を

返します。テキスト式を使用する次の例は、マクロ LOCATE によって生成されたテキストを割り当てる方法と、マクロ変数 NAME の値を割り当てる方法を示しています。

```
x=resolve('%locate');
x=resolve('&name');
```

- テキスト式を値として持つ DATA ステップ変数の名前。たとえば、次の例では、DATA ステップ変数 ADDR1 の現在の値に含まれるテキスト式を X に割り当てています。

```
addr1='&locate';
x=resolve(addr1);
```

- マクロ機能によって置換されるテキスト式を生成する文字式。たとえば、次の例では、マクロ名の作成において、DATA ステップ変数 STNUM の現在の値を使用しています。

```
x=resolve('%state'||left(stnum));
```

詳細

RESOLVE 関数は、特に置換先の変数に短い長さを割り当てていなければ、DATA ステップ文字変数の最大長の文字値を返します。それよりも長い値が返された場合、切り捨てられます。

RESOLVE は、引数で指定されたマクロ変数またはマクロを検出できなかった場合、未置換の引数を返し、マクロプロセッサによって警告メッセージが発行されます。

SYMPUT ルーチンを使用してマクロ変数を作成し、それと同じ DATA ステップ内で RESOLVE を使用してそのマクロ変数を置換できます。

比較

- RESOLVE は、DATA ステップまたは SCL プログラムの実行時にテキスト式の値を置換します。一方、マクロ変数参照は、DATA ステップの作成時または SCL プログラムのコンパイル時に置換されます。このため、置換されたマクロ変数参照の値は、DATA ステップまたは SCL プログラムが実行されている間、一定になります。これに対して、RESOLVE は、プログラムの各反復において、テキスト式の異なる値を返すことができます。
- RESOLVE は、SYMGET 関数が受け取るよりも、多くの種類の引数を受け取ることができます。SYMGET は 1 つのマクロ変数しか置換しません。一方、RESOLVE は任意のマクロ式を置換します。RESOLVE を使用することで、マクロを実行することや、複数のマクロ変数を置換することができます。
- マクロ変数の値に別のマクロ変数参照が含まれている場合、RESOLVE はその参照を置換しようとします。一方、その場合、SYMGET は置換しません。
- *argument* が存在しないマクロ変数を参照している場合、RESOLVE は未置換の参照を返します。一方、その場合、SYMGET は欠損値を返します。
- RESOLVE は、柔軟性が高いため、SYMGET よりもわずかに多くコンピュータリソースを必要とします。

例: サンプル参照の置換

次の例は、マクロ変数参照、マクロ呼び出し、およびマクロ呼び出しを値として持つ DATA ステップ変数と共に使用される RESOLVE を示しています。

```
%let event=Holiday;
```

```
%macro date;
New Year
%mend date;
data test;
length var1-var3 $ 15;
when='%date';
var1=resolve('&event'); /* macro variable reference */
var2=resolve('%date'); /* macro invocation */
var3=resolve(when); /* DATA step variable with macro invocation */
put var1= var2= var3=;
run;
```

このプログラムを実行すると、次の行が SAS ログに書き込まれます。

```
VAR1=Holiday VAR2=New Year VAR3=New Year
NOTE: The data set WORK.TEST has 1 observations and 4 variables.
```

SYMEXIST 関数

マクロ変数が存在するかどうかを示す値を返します。

種類: DATA ステップ関数

構文

SYMEXIST (*argument*)

必須引数

argument
次のいずれかの項目を指定できます。

- 引用符で囲み、アンパサンドを付けないマクロ変数の名前。
- DATA ステップ文字変数の名前。引用符を付けずに指定し、マクロ変数の名前を格納します。
- マクロ変数名を作成する文字式。

詳細

SYMEXIST 関数は、指定されたマクロ変数を、かっこで囲まれたローカルシンボルテーブルで検索し、次にグローバルシンボルテーブルで検索します。SYMEXIST 関数は、次のいずれかの値を返します。

- `1` マクロ変数が見つかった場合
- `0` マクロ変数が見つからなかった場合

例: SYMEXIST 関数の使用

次の例では、%TEST マクロに SYMEXIST 関数が含まれています。

```
%global x;
%macro test;
```

```
%local y;
data null;
if symexist("x") then put "x EXISTS";
else put "x does not EXIST";
if symexist("y") then put "y EXISTS";
else put "y does not EXIST";
if symexist("z") then put "z EXISTS";
else put "z does not EXIST";
run;
%mend test;
%test;
```

前述の例では、SYMEXIST 関数を含む%TEST マクロが実行されると、次の出力が SAS ログに書き込まれます。

```
x EXISTS
y EXISTS
z does not EXIST
```

SYMGET 関数

DATA ステップの実行時に、マクロ変数の値を DATA ステップに返します。

種類: DATA ステップ関数

参照項目: “RESOLVE 関数” (235 ページ)、“SYMGETN 関数” (241 ページ)、“CALL SYMPUT ルーチン” (228 ページ)、および“CALL SYMPUTN ルーチン” (232 ページ)

構文

SYMGET(*argument*)

必須引数

argument
次のいずれかの項目を指定できます。

- 引用符で囲み、アンパサンドを付けないマクロ変数の名前。マクロ変数値に別のマクロ変数参照が含まれる場合、SYMGET はその参照の置換を試みません。*argument* が存在しないマクロ変数を参照している場合、SYMGET は欠損値を返します。次の例は、マクロ変数 G の値を DATA ステップ変数 X に割り当てる方法を示しています。

  ```
  x=symget('g');
  ```

- DATA ステップ文字変数の名前。引用符を付けずに指定し、1 つ以上のマクロ変数の名前を格納します。この値が有効な SAS 名でない場合、またはマクロプロセッサがこの名前のマクロ変数を検出できなかった場合、注釈がログに書き込まれます。その注釈には、関数に不正な引数が渡されたため、戻り値に欠損値が設定されたことが示されます。たとえば、次のステートメントでは、マクロ変数名が格納された DATA ステップ変数 CODE の値を、DATA ステップ変数 KEY に割り当てています。

  ```
  length key $ 8;
  input code $;
  key=symget(code);
  ```

DATA ステップの反復ごとに、CODE の値によってマクロ変数名が指定され、そのマクロ変数の値が KEY に割り当てられます。

- マクロ変数名を作成する文字式。たとえば、次のステートメントでは、DATA ステップ自動変数_N_を使用して、文字 s と現在の反復回数を割り当てています。

```
score=symget('s'||left(_n_));
```

詳細

SYMGET は、DATA ステップ文字変数の最大長を持つ文字値を返します。それよりも長い値が返された場合、切り捨てられます。

SYMGET は、引数で指定されたマクロ変数を検出できなかった場合、欠損値を返し、不正な引数が関数に渡されたことを示すメッセージが発行されます。

SYMGET は、SCL プログラムなどのすべての SAS 言語プログラムで使用できます。SYMGET は、マクロの実行時ではなくプログラムの実行時に変数を置換するため、DATA ステップビュー、SQL ビュー、および SCL プログラムのマクロ変数値を返すために使用する必要があります。

比較

- SYMGET は、プログラムの実行時に、マクロ変数の値を返します。一方、SYMPUT 関数は、プログラムの実行時に、プログラムで生成された値をマクロ変数に割り当てます。
- SYMGET は、RESOLVE 関数よりも少ない種類の引数を受け取ります。SYMGET は、1 つのマクロ変数のみを置換します。RESOLVE を使用すると、マクロが実行されて、さらに値が置換されます。
- SYMGET は、すべての SAS プログラムで使用できます。一方、SYMGETN は、SCL プログラムでのみ使用できます。

例: データセットから以前割り当てられた変数値を取得する

```
data dusty;
input dept $ name $ salary @@;
datalines;
bedding Watlee 18000 bedding Ives 16000
bedding Parker 9000 bedding George 8000
bedding Joiner 8000 carpet Keller 20000
carpet Ray 12000 carpet Jones 9000
gifts Johnston 8000 gifts Matthew 19000
kitchen White 8000 kitchen Banks 14000
kitchen Marks 9000 kitchen Cannon 15000
tv Jones 9000 tv Smith 8000
tv Rogers 15000 tv Morse 16000
;
proc means noprint;
class dept;
var salary;
output out=stats sum=s_sal;
run;
proc print data=stats;
var dept s_sal;
title "Summary of Salary Information";
```

```
title2 "For Dusty Department Store";
run;
data _null_;
set stats;
if _n_=1 then call symput('s_tot',s_sal);
else call symput('s'||dept,s_sal);
run;
data new;
set dusty;
pctdept=(salary/symget('s'||dept))*100;
pcttot=(salary/&s_tot)*100;
run;
proc print data=new split="*";
label dept ="Department"
name ="Employee"
pctdept="Percent of *Department* Salary"
pcttot ="Percent of * Store * Salary";
format pctdept pcttot 4.1;
title "Salary Profiles for Employees";
title2 "of Dusty Department Store";
run;
```

このプログラムは、次の出力を生成します。

アウトプット 16.1 *給与情報の要約*

Summary of Salary Information
For Dusty Department Store

Obs	dept	s_sal
1		221000
2	bedding	59000
3	carpet	41000
4	gifts	27000
5	kitchen	46000
6	tv	48000

アウトプット 16.2　従業員の給与分析

Salary Profiles for Employees of Dusty Department Store

Obs	Department	Employee	salary	Percent of Department Salary	Percent of Store Salary
1	bedding	Watlee	18000	30.5	8.1
2	bedding	Ives	16000	27.1	7.2
3	bedding	Parker	9000	15.3	4.1
4	bedding	George	8000	13.6	3.6
5	bedding	Joiner	8000	13.6	3.6
6	carpet	Keller	20000	48.8	9.0
7	carpet	Ray	12000	29.3	5.4
8	carpet	Jones	9000	22.0	4.1
9	gifts	Johnston	8000	29.6	3.6
10	gifts	Matthew	19000	70.4	8.6
11	kitchen	White	8000	17.4	3.6
12	kitchen	Banks	14000	30.4	6.3
13	kitchen	Marks	9000	19.6	4.1
14	kitchen	Cannon	15000	32.6	6.8
15	tv	Jones	9000	18.8	4.1
16	tv	Smith	8000	16.7	3.6
17	tv	Rogers	15000	31.3	6.8
18	tv	Morse	16000	33.3	7.2

SYMGETN 関数

SAS コンポーネント制御言語(SCL)プログラムにおいて、グローバルマクロ変数の値を数値で返します。

種類: SCL 関数

参照項目: “SYMGET 関数” (238 ページ)、“CALL SYMPUT ルーチン” (228 ページ)、および“CALL SYMPUTN ルーチン” (232 ページ)

構文

SCL-variable=***SYMGETN***('*macro-variable*');

必須引数

SCL variable
macro-variable に格納されている値を格納する、数値 SCL 変数の名前。

macro-variable
アンパサンドを付けないグローバルマクロ変数名。一重引用符を付けていることに注意してください。あるいは、グローバルマクロ変数名が格納された SCL 変数の名前です。

詳細

SYMGETN は、グローバルマクロ変数の値を数値で返し、指定された数値 SCL 変数にそれを格納します。SYMGETN を使用して、SCL 変数に名前が格納されているマクロ変数の値を取得することもできます。たとえば、'UNIT'という値を持つ SCL 変数 UNITVAR から値を取得するには、次のコードサブミットします。

```
unitnum=symgetn(unitvar)
```

SYMGETN は、SCL プログラムの実行時に値を返します。SYMGETN は、*macro-variable* を検出できなかった場合、欠損値を返します。

SCL プログラムのコンパイル時に、マクロ変数に格納された値を返すには、次に示すように、割り当てステートメント内でマクロ変数参照を使用します。

```
SCL variable=&macro-variable;
```

注: SYMPUTN を使用して割り当てられていない値や、数値以外の値を、SYMGETN を使用して取得するのは効率的ではありません。

比較

- SYMGETN は、SCL プログラム内でのみ使用できます。一方、SYMGET は、DATA ステッププログラム内および SCL プログラム内で使用できます。
- SYMGETN は値を取得します。一方、SYMPUTN は値を割り当てます。

例: マクロ変数値を SCL プログラム内で数値として格納する

次のステートメントは、SCL プログラムの実行時に、マクロ変数 UNIT の値を SCL 変数 UNITNUM に格納します。

```
unitnum=symgetn('unit');
```

SYMGLOBL 関数

DATA ステップの実行時に、マクロ変数のスコープがグローバルかどうかを示す値を DATA ステップに返します。

種類: DATA ステップ関数

構文

SYMGLOBL (*argument*)

必須引数

argument
次のいずれかの項目を指定できます。

- 引用符で囲み、アンパサンドを付けないマクロ変数の名前。
- DATA ステップ文字変数の名前。引用符を付けずに指定し、マクロ変数名を格納します。
- マクロ変数名を作成する文字式。

詳細

SYMGLOBL 関数は、かっこで囲まれたスコープを検索して同じ名前のマクロ変数が存在するかどうかを調べ、そのマクロ変数がグローバルシンボルテーブル内に存在する場合は値 1 を返し、それ以外の場合は 0 を返します。グローバルシンボルテーブルとローカルシンボルテーブル、およびマクロ変数のスコープの詳細については、“マクロ変数のスコープ” (45 ページ)を参照してください。

例: SYMGLOBL 関数の使用

次の例では、%TEST マクロに SYMGLOBL 関数が含まれています。

```
 %global x;
%macro test;
%local y;
data null;
if symglobl("x") then put "x is GLOBAL";
else put "x is not GLOBAL";
if symglobl("y") then put "y is GLOBAL";
else put "y is not GLOBAL";
if symglobl("z") then put "z is GLOBAL";
else put "z is not GLOBAL";
run;
%mend test;
%test;
```

前述の例では、SYMGLOBL 関数を含む%TEST マクロが実行されると、次の出力が SAS ログに書き込まれます。

```
x is GLOBAL
y is not GLOBAL
z is not GLOBAL
```

SYMLOCAL 関数

DATA ステップの実行時に、マクロ変数のスコープがローカルかどうかを示す値を DATA ステップに返します。

種類: DATA ステップ関数

構文

SYMLOCAL (*argument*)

必須引数

argument
次のいずれかの項目を指定できます。

- 引用符で囲み、アンパサンドを付けないマクロ変数の名前。
- DATA ステップ文字変数の名前。引用符を付けずに指定し、マクロ変数名を格納します。
- マクロ変数名を作成する文字式。

詳細

SYMLOCAL 関数は、かっこで囲まれたスコープを検索して同じ名前のマクロ変数が存在するかどうかを調べ、そのマクロ変数がグローバルシンボルテーブル内に存在する場合は値 1 を返し、それ以外の場合は 0 を返します。グローバルシンボルテーブルとローカルシンボルテーブル、およびマクロ変数のスコープの詳細については、“マクロ変数のスコープ” (45 ページ)を参照してください。

例: SYMLOCAL 関数の使用

次の例では、%TEST マクロに SYMLOCAL 関数が含まれています。

```
 %global x;
%macro test;
%local y;
data null;
if symlocal("x") then put "x is LOCAL";
else put "x is not LOCAL";
if symlocal("y") then put "y is LOCAL";
else put "y is not LOCAL";
if symlocal("z") then put "z is LOCAL";
else put "z is not LOCAL";
run;
%mend test;
%test;
```

前述の例では、SYMLOCAL 関数を含む%TEST マクロが実行されると、次の出力が SAS ログに書き込まれます。

```
 x is not LOCAL
y is LOCAL
z is not LOCAL
```

17 章

マクロ関数

マクロ関数

各マクロ言語関数は、1 つ以上の引数を処理することで結果を生成します。

ディクショナリ

%BQUOTE 関数と%NRBQUOTE 関数

マクロの実行時に、置換された値に含まれている特殊文字やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: “%QUOTE 関数と%NRQUOTE 関数” (252 ページ) and “%SUPERQ 関数” (263 ページ)

構文

%BQUOTE (*character string* | *text expression*)

%NRBQUOTE (*character string* | *text expression*)

詳細

%BQUOTE 関数と%NRBQUOTE 関数は、マクロまたはマクロ言語ステートメントの実行時に、文字列またはテキスト式の置換された値をマスクします。これらの関数は、次に示す特殊文字とニーモニック演算子をマスクします。

```
' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

さらに、%NRBQUOTE 関数は次のものもマスクします。

```
& %
```

%NRBQUOTE 関数は、置換された引数値が次のものを含んでいる場合に使用すると便利です。

- マクロ変数参照のように見えるが実はそうではない文字列。この場合、マクロプロセッサがこのような文字列を次回検出した際に、マクロプロセッサが同文字列を置換しないようにする必要があります。
- マクロプロセッサによる次回検出時に、マクロプロセッサによって置換されたくないマクロ呼び出し。

注: マクロクォーティング関数の最大ネストレベルは 10 です。

ヒント: %BQUOTE 関数および%NRBQUOTE 関数はマクロ言語の要素として解釈可能なすべての文字やニーモニック演算子をマスクするため、これらの関数はすべての実行時マクロクォーティングに対して使用できます。

引用符(' ")をマークする必要はありません。

SAS マクロ言語におけるクォーティングに関する詳細は、“マクロクォーティング” (82 ページ)を参照してください。

比較

%NRBQUOTE 関数および%SUPERQ 関数は、同じ項目をマスクします。ただし、%SUPERQ 関数は、指定のマクロ変数の値に含まれているマクロ変数参照やマクロ呼び出しの置換を試みません。一方、%NRBQUOTE 関数は、それらの参照の置換を

試みます。%BQUOTE 関数や%NRBQUOTE 関数を使用する場合、引用符をマークする必要はありません。

例: 変数のクォーティング

次の例では、マクロ FILEIT に渡されるファイル名が引用符で始まるかどうかをテストしています。このテスト結果に基づいて、同マクロは正しい FILE コマンドを生成します。

```
%macro fileit(infile);
%if %bquote(&infile) NE %then
%do;
%let char1 = %bquote(%substr(&infile,1,1));
%if %bquote(&char1) = %str(%')
or %bquote(&char1) = %str(%")
%then %let command=FILE &infile;
%else %let command=FILE "&infile";
%end;
%put &command;
%mend fileit;
%fileit(myfile)
%fileit('myfile')
```

このプログラムを実行すると、次のメッセージがログに出力されます。

```
FILE "myfile"
FILE 'myfile'
```

%EVAL 関数

整数演算を使用して演算式や論理式を評価します。

種類: マクロ評価関数

参照項目: “%SYSEVALF 関数” (267 ページ)

構文

%EVAL (*arithmetic or logical expression*)

詳細

%EVAL 関数は、整数演算式または論理式を評価します。%EVAL 関数は、呼び出されると、まずその引数を文字値から数式または論理式に変換します。続いて、同関数は評価を実行します。最後に、%EVAL 関数は得られた結果を文字値に変換し、その値を返します。

すべてのオペランドが整数に変換できる場合、式は演算式として扱われます。数値に変換できないオペランドが 1 つでも存在する場合、式は論理式として扱われます。除算の結果が分数になる場合、その結果は整数に切り捨てられます。

論理(つまりブール)式は、true または false として評価される値を返します。マクロ言語では、0 以外のすべての数値は true になり、0 の値は false になります。

%EVAL 関数は、整数(標準形式または 16 進形式)を表す演算式でのみオペランドを受け付けます。ピリオド文字を含んでいるオペランドを整数演算式に含めると、エラーが発生します。%EVAL 関数の正しい使い方と誤った使い方の例をそれぞれ次に示します。

```
%let d=%eval(10+20); /* Correct usage */
%let d=%eval(10.0+20.0); /* Incorrect usage */
```

%EVAL 関数はピリオドを含んでいる値を数に変換しないため、これらのオペランドは文字オペランドとして評価されます。%EVAL 関数はピリオドを含んでいる値を検出すると、数値オペランドが必要な箇所に文字オペランドが見つかったというエラーメッセージを表示します。

%EVAL 関数に文字値を比較する式を指定した場合、お使いの動作環境における並べ替え順を使用して比較が行われます。お使いの動作環境における並べ替え順についての詳細は、*Base SAS プロシジャガイド*の The SORT PROCEDURE を参照してください。

式を評価するマクロ言語の部分(%IF ステートメントや%DO ステートメントなど)はすべて、%EVAL 関数を呼び出すことにより条件を評価します。マクロ式の評価方法に関する詳細は、“マクロ式” (73 ページ)を参照してください。

比較

%EVAL 関数は整数評価を実施します。一方、%SYSEVALF 関数は浮動小数点評価を実施します。

例

例 1: 整数演算評価の概要

次のステートメントは、様々なタイプの評価を表しています。

```
%let a=1+2;
%let b=10*3;
%let c=5/3;
%let eval_a=%eval(&a);
%let eval_b=%eval(&b);
%let eval_c=%eval(&c);
%put &a is &eval_a;
%put &b is &eval_b;
%put &c is &eval_c;
```

これらのステートメントをサブミットすると、次のメッセージが SAS ログに出力されます。

```
1+2 is 3
10*3 is 30
5/3 is 1
```

3 番目の%PUT ステートメントは、結果が分数となる整数除算を実行した場合、その小数部が%EVAL により破棄されることを示します。

例 2: カウンタのインクリメント

次の例に示すマクロ TEST は、%EVAL 関数を使用して、マクロ変数 I の値を 1 ずつインクリメントします。また、%DO %WHILE ステートメントで%EVAL 関数を呼び出すことにより、マクロ変数 I の値がマクロ変数 FINISH の値よりも大きいかどうかを評価します。

```
%macro test(finish);
```

```
%let i=1;
%do %while (&i<&finish);
%put the value of i is &i;
%let i=%eval(&i+1);
%end;
%mend test;
%test(5)
```

このプログラムを実行すると、次のメッセージが SAS ログに出力されます。

```
The value of i is 1
The value of i is 2
The value of i is 3
The value of i is 4
```

例 3: 論理式の評価

マクロ COMPARE は 2 つの数を比較します。

```
%macro compare(first,second);
%if &first>&second %then %put &first > &second;
%else %if &first=&second %then %put &first = &second;
%else %put &first<&second;
%mend compare;
%compare(1,2)
%compare(-1,0)
```

このプログラムを実行すると、次のメッセージが SAS ログに出力されます。

```
1 < 2
-1 < 0
```

%INDEX 関数

文字列の先頭文字の位置を返します。

種類: マクロ関数

構文

%INDEX (*source*, *string*)

必須引数

source
文字列またはテキスト式です。

string
文字列またはテキスト式です。

詳細

%INDEX 関数は、*source* を検索して *string* の最初のオカレンスを見つけ、その先頭文字の位置を返します。*string* が見つからない場合、この関数は 0 を返します。

例: 文字の検索

次のステートメントは、文字列内に文字 v が最初に現れる位置を返します。

```
 %let a=a very long value;
%let b=%index(&a,v);
%put V appears at position &b..;
```

このステートメントを実行すると、次のメッセージが SAS ログに出力されます。

```
V appears at position 3.
```

%LENGTH 関数

文字列の長さを返します。

種類: マクロ関数

構文

%LENGTH (*character string* | *text expression*)

詳細

引数が文字列である場合、%LENGTH 関数はその文字列の長さを返します。引数がテキスト式である場合、%LENGTH 関数はその置換後の値の長さを返します。引数がヌル値の場合、%LENGTH 関数は 0 を返します。

例: 文字列長を返す

次のステートメントは、文字列の長さとテキスト式の長さを調べます。

```
%let a=Happy;
%let b=Birthday;
%put The length of &a is %length(&a).;
%put The length of &b is %length(&b).;
%put The length of &a &b To You is %length(&a &b to you).;
```

これらのステートメントを実行すると、次のメッセージが SAS ログに出力されます。

```
The length of Happy is 5.
The length of Birthday is 8.
The length of Happy Birthday To You is 21.
```

%NRBQUOTE 関数

マクロの実行時に、置換された値に含まれている特殊文字(&、%など)やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: “%BQUOTE 関数と%NRBQUOTE 関数” (246 ページ)

構文

%NRBQUOTE (*character string* | *text expression*)

引数なし

マクロクォーティング関数の最大ネストレベルは 10 です。

%NRQUOTE 関数

マクロの実行時に、置換された値に含まれている特殊文字(&、%など)やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: “%QUOTE 関数と%NRQUOTE 関数” (252 ページ)

構文

%NRQUOTE (*character string* | *text expression*)

引数なし

マクロクォーティング関数の最大ネストレベルは 10 です。

%NRSTR 関数

マクロのコンパイル時に、定数テキストに含まれている特殊文字(&、%など)やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: “%STR 関数と%NRSTR 関数” (258 ページ)

構文

%NRSTR (*character-string*)

引数なし

マクロクォーティング関数の最大ネストレベルは 10 です。

%QSCAN 関数

特定のワードを検索し、特殊文字やニーモニック演算子をマスクします。

種類: マクロ関数

構文

%QSCAN (*argument*,*n*<,charlist<,modifiers> >)

引数なし

“%SCAN 関数と%QSCAN 関数” (254 ページ)

%QSUBSTR 関数

部分文字列を取り出し、特殊文字やニーモニック演算子をマスクします。

種類: マクロ関数

構文

%QSUBSTR (*argument*, *position*<, *length*>)

引数なし
次のドキュメントを参照: "%SUBSTR 関数と%QSUBSTR 関数" (261 ページ)

%QSYSFUNC 関数

関数を実行し、特殊文字やニーモニック演算子をマスクします。

種類: マクロ関数

構文

%QSYSFUNC (*function*(*argument-1* <...*argument-n*>)<, *format*>)

引数なし
次のドキュメントを参照: "%SYSFUNC 関数と%QSYSFUNC 関数" (273 ページ)

%QUOTE 関数と%NRQUOTE 関数

マクロの実行時に、置換された値に含まれている特殊文字やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: "%BQUOTE 関数と%NRBQUOTE 関数" (246 ページ), "%NRBQUOTE 関数" (250 ページ), "%NRSTR 関数" (251 ページ)、および"%SUPERQ 関数" (263 ページ)

構文

%QUOTE (*character string* | *text expression*)

%NRQUOTE (*character string* | *text expression*)

詳細

%QUOTE 関数と%NRQUOTE 関数は、マクロまたはマクロ言語ステートメントの実行時に、文字列またはテキスト式の置換された値をマスクします。これらの関数は、次に示す特殊文字とニーモニック演算子をマスクします。

```
+ - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

また、これらの関数は、次の文字がペアで検出された場合や、次の文字がペアマッチなしで検出され、その文字が先行する%によりマークされている場合に、その文字をマスクします。

```
' "
```

さらに、%NRBQUOTE 関数は次のものもマスクします。

```
& %
```

%NRQUOTE 関数は、置換したくないマクロ変数参照やマクロ呼び出しが引数に含まれている場合に使用すると便利です。

SAS マクロ言語におけるクォーティングに関する詳細は、“マクロクォーティング” (82 ページ)を参照してください。

マクロクォーティング関数の最大ネストレベルは 10 です。

比較

- %QUOTE 関数と%NRQUOTE 関数は、それぞれ%STR 関数と%NRSTR 関数がマスクするのと同じ項目をマスクします。ただし、%STR 関数と%NRSTR 関数は、置換された値ではなく、定数テキストをマスクします。また、%STR 関数と%NRSTR 関数はマクロのコンパイル時に動作しますが、%QUOTE 関数と%NRQUOTE 関数はマクロの実行時に動作します。
- %BQUOTE 関数および%NRBQUOTE 関数では、ペアマッチなしの引用符を先行する%でマークする必要はありません。一方、%QUOTE 関数および%NRQUOTE 関数では、それらの引用符をマークする必要があります。
- %QUOTE 関数と%NRQUOTE 関数は置換された値をマスクします。一方、%SUPERQ 関数は、値の中で発生する任意のマクロ呼び出しやマクロ変数参照の置換が行われないようにします。

例: ニーモニック演算子を含む値のクォーティング

マクロ DEPT1 は、州の略称を引数として受け付けます。ここで、DEPT1 の引数にオレゴン州の略称である OR を指定したとします。

```
%macro dept1(state);
/* without %quote -- problems might occur */
%if &state=nc %then
%put North Carolina Department of Revenue;
%else %put Department of Revenue;
%mend dept1;
%dept1(or)
```

前述のマクロ DEPT1 を実行すると、%IF 条件で%EVAL ステートメントが実行されるため、文字列 **or** はこの式では論理演算子として評価されることになります。このため、マクロプロセッサは式 **or=nc** に無効なオペランドが含まれているというエラーメッセージを出力します。

マクロ DEPT2 は、%QUOTE 関数を使って、&STATE を置換した結果得られる文字をクォーティングしています。

```
%macro dept2(state);
/* with %quote function--problems are prevented */
%if %quote(&state)=nc %then
%put North Carolina Department of Revenue;
%else %put Department of Revenue;
%mend dept2;
%dept2(or)
```

この場合、%IF 条件は文字列 or と nc を比較し、次のメッセージを SAS ログに出力します。

```
Department of Revenue
```

%QUPCASE 関数

値を大文字に変換し、特殊文字とニーモニック演算子をマスクした結果を返します。

種類: マクロ関数

構文

%QUPCASE (*character string* | *text expression*)

引数なし

次のドキュメントを参照: "%UPCASE 関数と%QUPCASE 関数" (280 ページ)

%SCAN 関数と%QSCAN 関数

文字列内の位置により指定されるワードを検索します。

種類: マクロ関数

参照項目: "%NRBQUOTE 関数" (250 ページ)および"%STR 関数と%NRSTR 関数" (258 ページ)

構文

%SCAN(*argument*, *n*<,*charlist* <,*modifiers*> >)

%QSCAN(*argument*, *n*<,*charlist* <,*modifiers*> >)

必須引数

argument

文字列またはテキスト式です。*argument* が次に示すような特殊文字やニーモニック演算子を含んでいる場合、%QSCAN を使用します。*argument* がカンマを含んでいる場合、%BQUOTE(*argument*)のように、クォーティング関数を使用して *argument* を囲みます。

n

この関数が返すワードの位置を表す整数、またはそのような整数を生成するテキスト式です(暗黙の%EVAL は、*n* 個の数値プロパティを提供します)。*n* が *argument* 内にあるワード数よりも大きい場合、この関数はヌル文字列を返します。

注: バージョン 8 以降の SAS システムでは、*n* が負数である場合、%SCAN 関数は文字列を検査し、その文字列の末尾にあるワードから逆方向に検索を実施します。

charlist

文字のリストを初期化するオプションの文字式を指定します。このリストは、複数のワードを区切るための区切り文字として使用される文字が決定するものです。次の規則が適用されます。

- デフォルトでは、*charlist* 内にあるすべての文字が区切り文字として使用されます。
- 引数 *modifier* にモディファイヤ K を指定すると、*charlist* 内に存在しないすべての文字が区切り文字として使用されます。

ヒント *charlist* に文字を追加するには、次に示す各種のモディファイヤを使用します。

modifier

%SCAN 関数の動作を変更する非空白文字を含む文字定数、変数、式を指定します。空白は無視されます。モディファイヤとして使用できる文字は次の通りです。

a または A　アルファベット文字を文字リストに追加します。

b または B　引数 *count* の符号にかかわらず、左から右へではなく、右から左にスキャンを実行します。

c または C　制御文字を文字リストに追加します。

d または D　数字を文字リストに追加します。

f または F　下線と英字(VALIDVARNAME=V7 オプション使用時の SAS 変数名で有効な先頭文字)を文字リストに追加します。

g または G　グラフィック文字を文字リストに追加します。グラフィック文字とは、紙の上にイメージとして印刷される文字のことです。

h または H　水平タブを文字リストに追加します。

i または I　大文字小文字を無視します。

k または K　文字リストに含まれていないすべての文字が区切り文字として扱われるようにします。つまり、モディファイヤ K を指定すると、文字リストに含まれている文字が、区切り文字として省略されるのではなく、戻り値内に保持されるようになります。K を指定しない場合、文字リスト内のすべての文字が区切り文字として扱われます。

l または L　小文字を文字リストに追加します。

m または M　複数の連続する区切り文字、および引数 *string* の先頭または末尾にある区切り文字が、長さゼロのワードを意味することを指定します。モディファイヤ M を指定しない場合、複数の連続する区切り文字は 1 つの区切り文字として扱われ、引数 *string* の先頭または末尾にある区切り文字は無視されます。

n または N　数字、下線、英字(VALIDVARNAME=V7 オプション使用時の SAS 変数名に表示される文字)を文字リストに追加します。

o または O　引数 *charlist* および引数 *modifier* を、%SCAN 関数が呼び出されるたびに処理するのではなく、一度だけ処理します。DATA ステップ(WHERE 句を除く)または SQL プロシジャでモディファイヤ O を使用すると、引数 *charlist* および *modifier* が変化しないようなループで%SCAN 関数を呼び出す場合、同関数の処理が高速になります。モディファイヤ O は、ユーザーが作成する SAS プログラム内の%SCAN 関数の各インスタンスに対して個別に適用されます。モ

ディファイヤ O は、%SCAN 関数のすべてのインスタンスで同じ区切り文字やモディファイヤが使われるようにするものではありません。

p または P	句読点を文字リストに追加します。
q または Q	引用符で囲まれた部分文字列内にある区切り文字を無視します。引数 *string* の値に一致しない引用符が含まれている場合、左から右にスキャンした場合と右から左にスキャンした場合とでは異なるワードが生成されます。
r または R	%SCAN 関数が返すワードから、先頭または末尾にある空白を削除します。モディファイヤ Q および R の両方を指定すると、%SCAN 関数はまずワードの先頭または末尾にある空白を削除します。続いて、同ワードが引用符で始まる場合、%SCAN 関数は引用符の 1 ペアを同ワードから削除します。
s または S	スペース類(空白、水平タブ、垂直タブ、復帰、改行、改ページ)を文字リストに追加します。
t または T	引数 *string* および引数 *charlist* の末尾にある空白を削除します。どちらか一方の引数だけから末尾の空白を削除したい場合、モディファイヤ T 付きの%SCAN 関数ではなく、TRIM 関数を使用してください。
u または U	大文字を文字リストに追加します。
w または W	印刷可能文字を文字リストに追加します。
x または X	16 進文字を文字リストに追加します。

ヒント 引数 *modifier* が文字定数である場合、それを引用符で囲む必要があります。複数のモディファイヤは、引用符で囲んで指定します。引数 *modifier* には、文字変数や文字式も指定できます。

詳細

%SCAN 関数と%QSCAN 関数は、*argument* を検索し、その *n* 番目のワードを返します。ワードとは、区切り文字(複数可)によって区切られた文字(複数可)のことです。

%SCAN 関数は、引数がそれまでマクロクォーティング関数によりマスクされていた場合であっても、同関数が返す結果内で特殊文字やニーモニック演算子をマスクしません。%QSCAN 関数は、同関数が返す結果内で次の特殊文字とニーモニック演算子をマスクします。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

区切り文字とワードの定義

区切り文字とは、ワードを区切るために使用される文字のことです。区切り文字を指定するには、引数 *charlist* および引数 *modifier* を使用します。

モディファイヤ Q を指定した場合、引用符で囲まれた部分文字列内にある区切り文字が無視されます。

%SCAN 関数では、ワードとは、次に示す条件をすべて満たしている部分文字列を指します。

- 左端が区切り文字に接しているか、または文字列の先頭であること
- 右端が区切り文字に接しているか、または文字列の末尾であること
- 区切り文字を含んでいないこと

区切り文字が文字列の先頭または末尾に存在する場合、あるいは文字列が 2 つ以上の連続する区切り文字を含んでいる場合、ワードの長さはゼロになります。ただし、モディファイヤ M を指定しない場合、%SCAN 関数は、長さがゼロであるワードを無視します。

ASCII 環境や EBCDIC 環境でのデフォルト区切り文字の使用

2 つの引数のみを持つ%SCAN 関数を使用する場合、デフォルトの区切り文字は、お使いのコンピュータが ASCII と EBCDIC のどちらをコード化文字セットとして使用しているかによって異なります。

- お使いのコンピュータが ASCII 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - . / ; < ^|

  文字^を含まない ASCII 環境では、%SCAN 関数は代わりに文字~を使用します。

- お使いのコンピュータが EBCDIC 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - . / ; < ¬ | ¢|

区切り文字としていかなる文字も指定せずに引数 *modifier* を使用した場合、使用できる区切り文字は、引数 *modifier* により定義された区切り文字のみになります。この場合、ASCII および EBCDIC 環境でのデフォルト区切り文字のリストは使用されません。つまり、モディファイヤは、引数 *charlist* に指定された区切り文字のリストに文字を追加します。モディファイヤは、デフォルトのモディファイヤのリストには文字を追加しません。

モディファイヤ M を伴う%SCAN 関数の使用

モディファイヤ M を指定すると、文字列内に含まれているワード数が、同じ文字列内に含まれている区切り文字の数に 1 を加えた数として定義されます。ただし、モディファイヤ Q を指定すると、引用符内にある区切り文字は無視されます。

モディファイヤ M を指定すると、次の条件のいずれかが true である場合、%SCAN 関数は長さがゼロのワードを返します。

- 文字列の先頭が区切り文字であり、ユーザーが最初のワードを要求した場合
- 文字列の末尾が区切り文字であり、ユーザーが最後のワードを要求した場合
- 文字列が 2 つの連続する区切り文字を含んでおり、ユーザーがこれら 2 つの区切り文字間にあるワードを要求した場合

モディファイヤ M を伴わない%SCAN 関数の使用

モディファイヤ M を指定しない場合、文字列内に含まれているワード数が、同じ文字列内に含まれている連続する非区切り文字からなる最大部分文字列の数として定義されます。ただし、モディファイヤ Q を指定すると、引用符内にある区切り文字は無視されます。

モディファイヤ M を指定しない場合、%SCAN 関数は次のように動作します。

- 文字列の先頭または末尾にある区切り文字を無視します。
- 2 つ以上の連続する区切り文字を、それらが単一の区切り文字であるかのように扱います。

文字列に区切り文字のみが含まれている場合、または文字列内のワード数の絶対値よりも大きいカウント数を指定した場合、%SCAN 関数は次のいずれかを返します。

- DATA ステップから%SCAN 関数を呼び出した場合、単一の空白
- マクロプロセッサから%SCAN 関数を呼び出した場合、長さがゼロの文字列

ヌル引数の使用

%SCAN 引数では、文字引数をヌルにできます。ヌル引数は長さがゼロの文字列として扱われます。数値引数はヌルにできません。

比較

%QSCAN 関数は、%NRBQUOTE 関数と同じ文字をマスクします。

例: %SCAN 関数と%QSCAN 関数のアクションの比較

次の例は、%SCAN 関数と%QSCAN 関数のアクションを比較するものです。

```
%macro a;
aaaaaa
%mend a;
%macro b;
bbbbbb
%mend b;
%macro c;
cccccc
%mend c;
%let x=%nrstr(%a*%b*%c);
%put X: &x;
%put The third word in X, with SCAN: %scan(&x,3,*);
%put The third word in X, with QSCAN: %qscan(&x,3,*);
```

前述の%PUT ステートメントは、次の行を SAS ログに出力します。

```
X: %a*%b*%c
The third word in X, with SCAN: cccccc
The third word in X, with QSCAN: %c
```

%STR 関数と%NRSTR 関数

マクロのコンパイル時に、定数テキストに含まれている特殊文字やニーモニック演算子をマスクします。

種類: マクロクォーティング関数

参照項目: “%NRQUOTE 関数” (251 ページ)

構文

%STR (*character-string*)

%NRSTR (*character-string*)

詳細

%STR 関数および%NRSTR 関数は、マクロまたはマクロ言語ステートメントのコンパイル時に、特定の文字をマスクします。これらの関数は、次に示す特殊文字とニーモニック演算子をマスクします。

```
+ - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

また、これらの関数は、次の文字がペアで検出された場合や、次の文字がペアマッチなしで検出され、その文字が先行する%によりマークされている場合に、その文字をマスクします。

```
' " ( )
```

さらに、%NRSTR 関数は次の文字もマスクします。

```
& %
```

表 17.1 *%STR 関数と%NSTR 関数における引数の使用*

引数	使用
引用符の前にあるパーセント記号、%'や%”など	引用符付きのパーセント記号 例: %let percent=%str(Jim%'s office);
丸かっこの前にあるパーセント記号、%(や%)など	2 つのパーセント記号(%%) 例: %let x=%str(20%%);
コメント記号付きの文字列、/* や -->など	各文字ごとに%STR 関数を適用 例: %str(/) %str(*) *comment-text* %str(*) %str(/)

%STR 関数は、次のものを含んでいる文字列を処理する場合に便利です。

- マクロプログラムステートメントの一部としてではなく、テキストとして扱う必要のあるセミコロン
- 意味のある空白
- ペアとしてマッチしない引用符や丸かっこ

ネストした%STR 関数や%QUOTE 関数の内部に、同じ引数を配置するのは冗長です。次の例では、マクロのコンパイル時に%STR 関数によりマスクされた引数が、マクロの実行時にもマスクされたままになることを示しています。したがって、この例で使用されている%QUOTE 関数には効果がありません。

```
%quote(%str(argument))
```

注意:
パラメータ値のリストを含む他のマクロ関数やマクロ呼び出しを、%STR 関数の引数に指定しないでください。%STR 関数はマッチしない丸かっこをマスクするため、マクロプロセッサは、関数の引数やマクロ呼び出しのパラメータ値を認識できなくなります。

SAS マクロ言語におけるクォーティングに関する詳細は、“マクロクォーティング” (82 ページ)を参照してください。

注: マクロクォーティング関数の最大ネストレベルは 10 です。

比較

- すべてのマクロクォーティング関数の中で、コンパイル時に有効となるのは%NRSTR 関数と%STR 関数のみです。それ以外のマクロクォーティング関数は、マクロの実行時に有効となります。
- %STR 関数と%NRSTR 関数は、それぞれ%QUOTE 関数と%NRQUOTE 関数がマスクするのと同じ項目をマスクします。ただし、%QUOTE 関数と%NRQUOTE 関数はマクロの実行時に有効となります。
- マクロ式を置換した結果、マスクが必要な項目が生成される場合、%STR 関数や%NRSTR 関数ではなく、%BQUOTE 関数や%NRBQUOTE 関数を使用します。

例

例 1: 先頭の空白の保持

次の例は、マクロ変数 TIME の値に先頭の空白を含めるようにします。

```
%let time=%str( now);
%put Text followed by the value of time:&time;
```

この例を実行すると、次の行が SAS ログに出力されます。

```
Text followed by the value of time: now
```

例 2: 空白を保護してテキストとしてコンパイルされるようにする

次の例では、%QSCAN 関数がワード間の区切り文字として空白を使用するよう指定します。

```
%macro words(string);
%local count word;
%let count=1;
%let word=%qscan(&string,&count,%str( ));
%do %while(&word ne);
%let count=%eval(&count+1);
%let word=%qscan(&string,&count,%str( ));
%end;
%let count=%eval(&count-1);
%put The string contains &count words.;
%mend words;
%words(This is a very long string)
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
The string contains 6 words.
```

例 3: マクロ参照を含む値のクォーティング

マクロ REVRS は、マクロ TEST により生成された文字列を逆順にします。%PUT ステートメント内の%NRSTR 関数は、文字列**%test&test** がマクロ呼び出しとして解釈されるのではなく、テキストとしてコンパイルされるように同文字列を保護します。

```
%macro revrs(string);
%local nstring;
%do i=%length(&string) %to 1 %by -1;
%let nstring=&nstring%qsubstr(&string,&i,1);
%end;
&nstring
%mend revrs;
```

```
%macro test;
Two words
%mend test;
%put %nrstr(%test%test) - %revrs(%test%test);
```

このプログラムを実行すると、次の行が SA ログに出力されます。

```
1 %macro revrs(string);
2 %local nstring;
3 %do i=%length(&string) %to 1 %by -1;
4 %let nstring=&nstring%qsubstr(&string,&i,1);
5 %end;&nstring
6 %mend revrs;
7
8 %macro test;
9 Two words
10 %mend test;
11
12 %put %nrstr(%test%test) - %revrs(%test%test);
%test%test - sdrow owTsdrow owT
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
real time 0.28 seconds
cpu time 0.12 seconds
```

%SUBSTR 関数と%QSUBSTR 関数

特定の文字列の部分文字列を作成します。

種類: マクロ関数

参照項目: “%NRBQUOTE 関数” (250 ページ)

構文

%SUBSTR (*argument*, *position*<, *length*>)

%QSUBSTR (*argument*, *position*<, *length*>)

必須引数

argument
文字列またはテキスト式です。*argument* が次に示すような特殊文字やニーモニック演算子を含んでいる場合、%QSUBSTR を使用します。

position
部分文字列内の先頭文字の位置を表す整数、またはそのような整数を生成する式(テキスト式、論理式、演算式)です。*position* の値が文字列内の文字数よりも大きい場合、%SUBSTR 関数および%QSUBSTR 関数は警告メッセージを発行し、ヌル値を返します。%EVAL 関数が自動的に呼び出されるため、*n* は数値として扱われます。

length
部分文字列内の文字数を表すオプション整数、またはそのような整数を生成する式(テキスト式、論理式、演算式)です。*length* の値が、*argument* 内の *position* 以降にある文字数よりも大きい場合、%SUBSTR 関数および%QSUBSTR 関数は警告メッセージを発行し、*position* から文字列の末尾までの文字を含む部分文字列を

返します。デフォルトでは、%SUBSTR 関数および%QSUBSTR 関数は、*position* から文字列の末尾までの文字を含む部分文字列を返します。

詳細

%SUBSTR 関数および%QSUBSTR 関数は、*argument* に指定された文字列の *position* 番目の文字から数えて *length* 個目までの文字を含む部分文字列を返します。

%SUBSTR 関数は、引数がそれまでマクロクォーティング関数によりマスクされていた場合であっても、同関数が返す結果内で特殊文字やニーモニック演算子をマスクしません。%QSUBSTR 関数は、次に示す特殊文字とニーモニック演算子をマスクします。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

比較

%QSUBSTR 関数は、%NRBQUOTE 関数と同じ文字をマスクします。

例

例 1: ファイル参照名を 8 文字に制限する例

マクロ MAKEFREF は、パラメータに 8 文字を超える長さの値が指定された場合、%SUBSTR 関数を使用して同パラメータ値の最初の 8 文字をファイル参照名に割り当てます。

```
%macro makefref(fileref,file);
%if %length(&fileref) gt 8 %then
%let fileref = %substr(&fileref,1,8);
filename &fileref "&file";
%mend makefref;
%makefref(humanresource,/dept/humanresource/report96)
```

このマクロを実行すると、次の SAS ステートメントが生成されます。

```
FILENAME HUMANRES "/dept/humanresource/report96";
```

例 2: セグメントに長いマクロ変数値を保存する例

マクロ SEPMSG は、マクロ変数 MSG の値を 40 文字からなるユニットに分割し、各ユニットを別々の変数に格納します。

```
%macro sepmsg(msg);
%let i=1;
%let start=1;
%if %length(&msg)>40 %then
%do;
%do %until(%length(&&msg&i)<40);
%let msg&i=%qsubstr(&msg,&start,40);
%put Message &i is: &&msg&i;
%let i=%eval(&i+1);
%let start=%eval(&start+40);
%let msg&i=%qsubstr(&msg,&start);
%end;
%put Message &i is: &&msg&i;
%end;
%else %put No subdivision was needed.;
```

```
%mend sepmsg;
%sepmsg(%nrstr(A character operand was found in the %EVAL function
or %IF condition where a numeric operand is required. A character
operand was found in the %EVAL function or %IF condition where a
numeric operand is required.));
```

このプログラムを実行すると、SAS ログに次のメッセージが出力されます。

```
Message 1 is: A character operand was found in the %EV
Message 2 is: AL function or %IF condition where a nu
Message 3 is: meric operand is required. A character
Message 4 is: operand was found in the %EVAL function
Message 5 is: or %IF condition where a numeric operan
Message 6 is: d is required.
```

例 3: %SUBSTR 関数と%QSUBSTR 関数のアクションの比較

次の例では、変数 C の値が%NRSTR 関数によりマスクされているため、変数 C の値はコンパイル時には置換されません。ただし、変数 C の値が%NRSTR 関数により事前にマスクされていたとしても、%SUBSTR 関数は、変数 C に含まれている特殊文字やニーモニック演算子をマスクせずに変数 C の値を処理するため、%SUBSTR 関数は置換された結果を生成します。

```
%let a=one;
%let b=two;
%let c=%nrstr(&a &b);
%put C: &c;
%put With SUBSTR: %substr(&c,1,2);
%put With QSUBSTR: %qsubstr(&c,1,2);
```

これらのステートメントを実行すると、次の行が SAS ログに表示されます。

```
C: &a &b
With SUBSTR: one
With QSUBSTR: &a
```

%SUPERQ 関数

マクロ実行時にすべての特殊文字とニーモニック演算子をマスクし、値の置換がそれ以降行われないようにします。

種類: マクロクォーティング関数

参照項目: “%NRBQUOTE 関数” (250 ページ)および“%BQUOTE 関数と%NRBQUOTE 関数” (246 ページ)

構文

%SUPERQ (*argument*)

必須引数

argument
先頭にアンパサンドが付いていないマクロ変数名か、または先頭にアンパサンドが付いていないマクロ変数名を生成するテキスト式を指定します。

詳細

%SUPERQ 関数は、値に含まれているマクロやマクロ変数参照を置換せずに、マクロ変数の値を返します。%SUPERQ 関数は、次に示す特殊文字とニーモニック演算子をマスクします。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

%SUPERQ 関数は、特にアンパサンドやパーセント記号を含んでいる可能性のあるマクロ変数が%INPUT ステートメント、%WINDOW ステートメント、SYMPUT ルーチンなどで使用される場合に、そのような変数をマスクするのに使用すると便利です。

SAS マクロ言語におけるクォーティングに関する詳細は、“マクロクォーティング” (82 ページ)を参照してください。

注: マクロクォーティング関数の最大ネストレベルは 10 です。

比較

- %SUPERQ 関数は、指定されたマクロ変数の値に含まれているマクロ変数やマクロ参照が置換されないようにする唯一のクォーティング関数です。
- %SUPERQ 関数は、その引数として、アンパサンドが付いていないマクロ変数の名前のみを受け付けます。一方、それ以外のクォーティング関数は、定数テキストを含む任意のテキスト式を引数として受け付けます。
- %SUPERQ 関数は、%NRBQUOTE 関数と同じ文字をマスクします。ただし、%SUPERQ 関数は、マクロ変数の値に含まれているいかなるものも置換しません。%NRBQUOTE 関数は、結果をマスクする前に、引数に含まれているマクロ参照やマクロ変数値を置換しようとします。

例: 未置換のマクロ変数の値を渡す

次の例では、%SUPERQ 関数を使って、マクロ変数 MV1 および MV2 の値をマクロ変数 TESTMV1 および TESTMV2 に割り当てる前に、MV1 および MV2 の値に含まれるマクロ参照がマクロプロセッサによって置換されないようにしています。

```
data _null_;
call symput('mv1','Smith&Jones');
call symput('mv2','%macro abc;');
run;
%let testmv1=%superq(mv1);
%let testmv2=%superq(mv2);
%put Macro variable TESTMV1 is &testmv1;
%put Macro variable TESTMV2 is &testmv2;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
Macro variable TESTMV1 is Smith&Jones
Macro variable TESTMV2 is %macro abc;
```

変数 TESTMV1 および TESTMV2 の値は、それぞれ変数 MV1 および MV2 の元の値の画像であると見なすことができます。%PUT ステートメントは、そのような画像をテキストで出力します。マクロプロセッサは置換を行いません。マクロプロセッサは、未置換の参照**&JONES** に関する警告メッセージや、%LET ステートメント内部でのマクロ定義の開始に関するエラーメッセージを発行しません。

%SYMEXIST 関数

マクロ変数が存在するかどうかを示す値を返します。

種類: マクロ関数

構文

%SYMEXIST(*macro-variable-name*)

必須引数

macro-variable-name
マクロ変数名か、またはマクロ変数名を生成するテキスト式を指定します。

詳細

%SYMEXIST 関数は、指定されたマクロ変数を、かっこで囲まれたローカルシンボルテーブルで検索し、次にグローバルシンボルテーブルを検索して、検索結果に応じて次のいずれかの値を返します。

- `1` マクロ変数が見つかった場合
- `0` マクロ変数が見つからなかった場合

例: %SYMEXIST マクロ関数の使用

次の例では、%IF %THEN %ELSE マクロステートメントを使用して、%SYMEXIST 関数が返す値 `1` および `0` を、それぞれ値 **`TRUE`** および **`FALSE`** に変換しています。

```
%global x;
%macro test;
%local y;
%if %symexist(x) %then %put %nrstr(%symexist(x)) = TRUE;
%else %put %nrstr(%symexist(x)) = FALSE;
%if %symexist(y) %then %put %nrstr(%symexist(y)) = TRUE;
%else %put %nrstr(%symexist(y)) = FALSE;
%if %symexist(z) %then %put %nrstr(%symexist(z)) = TRUE;
%else %put %nrstr(%symexist(z)) = FALSE;
%mend test;
%test;
```

前述のプログラムを実行すると、次の行が SAS ログに出力されます。

```
%symexist(x) = TRUE
%symexist(y) = TRUE
%symexist(z) = FALSE
```

%SYMGLOBL 関数

マクロ変数のスコープがグローバルであるかどうかを示す値を返します。

種類: マクロ関数

構文

%SYMGLOBL(*macro-variable-name*)

必須引数

macro-variable-name
マクロ変数名か、またはマクロ変数名を生成するテキスト式を指定します。

詳細

%SYMGLOBL 関数は、かっこで囲まれたスコープを検索して同じ名前のマクロ変数が存在するかどうかを調べ、そのマクロ変数がグローバルシンボルテーブル内に存在する場合は値 `1` を返し、それ以外の場合は `0` を返します。グローバル/ローカルシンボルテーブルやマクロ変数のスコープに関する詳細は、“マクロ変数のスコープ” (45 ページ) を参照してください。

例: %SYMGLOBL マクロ関数の使用

次の例では、%IF %THEN %ELSE マクロステートメントを使用して、%SYMGLOBL 関数が返す値 `1` および `0` を、それぞれ **`TRUE`** および **`FALSE`** に変換しています。

```
%global x;
%macro test;
%local y;
%if %symglobl(x) %then %put %nrstr(%symglobl(x)) = TRUE;
%else %put %nrstr(%symglobl(x)) = FALSE;
%if %symglobl(y) %then %put %nrstr(%symglobl(y)) = TRUE;
%else %put %nrstr(%symglobl(y)) = FALSE;
%if %symglobl(z) %then %put %nrstr(%symglobl(z)) = TRUE;
%else %put %nrstr(%symglobl(z)) = FALSE;
%mend test;
%test;
```

前述のプログラムを実行すると、次の行が SAS ログに出力されます。

```
%symglobl(x) = TRUE
%symglobl(y) = FALSE
%symglobl(z) = FALSE
```

%SYMLOCAL 関数

マクロ変数のスコープがローカルであるかどうかを示す値を返します。

種類: マクロ関数

構文

%SYMLOCAL(*macro-variable-name*)

必須引数

macro-variable-name
マクロ変数名か、またはマクロ変数名を生成するテキスト式を指定します。

詳細

%SYMLOCAL 関数は、かっこで囲まれたスコープを検索して同じ名前のマクロ変数が存在するかどうかを調べ、そのマクロ変数がグローバルシンボルテーブル内に存在する場合は値 1 を返し、それ以外の場合は 0 を返します。グローバル/ローカルシンボルテーブルやマクロ変数のスコープに関する詳細は、“マクロ変数のスコープ” (45 ページ) を参照してください。

例: %SYMLOCAL マクロ関数の使用

次の例では、%IF %THEN %ELSE マクロステートメントを使用して、%SYMLOCAL 関数が返す値 1 および 0 を、それぞれ **TRUE** および **FALSE** に変換しています。

```
%global x;
%macro test;
%local y;
%if %symlocal(x) %then %put %nrstr(%symlocal(x)) = TRUE;
%else %put %nrstr(%symlocal(x)) = FALSE;
%if %symlocal(y) %then %put %nrstr(%symlocal(y)) = TRUE;
%else %put %nrstr(%symlocal(y)) = FALSE;
%if %symlocal(z) %then %put %nrstr(%symlocal(z)) = TRUE;
%else %put %nrstr(%symlocal(z)) = FALSE;
%mend test;
%test;
```

前述のプログラムを実行すると、次の行が SAS ログに出力されます。

```
%symlocal(x) = FALSE
%symlocal(y) = TRUE
%symlocal(z) = FALSE
```

%SYSEVALF 関数

浮動小数点演算を使用して演算式や論理式を評価します。

種類: マクロ関数

参照項目: “%EVAL 関数” (247 ページ)

構文

%SYSEVALF(*expression*<, *conversion-type*>)

必須引数

expression
評価する演算式または論理式を指定します。

conversion-type

変換後の値の型を指定します。%SYSEVALF の戻り値は、ここに指定された値の型に変換されます。変換後の値は、その型の値を必要とする他の式で使用できます。*conversion-type* には次のいずれかを指定できます。

BOOLEAN

次のいずれかを返します。

- 式の結果がゼロまたは欠損値である場合、0
- 結果がそれ以外の場合、1

例:

```
%sysevalf(1/3,boolean) /* returns 1 */
%sysevalf(10+.,boolean) /* returns 0 */
```

CEIL

式の結果に等しいかまたはそれより大きい最小の整数を表す文字値を返します。ただし、結果の値と、それに最も近い整数との差の絶対値が 10^{-12} 以下である場合、この関数は結果に最も近い整数を表す文字を返します。欠損値を含んでいる式の場合、そのことを知らせるメッセージと共に欠損値を返します。

```
%sysevalf(1 + 1.1,ceil) /* returns 3 */
%sysevalf(-1 -2.4,ceil) /* returns -3 */
%sysevalf(-1 + 1.e-11,ceil) /* returns 0 */
%sysevalf(10+.) /* returns . */
```

FLOOR

式の結果に等しいかまたはそれより小さい最大の整数を表す文字値を返します。ただし、結果の値と、それに最も近い整数との差の絶対値が 10^{-12} である場合、この関数は結果に最も近い整数を表す文字を返します。欠損値を含む式の場合、欠損値が返されます。

```
%sysevalf(-2.4,floor) /* returns -3 */
%sysevalf(3,floor) /* returns 3 */
%sysevalf(1.-1.e-13,floor) /* returns 1 */
%sysevalf(.,floor) /* returns . */
```

INTEGER

結果の整数部を表す文字値を返します(小数部を切り捨てます)。ただし、結果の値と、それに最も近い整数との差の絶対値が 10^{-12} 以下である場合、この関数は結果に最も近い整数を表す文字を返します。式の結果が正数である場合、INTEGER は FLOOR と同じ結果を返します。式の結果が負数である場合、INTEGER は CEIL と同じ結果を返します。欠損値を含む式の場合、欠損値が返されます。

```
%put %sysevalf(2.1,integer); /* returns 2 */
%put %sysevalf(-2.4,integer); /* returns -2 */
%put %sysevalf(3,integer); /* returns 3 */
%put %sysevalf(-1.6,integer); /* returns -1 */
%put %sysevalf(1.-1.e-13,integer); /* returns 1 */
```

詳細

%SYSEVALF 関数は、浮動小数点演算を実行し、出力形式 BEST32.を使ってフォーマット化した値を返します。評価結果は常にテキストとなります。%SYSEVALF 関数は、浮動小数点や欠損値を含んでいる論理式を評価できる唯一のマクロ関数です。引数 conversion-type を指定すると、%SYSEVALF 関数が次のいずれかの値を返す場合に発生する問題を回避できます。

- 欠損値または浮動小数点数値を生成するマクロ式

- 整数値を必要とする他のマクロ式で使用されるマクロ変数

%SYSEVALF の引数に演算子が含まれておらず、かつ conversion-type も指定されていない場合、指定した引数がそのまま返されます。

SAS マクロ言語による式の評価についての詳細は、“マクロ式” (73 ページ)を参照してください。

比較

- %SYSEVALF 関数は浮動小数点数をサポートします。一方、%EVAL 関数は整数演算のみを実行します。
- 浮動小数点式を評価するマクロでは、%SYSEVALF マクロ関数を使用する必要があります。マクロ式を評価する場合、マクロプロセッサは自動的に%EVAL 関数を使用します。

例: 浮動小数点評価の例

次に示すマクロ FIGUREIT は、%SYSEVALF 関数の戻り値を様々な型に変換します。

```
%macro figureit(a,b);
%let y=%sysevalf(&a+&b);
%put The result with SYSEVALF is: &y;
%put The BOOLEAN value is: %sysevalf(&a +&b, boolean);
%put The CEIL value is: %sysevalf(&a +&b, ceil);
%put The FLOOR value is: %sysevalf(&a +&b, floor);
%put The INTEGER value is: %sysevalf(&a +&b, int);
%mend figureit;
%figureit(100,1.597)
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
The result with SYSEVALF is: 101.597
The BOOLEAN value is: 1
The CEIL value is: 102
The FLOOR value is: 101
The INTEGER value is: 101
```

%SYSMACEXEC 関数

マクロの実行ステータスを示す値を返します。

種類: マクロ関数

構文

%SYSMACEXEC(*macro_name*)

必須引数

macro_name
マクロ名か、またはマクロ名を生成するテキスト式を指定します。

詳細

%SYSMACEXEC 関数は、指定されたマクロが現在実行中である場合に数値 1 を返します。同マクロが実行中でない場合、%SYSMACEXEC 関数は数値 0 を返します。

%SYSMACEXIST 関数

指定のマクロ定義が WORK.SASMACR カタログ内に存在するかどうかを示す値を返します。それ以外の場合、値 0 を返します。

種類: マクロ関数

構文

%SYSMACEXIST(*macro-name*)

必須引数

macro-name
マクロ名か、またはマクロ名を生成するテキスト式を指定します。

詳細

%SYSMACEXIST 関数は、指定のマクロ定義が WORK.SASMACR カタログ内に存在する場合、値 1 を返します。マクロ定義が存在しない場合、値 0 を返します。

%SYSMEXECDEPTH 関数

%SYSMEXECDEPTH 関数の呼び出し点からのネストの深さを返します。

種類: マクロ関数

ヒント: %SYSMEXECDEPTH 関数と%SYSMEXECNAME 関数は組み合わせて使用できるように実装されていますが、組み合わせて使用することが必須ではありません。

参照項目: %SYSMEXECNAME 関数

構文

%SYSMEXECDEPTH

詳細

現在実行中のマクロのネストレベルを取得するには、%SYSMEXECDEPTH 関数を使用します。この関数は、ネストされたマクロ呼び出しにおける指定のマクロのネストの深さを表す数字を返します。%SYSMEXECDEPTH 関数は次の値を返します。

0 オープンコード

>0 ネストレベル

詳細については、次に示す例を参照してください。

```
8 %macro A;
```

```
9 %put %sysmexecdepth;
10 %mend A; /* The macro execution depth
of a macro called from open code */
11 %A; /* is one */
1
12
13 %macro B;
14 %put %nrstr(%%)sysmexecdepth=%sysmexecdepth;
15 %put %nrstr(%%)sysmexecname(1)=%sysmexecname(1);
16 %put %nrstr(%%)sysmexecname(2)=%sysmexecname(2);
17 %put %nrstr(%%)sysmexecname(0)=%sysmexecname(0);
18 %put %nrstr(%%)sysmexecname(%nrstr(%%)sysmexecdepth-1)=
%sysmexecname(%sysmexecdepth-1);
19 %mend B;
20
21 %macro C;
22 %B;
23 %mend;
24 %C;
%sysmexecdepth=2
%sysmexecname(1)=C
%sysmexecname(2)=B
%sysmexecname(0)=OPEN CODE
%sysmexecname(%sysmexecdepth-1)=C
25
26 %macro level1;
27 %level2;
28 %mend;
29 %macro level2;
30 %level3;
31 %mend;
32 %macro level3;
33 %level4;
34 %mend;
35 %macro level4;
36 %do i = %sysmexecdepth+1 %to -1 %by -1;
37 %put %nrstr(%%)sysmexecname(&i)=%sysmexecname(&i);
38 %end;
39 %mend;
40
41 %level1;
WARNING: Argument 1 to %SYSMEXECNAME function is out of range.
%sysmexecname(5)=
%sysmexecname(4)=LEVEL4
%sysmexecname(3)=LEVEL3
%sysmexecname(2)=LEVEL2
%sysmexecname(1)=LEVEL1
%sysmexecname(0)=OPEN CODE
WARNING: Argument 1 to %SYSMEXECNAME function is out of range.
%sysmexecname(-1)=
42
```

- マクロ **A** はマクロ **B** を呼び出します。マクロ **C** はマクロ **B** を呼び出します。マクロ **C** 内に配置された%SYSMEXECDEPTH 関数の呼び出しは、値 **2** をマクロ **B** に対して返します。

- マクロ c で、同マクロを呼び出したマクロ名を知りたい場合、%SYSMEXECNAME 関数を**%SYSMEXECNAME(%SYSMEXECDEPTH-1)として呼び出します。**
(%SYSMEXECNAME 関数の引数である *n* の値は、元のネストレベルの値である%SYSMEXECDEPTH の戻り値から 1 を引いた値になります)。この%SYSMEXECNAME 関数の呼び出しは、値 B を返します。

%SYSMEXECNAME 関数

要求されたネストレベルで実行しているマクロ名を返します。

種類: マクロ関数

ヒント: %SYSMEXECDEPTH 関数と%SYSMEXECNAME 関数は組み合わせて使用できるように実装されていますが、組み合わせて使用することが必須ではありません。

参照項目: %SYSMEXECDEPTH 関数

構文

%SYSMEXECNAME (*n*)

必須引数

n
マクロ名を要求するネストレベルを指定します。

0 オープンコード

>0 ネストレベル

詳細

%SYSMEXECNAME 関数は、ネストレベル *n* で実行しているマクロの名前を返します。次の例では 3 つのシナリオが示されています。

- *n = 0* の場合、**open code** が返されます。
- *n >%SYSMEXECDEPTH* の場合、ヌル文字列が返され、警告診断メッセージが SAS ログに出力されます。
- *n<0* の場合ヌル文字列が返され、警告診断メッセージが SAS ログに出力されます。

```
3 %put %sysmexecdepth; /* The macro execution depth of
Open Code is zero */
0
4 %put %sysmexecname(%sysmexecdepth);
OPEN CODE
5 %put %sysmexecname(%sysmexecdepth + 1);
WARNING: Argument 1 to %SYSMEXECNAME function is out of range.

6 %put %sysmexecname(%sysmexecdepth - 1);
WARNING: Argument 1 to %SYSMEXECNAME function is out of range.
```

%SYSFUNC 関数と%QSYSFUNC 関数

SAS 関数またはユーザー作成の関数を実行します。

種類: マクロ関数

ヒント: %SYSFUNC 関数と%QSYSFUNC 関数は最大 32 文字の SAS 関数名をサポートします。

構文

%SYSFUNC (*function*(*argument-1* <*...argument-n*>)<, *format*>)

%QSYSFUNC (*function*(*argument-1* <*...argument-n*>)<, *format*>)

必須引数

function
実行する関数名を指定します。SAS 関数、SAS/TOOLKIT ソフトウェアを使って作成した関数、または(“FCMP プロシジャ” (*Base SAS プロシジャガイド*)マクロ関数は指定できません。

%SYSFUNC 関数および%QSYSFUNC 関数では、すべての SAS 関数(ただし、表 17.2 (274 ページ)に記載されているものは除く)を使用できます。

単一の%SYSFUNC 関数で使用する場合、関数のネストは行えません。ただし、次の例のように、%SYSFUNC 関数の呼び出しはネストできます。

```
%let x=%sysfunc(trim(%sysfunc(left(&num))));
```

SAS 6.12 で導入された%SYSFUNC 関数で使用できる SAS 関数の構文については、%SYSFUNC 関数で使用する関数の構文 (373 ページ) をご覧ください。

argument-1 <...argument-n>
function が使用する 1 つ以上の引数を指定します。各 argument には、関数の引数を生成するマクロ変数参照やテキスト式を指定できます。*argument* が次に示すような特殊文字やニーモニック演算子を含んでいる場合、%QSYSFUNC を使用します。

format
function の結果に適用されるオプションの出力形式を指定します。SAS 提供の出力形式、FORMAT プロシジャにより生成される出力形式、または SAS/TOOLKIT を使って作成された出力形式を指定できます。*format* のデフォルト値はありません。*format* を指定しない場合、SAS マクロ機能は結果に対して *format* 操作を実行せず、*function* のデフォルト値を使用します。

詳細

%SYSFUNC 関数はマクロ関数であるため、DATA ステップ関数の場合とは異なり、文字値を引用符で囲む必要はありません。たとえば、OPEN 関数を単独で指定する場合、その引数を引用符で囲んで指定しますが、%SYSFUNC 関数の内部で使用する場合、引用符は必要はありません。次のステートメントは、この違いを示しています。

- `dsid=open("sasuser.houses","i");`
- `dsid=open("&mydata","&mode");`
- `%let dsid = %sysfunc(open(sasuser.houses,i));`

- `%let dsid=%sysfunc(open(&mydata,&mode));`

%SYSFUNC 関数の内部にある DATA ステップ関数の引数はすべてカンマで区切る必要があります。OF というワードで始まる引数リストは使用できません。

注: %SYSFUNC 関数の引数は、SAS マクロ言語の規則に従って評価されます。これには、関数名および関数の引数リストの両者が含まれます。特に、引数位置に空を指定すると、NULL 引数ではなく、長さがゼロの引数が生成されます。

%SYSFUNC 関数は、結果に含まれる特殊文字やニーモニック演算子をマスクしません。一方、%QSYSFUNC 関数は、結果に含まれる次の特殊文字とニーモニック演算子をマスクします。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

%SYSFUNC 関数または%QSYSFUNC 関数により呼び出される関数が数値引数を必要とする場合、マクロ機能は引数を数値に変換します。%SYSFUNC 関数および%QSYSFUNC 関数は、それらが実行する関数が浮動小数点数をサポートする場合には浮動小数点数を返します。

***表 17.2** %SYSFUNC 関数および%QSYSFUNC 関数で使用できない SAS 関数*

すべての変数情報関数	ALLCOMB	ALLPERM
DIF	DIM	HBOUND
IORCMSG	INPUT	LAG
LBOUND	LEXCOMB	LEXCOMBI
LEXPERK	LEXPERM	MISSING
PUT	RESOLVE	SYMGET

注: INPUT 関数や PUT 関数は%SYSFUNC 関数および%QSYSFUNC 関数では使用できないため、代わりに INPUTN 関数、INPUTC 関数、PUTN 関数、PUTC 関数を使用してください。

注: 変数情報関数には、VNAME 関数や VLABEL 関数が含まれます。変数情報関数の完全な一覧については、*SAS 関数と CALL ルーチン: リファレンス*の関数と CALL ルーチンの定義を参照してください。

注意:

SAS 関数が返す値は切り詰められる場合があります。 マクロ変数により返される値は DATA ステップにより定められている長さには制限されませんが、SAS 関数により返される値はこの制限を受けます。

比較

%QSYSFUNC 関数は、%NRBQUOTE 関数と同じ文字をマスクします。

例

例 1: TITLE ステートメントでの現在の日付のフォーマット

次の例では、DATE 関数と WORDDATE.出力形式を使用して、現在の日付を含む TITLE ステートメントをフォーマットしています。

```
title "%sysfunc(date(),worddate.) Absence Report";
```

2008 年 7 月 18 日にプログラムを実行した場合、次の TITLE ステートメントが生成されます。

```
title "July 18, 2008 Absence Report"
```

例 2: %SYSFUNC により生成された値のフォーマット

次に示すマクロ TRY は、PUTN 関数と CATEGORY.入力形式を使用して引数 PARM の値を変換します。

```
proc format;
value category
Low-<0 = 'Less Than Zero'
0 = 'Equal To Zero'
0<-high = 'Greater Than Zero'
other = 'Missing';
run;
%macro try(parm);
%put &parm is %sysfunc(putn(&parm,category.));
%mend;
%try(1.02)
%try(.)
%try(-.38)
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
1.02 is Greater Than Zero
. is Missing
-.38 is Less Than Zero
```

例 3: 文字の変換

次の例では、%SYSFUNC 関数で TRANSLATE 関数を実行することにより、文字列内に含まれている文字 N を文字 P に変換します。

```
%let string1 = V01N01-V01N10;
%let string1 = %sysfunc(translate(&string1,P, N));
%put With N translated to P, V01N01-V01N10 is &string1;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
With N translated to P, V01N01-V01N10 is V01P01-V01P10
```

例 4: SAS データセットの存在の確認

次に示すマクロ CHECKDS は、%SYSFUNC 関数を使用して EXIST 関数を実行することにより、データセットが存在するかどうかチェックします。

```
%macro checkds(dsn);
%if %sysfunc(exist(&dsn)) %then
%do;
proc print data=&dsn;
run;
%end;
```

```
%else
%put The data set &dsn does not exist.;
%mend checkds;
%checkds(sasuser.houses)
```

このプログラムを実行すると、次のステートメントが生成されます。

```
PROC PRINT DATA=SASUSER.HOUSES;
RUN;
```

例 5: データセットの変数とオブザベーションの数の決定

1 つの SAS データセット内に存在する変数とオブザベーションの数を取得するために、これまで多くのソリューションが作成されてきました。過去に作成されたソリューションのほとんどは、NULL_ DATA ステップ、NOBS=オプション付きの SET ステートメント、配列を組み合わせて使用することで、この情報を取得していました。現在は、OPEN 関数および ATTRN 関数を使用することにより、ステップ境界条件に干渉することなく、この情報を素早く取得できます。

```
%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)
```

このプログラムを実行すると、次のメッセージが SAS ログに出力されます。

```
sasuser.houses has 6 variable(s) and 15 observation(s).
```

%SYSGET 関数

指定された動作環境変数の値を返します。

種類: マクロ関数

構文

%SYSGET(*environment-variable*)

必須引数

environment-variable

環境変数名を指定します。*environment-variable* に指定する変数名の大文字小文字は、動作環境に保存されている変数の大文字小文字と一致していなければなりません。

詳細

%SYSGET 関数は、値を文字列として返します。値が切り捨てられた場合や、変数が動作環境で定義されていない場合、%SYSGET 関数は SAS ログに警告メッセージを表示します。

%SYSGET 関数が返す値は、それ以降のアクションを実施するかどうかを決定するための条件として、または実行する SAS プログラムの一部として使用できます。これにより、たとえば、プログラムで特定の処理を制限することや、ユーザーに固有のコマンドを発行することが可能となります。

詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

例: UNIX 環境での%SYSGET 関数の使用例

次の例は、UNIX オペレーティングシステム上でのユーザー ID を返すものです。

```
%let person=%sysget(USER);
%put User is &person;
```

ユーザー ABCDEF がこれらのステートメントを実行すると、次の行が SAS ログに出力されます。

```
User is abcdef
```

%SYSPROD 関数

現在のサイトで SAS ソフトウェア製品がライセンスされているかどうかを報告します。

種類: マクロ関数

参照項目: "%SYSEXEC ステートメント" (323 ページ)"SYSSCP 自動マクロ変数と SYSSCPL 自動マクロ変数" (216 ページ)および"SYSVER 自動マクロ変数" (222 ページ)

構文

%SYSPROD (*product*)

必須引数

product

SAS 製品のコードを生成する文字列またはテキスト式を指定します。よく使われるコードは次の通りです。

表 17.3 *よく使われるコード*

AF	CPE	GRAPH	PH-CLINICAL
ASSIST	EIS	IML	QC
BASE	ETS	INSIGHT	SHARE
CALC	FSP	LAB	STAT
CONNECT	GIS	OR	TOOLKIT

その他の SAS ソフトウェア製品のコードについては、オンサイトの SAS サポート担当者にお尋ねください。

詳細

%SYSPROD 関数は次の値を返します。

表 17.4 %SYSPROD 関数の戻り値と説明

値	説明
1	その SAS 製品はライセンスされています。
0	その SAS 製品はライセンスされていません。
−1	その製品は SAS ソフトウェアではありません(製品コードのスペルが誤っていた場合など)。

例: GPLOT プロシジャの実行前に SAS/GRAPH がインストールされているかどうか確認する

次の例では、%SYSPROD 関数を使用して、SAS/GRAPH ソフトウェアがインストールされているかどうかに応じて、PROC GPLOT ステートメントまたは PROC PLOT ステートメントのどちらを実行するかを判定しています。

```
%macro runplot(ds);
%if %sysprod(graph)=1 %then
%do;
title "GPLOT of %upcase(&ds)";
proc gplot data=&ds;
plot style*price / haxis=0 to 150000 by 50000;
run;
quit;
%end;
%else
%do;
title "PLOT of %upcase(&ds)";
proc plot data=&ds;
plot style*price;
run;
quit;
%end;
%mend runplot;
%runplot(sasuser.houses)
```

このプログラムを実行すると、SAS/GRAPH がインストールされている場合には、次のステートメントが生成されます。

```
TITLE "GPLOT of SASUSER.HOUSES";
PROC GPLOT DATA=SASUSER.HOUSES;
PLOT STYLE*PRICE / HAXIS=0 TO 150000 BY 50000;
RUN;
```

%UNQUOTE 関数

マクロの実行時に、値に含まれているすべての特殊文字とニーモニック演算子をアンマスクします。

種類: マクロ関数

参照項目: “%BQUOTE 関数と%NRBQUOTE 関数” (246 ページ)、“%NRBQUOTE 関数” (250 ページ)、“%NRQUOTE 関数” (251 ページ)、“%NRSTR 関数” (251 ページ)、“%QUOTE 関数と%NRQUOTE 関数” (252 ページ)、“%STR 関数と%NRSTR 関数” (258 ページ)、および“%SUPERQ 関数” (263 ページ)

構文

%UNQUOTE (*character string* | *text expression*)

詳細

%UNQUOTE 関数は値をアンマスクすることにより、その値に含まれている特殊文字がテキストとしてではなく、マクロ言語要素として解釈されるようにします。
%UNQUOTE 関数の最も重要な機能は、先行するマクロクォーティング関数によりそのトークン化が変更されていた値の正常なトークン化を復元することです。
%UNQUOTE 関数はマクロの実行時に有効となります。

詳細は“マクロクォーティング” (82 ページ)を参照してください。

例: %UNQUOTE 関数を使用した値のアンマスク

この例では、マクロクォーティング関数を使ってマクロ変数の値を割り当て、その後、DATA ステップで同変数を参照した場合に発生する可能性のある問題に対処しています。値が SAS コンパイラに到達する前にアンマスクされていない場合、DATA ステップは正しくコンパイルされず、エラーメッセージが出力されます。一部のマクロ関数は自動的に値をアンマスクしますが、変数はこれらの関数によって処理されません。

次のプログラムを実行すると、TESTVAL の値が SAS コンパイラに到達した時点でマスクされたままになっているため、エラーメッセージが SAS ログに出力されます。

```
%let val = aaa;
%let testval = %str(%'&val%');
data _null_;
val = &testval;
put 'VAL =' val;
run;
```

次のプログラムは、%UNQUOTE 関数により TESTVAL の値をアンマスクしているため、正しく動作します。

```
%let val = aaa;
%let testval = %str(%'&val%');
data _null_;
val = %unquote(&testval);
put 'VAL =' val;
run;
```

このプログラムを実行すると次の行が SAS ログに出力されます。

```
VAL=aaa
```

%UPCASE 関数と%QUPCASE 関数

値を大文字に変換します。

種類: マクロ関数

参照項目: “%LOWCASE 自動呼び出しマクロと%QLOWCASE 自動呼び出しマクロ” (182 ページ)、“%NRBQUOTE 関数” (250 ページ)、および“%QLOWCASE 自動呼び出しマクロ” (184 ページ)

構文

%UPCASE (*character string* | *text expression*)

%QUPCASE(*character string* | *text expression*)

詳細

%UPCASE 関数および%QUPCASE 関数は、引数に含まれている小文字を大文字に変換します。%UPCASE 関数は、引数がそれまでマクロクォーティング関数によりマスクされていた場合であっても、同関数が返す結果内で特殊文字やニーモニック演算子をマスクしません。

次に示すような特殊文字やニーモニック演算子が引数に含まれている場合、%QUPCASE を使用します。%QUPCASE 関数は、結果に含まれる次の特殊文字とニーモニック演算子をマスクします。

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

%UPCASE 関数と%QUPCASE 関数は、文字値を比較する場合に役立ちます。マクロ機能は、文字値を比較する前に小文字から大文字への変換を自動的に行わないためです。

比較

- %QUPCASE 関数は、%NRBQUOTE 関数と同じ文字をマスクします。
- 文字を小文字に変換するには、%LOWCASE または%QLOWCASE 自動呼び出しマクロを使用します。

例

例 1: 比較する値を大文字に変換する

次に示すマクロ RUNREPT は、マクロ変数 MONTH に入力された値を文字列 DEC と比較します。大文字化した値が DEC に等しい場合、REPORTS.ENDYEAR という名前のデータセットに関して FSVIEW プロシジャを実行します。それ以外の場合、REPORTS データライブラリ内の対応する月名を持つデータセットに関して FSVIEW プロシジャを実行します。

```
%macro runrept(month);
%if %upcase(&month)=DEC %then
%str(proc fsview data=reports.endyear; run;);
%else %str(proc fsview data=reports.&month; run;);
%mend runrept;
```

このマクロを次のどの形式で呼び出した場合でも、同マクロ内の%IF 条件が true になります。

```
%runrept(DEC)
%runrept(Dec)
%runrept(dec)
```

例 2: %UPCASE 関数と%QUPCASE 関数の比較

次のステートメントは、%UPCASE 関数と%QUPCASE 関数により生成される結果を比較するものです。

```
%let a=begin;
%let b=%nrstr(&a);
%put UPCASE produces: %upcase(&b);
%put QUPCASE produces: %qupcase(&b);
```

これらのステートメントを実行すると、次の行が SAS ログに表示されます。

```
UPCASE produces: begin
QUPCASE produces: &A
```

18 章

マクロの SQL 句

マクロの SQL 句

構造化照会言語(SQL)は、データベースやリレーショナルテーブル内のデータの取り出しや更新を行うために広く使用されている標準化された言語です。

ディクショナリ

INTO 句

SQL プロシジャにより生成された値をマクロ変数に割り当てます。

種類: SELECT ステートメント、SQL プロシジャ

構文

INTO : *macro-variable-specification-1* < ..., : *macro-variable-specification-n* >

必須引数

macro-variable-specification
作成または更新するマクロ変数を 1 つ以上指定します。各マクロ変数名の先頭にはコロン(:)を付けます。マクロ変数は、次の形式で指定できます。

: *macro-variable*
1 つまたは複数のマクロ変数を指定します。値がマクロ変数に保存される際に、値の先頭および末尾にある空白は削除されません。

```
select style, sqfeet
into :type, :size
from sasuser.houses;
```

:*macro-variable-1* − : *macro-variable-n* <NOTRIM>
:*macro-variable-1* THROUGH : *macro-variable-n* <NOTRIM>
:*macro-variable-1* THRU : *macro-variable-n* <NOTRIM>
マクロ変数の番号付きリストを指定します。値がマクロ変数に保存される際に、値の先頭および末尾にある空白は削除されます。先頭および末尾の空白を削除したくない場合、NOTRIM オプションを使用します。NOTRIM オプションは、この形式の INTO 句に含まれる各要素に対して個々に適用されます。すなわち、同オプションは 1 つの要素に対してのみ適用され、それ以外の要素には適用されません。

```
select style, sqfeet
into :type1 - :type4 notrim, :size1 - :size3
from sasuser.houses;
```

:*macro-variable* SEPARATED BY '*characters* ' <NOTRIM>
1 つの列のすべての値を含む 1 つのマクロ変数を指定します。このリスト内の値は、1 つまたは複数の *characters* で区切ります。この形式の INTO 句は、項目のリストを構築する場合に役立ちます。値がマクロ変数に保存される際に、値の先頭および末尾にある空白は削除されます。先頭および末尾の空白を削除したくない場合、NOTRIM オプションを使用します。一意の列(変数)値のみを保存するには、次に示すように SELECT ステートメントで DISTINCT オプションを使用します。

```
select distinct style
into :types separated by ','
from sasuser.houses;
```

詳細

SELECT ステートメントの INTO 句は、計算結果やデータ列(変数)の値をマクロ変数に割り当てます。マクロ変数が存在しない場合、INTO 句はそれを自動的に作成します。SQL プロシジャのマクロ変数 SQLOBS をチェックすることで、SELECT ステートメントにより生成される行(オブザベーション)の数を確認できます。

INTO 句は、SELECT ステートメントの外側クエリでのみ使用可能であり、サブクエリでは使用できません。INTO 句は、テーブルの作成時(CREATE TABLE)やビューの作成時(CREATE VIEW)には使用できません。

INTO 句により作成されたマクロ変数は、%LET ステートメントのスコープ規則に従います。

INTO 句により割り当てられた値は、BEST12.出力形式を使用します。

比較

SQL プロシジャ内で、INTO 句は SYMPUT ルーチンと同様の役割を実行します。

例

例 1: 宣言されたマクロ変数に列の値を保存する

次の例は、データセット SASUSER.HOUSES に基づいて、テーブルの最初の行(またはデータセット内のオブザベーション)に含まれている列(変数)STYLE および SQFEET の値を、マクロ変数 TYPE および SIZE に保存します。%LET ステートメントは、変数 TYPE の値から末尾の空白を取り除き、変数 SIZE の値から先頭の空白を取り除きます。INTO 句を次のような形式で使用した場合、デフォルトではこれらの空白は取り除かれません。

```
proc sql noprint;
```

```
select style, sqfeet
into :type, :size
from sasuser.houses;
%let type=&type;
%let size=&size;
%put The first row contains a &type with &size square feet.;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
The first row contains a RANCH with 1250 square feet.
```

例 2: マクロ変数のリストに行の値を保存する

次の例では、TYPE1～TYPE4 および SIZE1～SIZE4 という 2 つのマクロ変数のリストを作成し、SASUSER.HOUSES データセットの最初の 4 つの行(オブザベーション)に含まれている値を、これらの変数リスト内に保存します。変数リスト TYPE1～TYPE4 に対して NOTRIM オプションが指定されているため、これらの値の末尾の空白は保持されたままになります。

```
proc sql noprint;
select style, sqfeet
into :type1 - :type4 notrim, :size1 - :size4
from sasuser.houses;
%macro putit;
%do i=1 %to 4;
%put Row&i: Type=**&&type&i** Size=**&&size&i**;
%end;
%mend putit;
%putit
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
Row1: Type=**RANCH ** Size=**1250**
Row2: Type=**SPLIT ** Size=**1190**
Row3: Type=**CONDO ** Size=**1400**
Row4: Type=**TWOSTORY** Size=**1810**
```

例 3: 1 つのマクロ変数にすべての行の値を保存する

次の例では、列(変数)STYLE のすべての値をマクロ変数 TYPES に保存します。これらの値はカンマと空白で区切られます。

```
proc sql;
select distinct quote(style)
into :types separated by ', '
from sasuser.houses;
%put Types of houses=&types.;
```

このプログラムを実行すると、次の行が SAS ログに出力されます。

```
Types of houses=CONDO, RANCH, SPLIT, TWOSTORY
```

19 章

マクロステートメント

マクロステートメント

マクロ言語ステートメントは、マクロプロセッサに特定の操作を実行するよう命令します。マクロ言語ステートメントは、キーワードの文字列、SAS 名、および特殊文字と演算子から成り、セミコロンで終わります。マクロステートメントの末尾にはセミコロンを付けます。一部のマクロ言語ステートメントは、マクロ定義の内部でのみ使用できます。それ以外のマクロ言語ステートメントは、SAS セッションや SAS ジョブの任意の場

所で、マクロ定義の内外にかかわらず使用できます。SAS セッションや SAS ジョブにおけるマクロ定義の外側のことをオープンコードと呼びます。

ディクショナリ

%ABORT ステートメント

現在の DATA ステップ、SAS ジョブ、SAS セッションと共に実行しているマクロを停止します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

構文

%ABORT <ABEND | CANCEL <FILE> | RETURN | <*n*> > ;

必須引数

ABEND
現在のマクロおよび SAS ジョブ(または SAS セッション)を異常終了させます。結果は動作モードや動作環境により異なります。

- バッチモードおよび非対話モードの場合
  - 処理を即座に停止します。
  - %ABORT マクロステートメントの ABEND オプションにより実行が停止されたことを知らせるエラーメッセージを SAS ログに送信します。
  - 後続ステートメントの実行や構文チェックは行いません。
  - オペレーティングシステムに制御を戻します。これ以降の処理は、お使いのオペレーティングシステムによる異常終了ジョブの取り扱い方法に基づいて実施されます。
- ウィンドウ環境および対話型ラインモードの場合
  - マクロ、ウィンドウ環境、対話型ラインモードによる処理を即座に停止し、オペレーティングシステムに制御を戻します。

CANCEL <FILE>
現在サブミットされているステートメントを取り消します。結果は動作モードや動作環境により異なります。

バッチモードや非対話型モードの場合、CANCEL オプションを指定すると次のことが起こります。

- SAS プログラム全体および SAS システム全体が停止されます。
- エラーメッセージが SAS ログに出力されます。

ウィンドウ環境や対話型ラインモードの場合、CANCEL オプションを指定すると次のことが起こります。

- 現在サブミットされているプログラムのみがクリアされます。

- それ以外のサブミット済みのプログラムは影響を受けません。
- エラーメッセージが SAS ログに出力されます。

ワークスペースサーバーやストアドプロセスサーバーの場合、CANCEL オプションを指定すると次のことが起こります。

- 現在サブミットされているプログラムのみがクリアされます。
- それ以外のサブミット済みのプログラムは影響を受けません。
- エラーメッセージが SAS ログに出力されます。

SAS IntrNet アプリケーションサーバーの場合、CANCEL オプションを指定すると次のことが起こります。

- 要求ごとに独立した実行が生成されます。この実行が要求コードをサブミットします。要求コード内に CANCEL オプションが含まれていると、現在サブミットされているコードはクリアされますが、実行や SAS セッションは停止されません。

FILE
autoexec ファイルまたは%INCLUDE ファイル内で CANCEL 引数のオプションとして指定した場合、autoexec ファイルまたは%INCLUDE ファイルの内容だけが%ABORT ステートメントによりクリアされます。サブミットされた他のソーステートメントは、autoexec ファイルまたは%INCLUDE ファイルの後に実行されます。

制限事項 CANCEL 引数は、SAS/SHARE、SAS/CONNECT、SAS/AF を使用ステートメントいる場合にはサブミットできません。

注意 **%ABORT CANCEL FILE オプションを%INCLUDE ファイル内で実行すると、すべてのオープンされているマクロはクローズされ、プログラムの次のソース行が読み込まれた時点で実行が再開されます。**

RETURN
現在のマクロおよび SAS ジョブ(または SAS セッション)を異常終了させます。結果は動作モードや動作環境により異なります。

- バッチモードおよび非対話モードの場合
  - 処理を即座に停止します。
  - %ABORT マクロステートメントの RETURN オプションにより実行が停止されたことを知らせるエラーメッセージを SAS ログに送信します。
  - 後続ステートメントの実行や構文チェックは行いません。
  - エラーを示すコンディションコードと共に、オペレーティングシステムに制御を戻します。
- ウィンドウ環境および対話型ラインモードの場合
  - マクロ、ウィンドウ環境、対話型ラインモードによる処理を即座に停止し、オペレーティングシステムに制御を戻します。

n
ユーザーによるコンディションコードの指定を可能にする整数です。

- CANCEL ステートメントと共に使用すると、この値が SYSINFO マクロ変数に格納されます。

- CANCEL ステートメントと共に使用しない場合、実行の停止時に、SAS システムはこの値をオペレーティングシステムに返します。*n* の値の範囲は、動作環境により異なります。

詳細

引数を指定しない場合、%ABORT マクロステートメントは、動作モードや動作環境に応じて次の結果を生成します。

- バッチモードおよび非対話モードの場合
  - 現在のマクロと DATA ステップの処理を停止し、エラーメッセージを SAS ログに出力します。SAS システムが%ABORT マクロステートメントをいつ検出したかに応じて、データセットに不完全な数のオブザベーションが含まれるか、またはデータセットにオブザベーションが一切含まれないかのいずれかになります。
  - OBS=システムオプションの値を 0 に設定します。
  - SAS ジョブの残りの限定的な処理を続行します。これにはマクロステートメントの実行、システムオプションステートメントの実行、プログラムステートメントの構文チェックが含まれます。
- ウィンドウ環境
  - 現在のマクロと DATA ステップの処理を停止します。
  - %ABORT ステートメントの検出前に処理されたオブザベーションを含むデータセットを作成します。
  - %ABORT マクロステートメントにより DATA ステップが停止されたことを伝えるメッセージを SAS ログに出力します。
- 対話型ラインモード
  - 現在のマクロと DATA ステップの処理を停止します。それ以降の DATA ステップやプロシジャは正常に実行されます。

比較

%ABORT マクロステートメントは、SAS システムによる現在のマクロと DATA ステップの処理を停止します。それ以降のアクションは、次に示す条件により決定されます。

- ユーザーが SAS ステートメントをサブミットするのに使用した方法
- ユーザーが%ABORT ステートメントに指定した引数
- ユーザーの動作環境

%ABORT マクロステートメントは、通常、エラー状況が発生した場合に処理を停止するよう設計された%IF-%THEN マクロステートメントの句として記述されます。

注: ERRORABEND システムオプションが有効である場合、%ABORT マクロステートメントにより生成されるリターンコードは SAS システムにより無視されます。

注: %ABORT マクロステートメントを DATA ステップで実行すると、SAS システムは、同じ名前の既存のデータセットを置き換える場合に、その DATA ステップで作成されたデータセットを使用しません。

%*マクロコメントステートメント

コメントテキストを指定します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードでのみ使用可能

構文

%*commentary*;

必須引数

commentary
任意の長さの説明メッセージを指定します。

詳細

マクロコメントステートメントはマクロプログラムを説明する場合に使用します。マクロコメントステートメント内のテキストは定数テキストではないため、コンパイル済みマクロ内には保存されません。コメントステートメントはセミコロンで終了するため、コメント内にセミコロンを含める場合、そのセミコロンを引用符で囲む必要があります。マクロコメントステートメントを引用符で囲んだ場合、そのコメントステートメントは認識されません。

マクロコメントステートメントは完全なマクロステートメントであり、マクロ機能により処理されます。マクロコメントの内部にある引用符はペアとしてマッチする必要があります。

マクロステートメントがマクロ機能により処理されないようにするには、マクロ定義やオープンコード内でマクロステートメントマクロコメントステートメントを使うか、または /*commentary*/形式の SAS コメントを使用します。

比較

形式

```
*commentary;
```

または

```
comment commentary;
```

を持つ SAS コメントステートメントは、完全な SAS ステートメントです。この形式を持つコメントステートメントはトークナイザやマクロ機能により処理されるため、セミコロンやペアマッチしない引用符を同ステートメントに含めることはできません。形式

```
*commentary;
```

または

```
comment commentary;
```

を持つ SAS コメントステートメントは、コンパイル済みマクロ内に定数テキストとして保存されます。これらの 2 種類の SAS コメントステートメントは、コメント内にマクロステートメントが含まれている場合、それらのマクロステートメントをすべて実行します。このため、これらの SAS コメントステートメントをマクロ定義内では使用しないよう推奨します。

一方、形式

/*commentary*/

を持つ SAS コメントはトークン化されず、1 つの文字列として処理されます。この形式のコメントは、単一の空白が記述できる場所であればどこでも記述可能であり、セミコロンやペアマッチしない引用符を含むこともできます。形式

/*commentary*/

を持つ SAS コメントは、コンパイル済みマクロには保存されません。

例: 各種のコメントタイプの比較

次のプログラムは、データエラーをチェックするマクロ VERDATA を定義して呼び出すものです。このプログラムには、マクロコメントステートメントと、2 つの形式(***/*commentary*/***および****commentary;***を持つ SAS コメントステートメント が含まれています。

```
%macro verdata(in, thresh);
*%let thresh = 5;
/* The preceding SAS comment does not hide the %let statement
as does this type of SAS comment.
%let thresh = 6;
*/
%if %length(&in) > 0 %then %do;
%* infile given;
data check;
/* Jim's data */
infile &in;
input x y z;
* check data;
if x<&thresh or y<&thresh or z<&thresh then list;
run;
%end;
%else %put Error: No infile specified;
%mend verdata;
%verdata(ina, 0)
```

マクロ VERDATA を実行すると、次の行が生成されます。

```
DATA CHECK;
INFILE INA;
INPUT X Y Z;
* CHECK DATA;
IF X<5 OR Y<5 OR Z<5 THEN LIST;
RUN;
```

%COPY ステートメント

SAS マクロライブラリ内にある指定の項目をコピーします。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%MACRO ステートメント” (309 ページ)および“SASMSTORE=システムオプション” (363 ページ)

構文

%COPY *macro-name* /<*option1* <*...option-n*>> SOURCE

必須引数

macro-name
%COPY ステートメントで使用するマクロの名前を指定します。

SOURCESRC
出力先にコピーするマクロのソースコードを指定します。OUTFILE=オプションを省略した場合、ここで指定したソースコードが SAS ログに出力されます。

***option1* <*...option-n*>**
次のオプションのうち 1 つまたは複数を指定します。

LIBRARY= *libref*LIB=
コンパイル済みマクロのカタログを含んでいる SAS ライブラリのライブラリ参照名を指定します。ライブラリを省略すると、SASMSTORE=オプションで指定したライブラリ参照名が使用されます。

制限事項: このライブラリ参照名として、WORK は指定できません。

OUTFILE=*fileref* | *'external file'*OUT=
%COPY ステートメントの出力先を指定します。この値には、ファイル参照名か外部ファイルを指定できます。

例: %COPY ステートメントの使用

次の例では、%COPY ステートメントは保存されたソースコードを SAS ログに出力しています。

```
/* commentary */ %macro foobar(arg) /store source
des="This macro does not do much";
%put arg = &arg;
* this is commentary!!!;
%* this is macro commentary;
%mend /* commentary; */; /* Further commentary */
NOTE: The macro FOOBAR completed compilation without errors.
%copy foobar/source;
```

このプログラムを実行すると、次の結果が SAS ログに出力されます。

```
%macro foobar(arg) /store source
des="This macro does not do much";
%put arg = &arg;
* this is commentary!!!;
%* this is macro commentary;
%mend /* commentary; */;
```

%DISPLAY ステートメント

マクロウィンドウを表示します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードでのみ使用可能

参照項目: “%WINDOW ステートメント” (327 ページ)

構文

%DISPLAY *window*<*.group*> <NOINPUT> <BLANK>
<BELL> <DELETE> ;

必須引数

***window* <*.group*>**
表示するウィンドウとフィールドグループを指定します。ウィンドウが複数のフィールドグループを含んでいる場合、完全な *window.group* 指定を行う必要があります。ウィンドウが単一の名付けられていないグループを含んでいる場合、*window* のみを指定します。

NOINPUT
ウィンドウに表示されるフィールドには値を入力できないことを指定します。NOINPUT オプションを省略すると、ウィンドウに表示される保護されていないフィールドに値を入力できるようになります。%DISPLAY ステートメントがマクロ定義内にあり、かつ複数のフィールドグループを単一の表示にマージしたい場合、NOINPUT オプションを使用します。特定の%DISPLAY ステートメントで NOINPUT を指定すると、後で複数のグループを表示する場合に、特定のグループが表示されたままになります。

BLANK
ウィンドウ内の表示をクリアします。BLANK オプションを使用すると、以前の表示に含まれていたフィールドが現在の表示に現れないようになります。このオプションは、%DISPLAY ステートメントがマクロ定義内に含まれており、しかもそれが *window.group* 指定の一部である場合にのみ有益です。%DISPLAY ステートメントがマクロ定義外にある場合、%DISPLAY ステートメントを実行するたびにウィンドウ内の表示は自動的にクリアされます。

BELL
ウィンドウを表示する際に、パーソナルコンピュータのベルを鳴らします(使用可能な場合)。

DELETE
このオプションを記述した%DISPLAY ステートメントからの処理が通過した時点で、ウィンドウの表示を削除します。DELETE オプションは、%DISPLAY ステートメントがマクロ定義内にある場合に有益です。

詳細

%DISPLAY ステートメントの各実行では、それぞれ 1 つのフィールドグループだけを表示できます。保護されていないフィールドを含むウィンドウを表示する場合、必要なフィールドに値を入力して ENTER キーを押すと、対応する表示がウィンドウから削除されます。

ウィンドウが保護フィールドのみを含んでいる場合、ENTER キーを押すと、対応する表示がウィンドウから削除されます。あるウィンドウが表示されている間、コマンドやファンクションキーを使用することにより、他のウィンドウの表示や、現在のウィンドウサイズの変更などが行えます。

%DO ステートメント

%DO グループを開始します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%END ステートメント” (299 ページ)

構文

%DO;
text and macro language statements

%END;

詳細

%DO ステートメントは、マクロ定義の特定セクションの開始を指定します。このセクションは、対応する%END ステートメントを検出するまで 1 つの単位として扱われます。このマクロセクションのことを%DO グループと呼びます。%DO グループはネストが可能です。

単純な%DO ステートメントは、%IF 条件の true または false に応じて処理されるマクロセクションを指定するために、しばしば%IF-%THEN/%ELSE ステートメントと組み合わせて使用されます。

例: 2 つのレポートのうちどちらか 1 つを生成する

次に示すマクロでは、2 つの%DO グループを%IF-%THEN/%ELSE ステートメントと組み合わせて使用することで、条件に応じて 2 つのレポートのうちどちらか 1 つを出力します。

```
%macro reportit(request);
%if %upcase(&request)=STAT %then
%do;
proc means;
title "Summary of All Numeric Variables";
run;
%end;
%else %if %upcase(&request)=PRINTIT %then
%do;
proc print;
title "Listing of Data";
run;
%end;
%else %put Incorrect report type. Please try again.;
title;
%mend reportit;
%reportit(stat)
%reportit(printit)
```

マクロ変数 REQUEST の値として **stat** を指定すると、PROC MEANS ステップが生成されます。**printit** を指定すると、PROC PRINT ステップが生成されます。それ以外の値を指定すると、カスタマイズしたエラーメッセージが SAS ログに出力されます。

%DO、反復ステートメント

インデックス変数の値に基づいて、マクロの特定セクションを繰り返し実行します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%END ステートメント” (299 ページ)

構文

%DO *macro-variable=start* %TO *stop* <%BY *increment*> ;
text and macro language statements

%END;

必須引数

macro-variable
マクロ変数名を指定するか、またはマクロ変数名を生成するテキスト式を指定します。このマクロ変数の値は、%DO ループの反復回数を決定するインデックスとして機能します。インデックスとして指定されたマクロ変数が存在しない場合、マクロプロセッサは同変数をローカルシンボルテーブル内に作成します。

ユーザーはインデックス変数の値を処理中に変更できます。たとえば、ある条件が満たされた場合にインデックス変数の値を *stop* 値よりも大きい値に設定することで、ループの処理を終了できます。

startstop
反復%DO ステートメントと%END ステートメント間にあるマクロの部分を処理する回数を制御する整数を指定するか、またはそのような整数を生成するマクロ式を指定します。

%DO グループの初回反復時に、*macro-variable* の値は *start* に等しくなります。処理を続行すると、*macro-variable* の値は *increment* の値だけ変化し、*macro-variable* の値が *start*～*stop* の値の範囲外になるまで反復処理が続けられます。

increment
ループを繰り返すたびにインデックス変数に加算される整数(ゼロ以外)を指定するか、またはそのような整数を生成するマクロ式を指定します。デフォルトでは、*increment* は 1 になります。*increment* は、ループの初回反復前に評価されます。このため、この値はループの反復時には変更できません。

例: 一連の DATA ステップの作成

次の例は、マクロ定義における反復%DO グループの使い方を示すものです。

```
%macro create(howmany);
%do i=1 %to &howmany;
data month&i;
infile in&i;
input product cost date;
run;
%end;
%mend create;
%create(3)
```

前述のマクロ CREATE を実行すると、次のステートメントが生成されます。

```
DATA MONTH1;
INFILE IN1;
INPUT PRODUCT COST DATE;
RUN;
DATA MONTH2;
```

```
INFILE IN2;
INPUT PRODUCT COST DATE;
RUN;
DATA MONTH3;
INFILE IN3;
INPUT PRODUCT COST DATE;
RUN;
```

%DO %UNTIL ステートメント

条件が true になるまでマクロのセクションを繰り返し実行します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%END ステートメント” (299 ページ)

構文

%DO %UNTIL (*expression*);
text and macro language statements

%END;

必須引数

expression
論理値に置換される任意のマクロ式を指定します。マクロプロセッサは、各反復の末尾でこの式を評価します。この式の値がゼロ以外の整数である場合、この式は true になります。この式の値がゼロである場合、この式は false になります。この式がヌル値に置換されるか、または非数値文字を含む値に置換される場合、マクロプロセッサはエラーメッセージを発行します。

%DO %UNTIL ステートメントに指定する式の例を次に示します。

- %do %until(&hold=no);
- %do %until(%index(&source,&excerpt)=0);

詳細

%DO %UNTIL ステートメントは、各反復の末尾で、条件の値をチェックします。このため、%DO %UNTIL ループは最低 1 回は必ず反復されます。

例: パラメータの検証

次の例では、%DO %UNTIL ステートメントを使用してオプションのリストをスキャンし、パラメータ TYPE の有効性を検証しています。

```
%macro grph(type);
%let type=%upcase(&type);
%let options=BLOCK HBAR VBAR;
%let i=0;
%do %until (&type=%scan(&options,&i) or (&i>3)) ;
%let i = %eval(&i+1);
```

```
%end;
%if &i>3 %then %do;
%put ERROR: &type type not supported;
%end;
%else %do;
proc chart;&type sex / group=dept;
run;
%end;
%mend grph;
```

値 HBAR を引数とするマクロ GRPH を呼び出すと、次のステートメントが生成されます。

```
PROC CHART;
HBAR SEX / GROUP=DEPT;
RUN;
```

値 PIE を引数とするマクロ GRPH を呼び出すと、次の行が SAS ログに出力されます。

```
ERROR: PIE type not supported
```

%DO %WHILE ステートメント

条件が true の間はマクロのセクションを繰り返し実行します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%END ステートメント” (299 ページ)

構文

%DO %WHILE (*expression*);
text and macro program statements

%END;

必須引数

expression
論理値に置換される任意のマクロ式を指定します。マクロプロセッサは、各反復の先頭でこの式を評価します。この式の値がゼロ以外の整数である場合、この式は true になります。この式の値がゼロである場合、この式は false になります。この式がヌル値に置換されるか、または非数値文字を含む値に置換される場合、マクロプロセッサはエラーメッセージを発行します。

%DO %WHILE ステートメントに指定する式の例を次に示します。

- %do %while(&a<&b);
- %do %while(%length(&name)>20);

詳細

%DO %WHILE ステートメントは、ループの先頭で条件をテストします。マクロプロセッサが条件を初めてテストした場合にその条件が false であった場合、%DO %WHILE は反復されません。

例: タイトルからマークアップタグを削除する

次の例では、%DO %WHILE ステートメントを使用して、テキストからマークアップ(SGML)タグを削除し、TITLE ステートメントを生成します。

```
%macro untag(title);
%let stbk=%str(<);
%let etbk=%str(>);
/* Do loop while tags exist */
%do %while (%index(&title,&stbk)>0) ;
%let pretag=;
%let posttag=;
%let pos_et=%index(&title,&etbk);
%let len_ti=%length(&title);
/* Is < first character? */
%if (%qsubstr(&title,1,1)=&stbk) %then %do;
%if (&pos_et ne &len_ti) %then
%let posttag=%qsubstr(&title,&pos_et+1);
%end;
%else %do;
%let pretag=%qsubstr(&title,1,(%index(&title,&stbk)-1));
/* More characters beyond end of tag (>) ? */
%if (&pos_et ne &len_ti) %then
%let posttag=%qsubstr(&title,&pos_et+1);
%end;
/* Build title with text before and after tag */
%let title=&pretag&posttag;
%end;
title "&title";
%mend untag;
```

マクロ UNTAG を次のように呼び出したとします。

```
%untag(<title>Total <emph>Overdue </emph>Accounts</title>)
```

この場合、同マクロは次のような TITLE ステートメントを生成します。

```
TITLE "Total Overdue Accounts";
```

タイトルテキストにカンマなどの特殊文字が含まれている場合、引数に対して%NRSTR 関数を適用した上でマクロ UNTAG を呼び出します。

```
%untag(
%nrstr(<title>Accounts: Baltimore, Chicago, and Los Angeles</title>))
```

%END ステートメント

%DO グループを終了します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

構文

%END;

例: %DO グループを終了する

次のマクロは、%END ステートメントで終わる%DO %WHILE ループを含んでいます。

```
%macro test(finish);
%let i=1;
%do %while (&i<&finish);
%put the value of i is &i;
%let i=%eval(&i+1);
%end;
%mend test;
%test(5)
```

finish の値に 5 を指定してマクロ TEST を呼び出すと、次の行が SAS ログに出力されます。

```
The value of i is 1
The value of i is 2
The value of i is 3
The value of i is 4
```

%GLOBAL ステートメント

実行中の SAS セッションの全体で利用できるマクロ変数を作成します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: "%LOCAL ステートメント" (308 ページ)

構文

%GLOBAL *macro-variable-1 <...macro-variable-n>*;

必須引数

macro-variable-1 <...macro-variable-n>
1 つ以上のマクロ変数名を指定するか、または 1 つ以上のマクロ変数名を生成するテキスト式を指定します。%GLOBAL ステートメントでは、SAS 変数のリストや、SAS 変数のリストを生成するマクロ式は使用できません。

詳細

%GLOBAL ステートメントは、1 つ以上のマクロ変数を作成し、同変数にヌル値を割り当てます。グローバルマクロ変数とは、実行中の SAS セッションまたは SAS ジョブの全体で利用できる変数です。

%GLOBAL ステートメントにより作成されるマクロ変数は、ユーザーが別の値を割り当てるまで、ヌル値を保持します。すでに存在するマクロ変数を%GLOBAL ステートメントで指定した場合、既存の値は変更されません。

比較

- %GLOBAL ステートメントおよび%LOCAL ステートメントは、どちらも固有のスコープを持つマクロ変数を作成します。ただし、%GLOBAL ステートメントは、SAS セッションや SAS ジョブの実行全体を通じて存在するグローバルマクロ変数を作成します。一方、%LOCAL ステートメントは、その変数を定義しているマクロの実行時のみ存在するローカルマクロ変数を作成します。
- グローバルマクロ変数とローカルマクロ変数を同じ名前で定義した場合、マクロプロセッサは、そのローカル変数を含んでいるマクロの実行時には、同ローカル変数の値を使用します。そのローカル変数を含んでいるマクロが実行されていない場合、マクロプロセッサは同グローバル変数の値を使用します。

例: マクロ定義にグローバル変数を作成する

```
%macro vars(first=1,last=);
%global gfirst glast;
%let gfirst=&first;
%let glast=&last;
var test&first-test&last;
%mend vars;
```

次のプログラムをサブミットすると、マクロ VARS は、VAR ステートメントと、TITLE ステートメントで使用されるマクロ変数の値を生成します。

```
proc print;
%vars(last=50)
title "Analysis of Tests &gfirst-&glast";
run;
```

このマクロを実行すると、次の SAS ステートメントが生成されます。

```
PROC PRINT;
VAR TEST1-TEST50;
TITLE "Analysis of Tests 1-50";
RUN;
```

%GOTO ステートメント

指定したラベルにマクロ処理を分岐させます。

種類: マクロステートメント

別名: %GO TO

制限事項: マクロ定義でのみ使用可能

参照項目: “%label ステートメント” (306 ページ)

構文

%GOTO *label*;

必須引数

label

実行の分岐先にしたいラベル名を指定するか、またはそのようなラベルを生成するテキスト式を指定します。%GOTO ステートメント内でラベルを生成するテキスト式のことを、*計算される%GOTO の分岐先*と呼びます。[1]

次の例は、*label* の使い方を示すものです。

- `%goto findit; /* branch to the label FINDIT */`
- `%goto &home; /* branch to the label that is */`
  `/* the value of the macro variable HOME */`

注意:

%GOTO ステートメント内のラベル名の前にはパーセント記号(%)を付けません。
%GOTO ステートメントの構文では、ラベル名の前に%を含めません。%を使用した場合、マクロプロセッサはラベルを生成するために、その名前のマクロを呼び出そうとします。

詳細

%GOTO ステートメントによる分岐には 2 つの制限事項があります。1 つ目の制限は、%GOTO ステートメントのターゲットとなるラベルが現在のマクロ内に存在していなければならないことです。%GOTO ステートメントでは、別のマクロ内のラベルへは分岐できません。2 つ目の制限は、%GOTO ステートメントは、現在実行されていない反復%DO ループ、%DO %UNTIL ループ、%DO %WHILE ループ内のポイントへの分岐は実行できないことです。

例: 大きいマクロに Exit を指定する

%GOTO ステートメントは、大きなマクロで、エラーが発生した場合の出口(Exit)を提供したい場合に便利です。

```
%macro check(parm);
%local status;
%if &parm= %then %do;
%put ERROR: You must supply a parameter to macro CHECK.;
%goto exit;
%end;
more macro statements that test for error conditions
%if &status > 0 %then %do;
%put ERROR: File is empty.;
%goto exit;
%end;
more macro statements that generate text
%put Check completed successfully.;
%exit: %mend check;
```

%IF-%THEN/%ELSE ステートメント

マクロの一部を条件付きで処理します。

種類: マクロステートメント

[1] 計算される%GOTO の分岐先は%や&を含んでおり、ラベルに置き換えられます。

制限事項: マクロ定義でのみ使用可能

構文

%IF *expression* **%THEN** *action*;

< %ELSE *action*;>

必須引数

expression
整数に置き換えられる任意のマクロ式を指定します。この式がゼロ以外の整数に置換される場合、同式は true となり、%THEN 句が処理されます。この式がゼロに置換される場合、同式は false となり、%ELSE 句が(存在するならば)処理されます。この式がヌル値に置換されるか、または非数値文字を含む値に置換される場合、マクロプロセッサはエラーメッセージを発行します。マクロ式の作成とその評価方法についての詳細は、"マクロ式" (73 ページ)を参照してください。

%IF-%THEN ステートメントでの式の使用例を次に示します。

- `%if &name=GEORGE %then %let lastname=smith;`
- `%if %upcase(&name)=GEORGE %then %let lastname=smith;`
- `%if &i=10 and &j>5 %then %put check the index variables;`

action
定数テキスト、テキスト式、マクロステートメントのいずれかを指定します。*action* にセミコロンが含まれている場合(たとえば SAS ステートメントを指定する場合など)、%THEN の直後の最初のセミコロンで%THEN 句が終了します。*action* 内にあるセミコロンが原因で%IF-%THEN ステートメントが終了するのを防ぐには、%DO グループを使用するか、または%STR などのクォーティング関数を使用します。次の例では、セミコロンを含むテキストを条件に応じて生成する 2 つの方法を示しています。

- ```
  %if &city ne %then %do;
  keep citypop statepop;
  %end;
  %else %do;
  keep statepop;
  %end;
  ```
- ```
  %if &city ne %then %str(keep citypop statepop;);
  %else %str(keep statepop;);
  ```

詳細

マクロ言語には、サブセット化%IF ステートメントは含まれていません。このため、%THEN を指定せずに%IF ステートメントを使うことはできません。

%IF-%THEN ステートメントで文字値を比較する式では、ホストオペレーティングシステムの並べ替え順を使用して比較が行われます。お使いの動作環境における並べ替え順についての詳細は、*Base SAS プロシジャガイド*の The SORT PROCEDURE を参照してください。

比較

%IF-%THEN/%ELSE ステートメントと IF-THEN/ELSE ステートメントは似ていますが、両者は別の言語に属しています。SAS マクロ言語の一部である%IF-%THEN/

%ELSE ステートメントは、通常、条件に応じてテキストを生成します。一方、SAS 言語の一部である IF-THEN/ELSE ステートメントは、DATA ステップの実行時に、条件に応じて SAS ステートメントを実行します。

%IF-%THEN/%ELSE ステートメントの条件として指定される式には、定数テキストか、または定数テキストを生成するテキスト式であるオペランドのみを含めることができます。一方、IF-THEN/ELSE ステートメントの条件として指定される式には、DATA ステップ変数、文字定数、数値定数、日付および時刻定数のみを含めることができます。

%IF-%THEN/%ELSE ステートメントが DATA ステップの一部となるテキストを生成する場合、そのテキストは DATA ステップコンパイラによりコンパイルされ実行されます。一方、IF-THEN/ELSE ステートメントが DATA ステップ内で実行される時点で、マクロ機能により生成されたテキストはすべてその置換、トークン化、コンパイルが済んでいます。コンパイル済みのコード内にはマクロ言語要素は存在しません。両ステートメントの違いについては、次に示す例 1: %IF-%THEN/%ELSE ステートメントと IF-THEN/ELSE ステートメントを組み合わせて使用するを参照してください。

詳細は、“SAS プログラムとマクロ処理” (13 ページ)および“マクロ式” (73 ページ)を参照してください。

例

例 1: %IF-%THEN/%ELSE ステートメントと IF-THEN/ELSE ステートメントを組み合わせて使用する

次に示すマクロ SETTAX では、%IF-%THEN/%ELSE ステートメントを使用してマクロ変数 TAXRATE の値を検査することで、2 つの DATA ステップのうちどちらを生成するかを制御しています。最初の DATA ステップには、DATA ステップ変数 SALE を使用して DATA ステップ変数 TAX の値を設定する IF-THEN/ELSE ステートメントが含まれています。

```
%macro settax(taxrate);
%let taxrate = %upcase(&taxrate);
%if &taxrate = CHANGE %then
%do;
data thisyear;
set lastyear;
if sale > 100 then tax = .05;
else tax = .08;
run;
%end;
%else %if &taxrate = SAME %then
%do;
data thisyear;
set lastyear;
tax = .03;
run;
%end;
%mend settax;
```

マクロ変数 TAXRATE の値が **CHANGE** である場合、このマクロは次の DATA ステップを生成します。

```
DATA THISYEAR;
SET LASTYEAR;
IF SALE > 100 THEN TAX = .05;
ELSE TAX = .08;
RUN;
```

マクロ変数 TAXRATE の値が **SAME** である場合、このマクロは次の DATA ステップを生成します。

```
DATA THISYEAR;
SET LASTYEAR;
TAX = .03;
RUN;
```

例 2: レポートの条件付き表示

次の例では、%IF-%THEN/%ELSE ステートメントを使用して、2 つのレポートのうちどちらかを生成するステートメントを生成しています。

```
%macro fiscal(report);
%if %upcase(&report)=QUARTER %then
%do;
title 'Quarterly Revenue Report';
proc means data=total;
var revenue;
run;
%end;
%else
%do;
title 'To-Date Revenue Report';
proc means data=current;
var revenue;
run;
%end;
%mend fiscal;
%fiscal(quarter)
```

マクロ FISCAL を呼び出すと、次のステートメントが生成されます。

```
TITLE 'Quarterly Revenue Report';
PROC MEANS DATA=TOTAL;
VAR REVENUE;
RUN;
```

%INPUT ステートメント

マクロの実行時に、マクロ変数に値を割り当てます。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%PUT ステートメント” (316 ページ)“%WINDOW ステートメント” (327 ページ)および “SYSBUFFR 自動マクロ変数” (196 ページ)

構文

%INPUT<*macro-variable-1* <*...macro-variable-n*>> ;

必須引数

引数なし
入力されたすべてのテキストを、自動マクロ変数 SYSBUFFR に割り当てます。

***macro-variable-1* <*...macro-variable-n*>**
マクロ変数名か、またはマクロ変数名を生成するマクロテキスト式を指定します。%INPUT ステートメントでは任意の数の変数名を指定できます。複数の変数名は空白で区切って指定します。

詳細

マクロプロセッサは、%INPUT ステートメントの直後にサブミットされた行を、その%INPUT ステートメントに対する応答として解釈します。この行は、対話型ラインモードセッションの一部とするか、またはウィンドウ環境セッション時に **Program Editor** ウィンドウ内でサブミットできます。

%INPUT ステートメントを対話型ラインモードセッションの一部として実行する場合、マクロプロセッサは値を含む行をユーザーが入力するまで待機します。ウィンドウ環境セッションでは、マクロプロセッサはユーザーが値を入力するまで待機しません。その代わりに、マクロプロセッサは、プログラム内で処理される次の行を読み取り、変数値を割り当てようとします。同様に、ウィンドウ環境において、長いプログラムの一部としてオープンコード内に%INPUT ステートメントを含んでいるマクロを呼び出した場合、マクロプロセッサは、そのマクロ呼び出しに続くプログラム内に含まれている次の行を読み取ります。ウィンドウ環境において、オープンコード内で%INPUT ステートメントをサブミットする場合、%INPUT ステートメント(または%INPUT ステートメントを含んでいるマクロ呼び出し)に続く行に、割り当てたい値が含まれていることを確認してください。

%INPUT ステートメントで変数を指定した場合、マクロプロセッサは、各変数の位置に基づいて、変数とユーザーの応答に含まれている値とを対応付けます。すなわち、ユーザーが入力した最初の値は%INPUT ステートメント内に指定されている最初の変数に割り当てられ、2 番目の値は同ステートメント内の 2 番目の変数に割り当てられる、という具合になります。

各変数に割り当てられる値は、単一のワードであるか、または引用符で囲まれた文字列でなければなりません。複数の値は空白で区切って指定します。すべての値がマクロ変数名に対応付けられた後、超過分のテキストは自動マクロ変数 SYSBUFFR の値になります。

例: 応答をマクロ変数に割り当てる

対話型ラインモードセッションで次のステートメントを実行すると、プロンプトが表示され、ユーザーによる応答がマクロ変数 FIRST に割り当てられます。

```
%put Enter your first name:;
%input first;
```

%label ステートメント

%GOTO ステートメントの分岐先を指定します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%GOTO ステートメント” (301 ページ)

構文

%label: *macro-text*

必須引数

label
SAS 名を指定します。

macro-text
マクロステートメント、テキスト式、定数式のいずれかを指定します。それぞれの例を次に示します。

- `%one: %let book=elementary;`
- `%out: %mend;`
- `%final: data _null_;`

詳細

- ラベル名の前には%を付けます。このラベルを%GOTO ステートメント内で指定する場合には、ラベルの前に%は付けません。
- %GOTO ステートメントとステートメントラベルを使用する代わりに、%IF-%THEN ステートメントと%DO グループを使用することもできます。

例: プログラムのフローの制御

次に示すマクロ INFO を、パラメータ TYPE に値 **short** を指定して呼び出した場合、%GOTO ステートメントによりラベル QUICK へのジャンプが実行されます。

```
%macro info(type);
%if %upcase(&type)=SHORT %then %goto quick; /* No % here */
proc contents;
run;
proc freq;
tables _numeric_;
run;
%quick: proc print data=_last_(obs=10); /* Use % here */
run;
%mend info;
%info(short)
```

パラメータ TYPE に **short** を指定してマクロ INFO を呼び出すと、次のステートメントが生成されます。

```
PROC PRINT DATA=_LAST_(OBS=10);
RUN;
```

%LET ステートメント

マクロ変数を作成し、その変数に値を割り当てます。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%STR 関数と%NRSTR 関数” (258 ページ)

構文

%LET *macro-variable* =<*value*> ;

必須引数

macro-variable
マクロ変数名か、またはマクロ変数名を生成するテキスト式を指定します。この名前は、新規または既存のマクロ変数を参照します。

value
文字列またはテキスト式を指定します。*value* を省略するとヌル値(長さがゼロの文字)が生成されます。*value* の先頭および末尾にある空白は無視されます。これらの空白に意味がある場合、*value* を%STR 関数で囲みます。

詳細

%LET ステートメントに指定したマクロ変数がすでに存在していた場合、その%LET ステートメントにより値が変更されます。%LET ステートメントが一度に定義できるマクロ変数は 1 つだけです。

例: %LET ステートメントの例

%LET ステートメントの例を次に示します。

```
%macro title(text,number);
title&number "&text";
%mend;
%let topic= The History of Genetics ; /* Leading and trailing */
/* blanks are removed */
%title(&topic,1)
%let subject=topic; /* &subject resolves */
%let &subject=Genetics Today; /* before assignment */
%title(&topic,2)
%let subject=The Future of Genetics; /* &subject resolves */
%let topic= &subject; /* before assignment */
%title(&topic,3)
```

これらのステートメントをサブミットすると、マクロ TITLE により次のステートメントが生成されます。

```
TITLE1 "The History of Genetics";
TITLE2 "Genetics Today";
TITLE3 "The Future of Genetics";
```

%LOCAL ステートメント

その変数自身が定義されているマクロの実行時にのみ使用可能となるマクロ変数を作成します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

参照項目: “%GLOBAL ステートメント” (300 ページ)

構文

%LOCAL *macro-variable-1 <...macro-variable-n>*;

必須引数

***macro-variable-1* <*..macro-variable-n*>**
1 つ以上のマクロ変数名を指定するか、または 1 つ以上のマクロ変数名を生成するテキスト式を指定します。%LOCAL ステートメントでは、SAS 変数のリストや、SAS 変数のリストを生成するマクロ式は使用できません。

詳細

%LOCAL ステートメントは、1 つ以上のローカルマクロ変数を作成します。%LOCAL ステートメントにより作成されるマクロ変数は、ユーザーが別の値を割り当てるまで、ヌル値を保持します。ローカルマクロ変数とは、その変数自身が定義されているマクロの実行時にのみ使用可能となる変数のことです。

%LOCAL ステートメントを使用すると、先にプログラム内で作成したマクロ変数の値が、現在のマクロ内の同じ名前の変数に割り当てた値によってうっかり変更されてしまうことを防止できます。すでに存在するマクロ変数を%LOCAL ステートメントで指定した場合、既存の値は変更されません。

比較

- %LOCAL ステートメントおよび%GLOBAL ステートメントは、どちらも固有のスコープを持つマクロ変数を作成します。ただし、%LOCAL ステートメントは、その変数を含んでいるマクロの実行時のみ存在するローカルマクロ変数を作成します。一方、%GLOBAL ステートメントは、SAS セッションや SAS ジョブの実行全体を通じて存在するグローバルマクロ変数を作成します。
- ローカルマクロ変数とグローバルマクロ変数を同じ名前で定義した場合、マクロ機能は、そのローカル変数を含んでいるマクロの実行時には、同ローカル変数の値を使用します。そのローカル変数を含んでいるマクロが実行されていない場合、マクロ機能は同グローバル変数の値を使用します。

例: グローバル変数と同一名のローカル変数の使用

```
%let variable=1;
%macro routine;
%put ***** Beginning ROUTINE *****;
%local variable;
%let variable=2;
%put The value of variable inside ROUTINE is &variable;
%put ***** Ending ROUTINE *****;
%mend routine;
%routine
%put The value of variable outside ROUTINE is &variable;
```

これらのステートメントをサブミットすると、次の行が SAS ログに表示されます。

```
***** Beginning ROUTINE *****
The value of variable inside ROUTINE is 2
***** Ending ROUTINE *****
The value of variable outside ROUTINE is 1
```

%MACRO ステートメント

マクロ定義を開始します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%MEND ステートメント” (316 ページ)および“SYSPBUFF 自動マクロ変数” (214 ページ)

構文

%MACRO *macro-name* <(*parameter-list*)> </ *option-1* <...*option-n*>> ;

必須引数

macro-name
マクロ名を指定します。マクロ名は SAS 名でなければなりません。%MACRO ステートメント内では、マクロ名を生成するテキスト式は使用できません。また、マクロ機能の予約語はマクロ名として使用できません(マクロ機能の予約語の一覧については、“ マクロ機能の予約語” (369 ページ)を参照してください)。

parameter-list
1 つまたは複数のローカルマクロ変数を指定します。これらの変数の値は、当該マクロの呼び出し時にユーザーにより指定されます。ここで指定したパラメータは、それを定義したマクロにとってローカルとなります。個々のパラメータを指定する必要があります。パラメータを生成するテキスト式は使用できません。パラメータリストには任意の数のマクロパラメータを含めることができます。複数のパラメータはカンマで区切って指定します。このパラメータリスト内にある変数は、通常、当該マクロ内で参照されます。

- *parameter-list* は次の形式のいずれかで指定できます。
  - *<positional parameter-1><. . . ,positional parameter-n>*
  - *<keyword-parameter=<value> <. . . ,keyword-parameter-n=<value>>>*

positional-parameter-1 <. . . ,positional-parameter-n>	は、1 つ以上の位置パラメータを指定します。位置パラメータは任意の順番で指定できますが、マクロの呼び出し時には、各パラメータ値の指定順と、%MACRO ステートメント内での各パラメータの指定順が一致する必要があります。複数の位置パラメータを指定する場合、パラメータ間をカンマで区切ります。呼び出し時に位置パラメータに値を指定しなかった場合、マクロ機能はそのパラメータにヌル値を割り当てます。
keyword-parameter=<value> <. . . ,keyword-parameter-n=<value>>	には、1 つ以上のキーワードパラメータを指定します。キーワードパラメータとは、末尾に等号が付いたマクロパラメータのことです。等号に続いてデフォルト値を指定できます。等号の後のデフォルト値を省略すると、そのキーワードパラメータにはヌル値が割り当てられます。デフォルトを使用すると、より柔軟なマクロ定義の作成が可能となるほか、マクロの呼び出し時に指定する必要のあるパラメータの数を削減できます。デフォルト値をオーバーライドするには、マクロの呼び出し時に、マクロ変数名に続く等号の後に新しい値を指定します。

注: 定義できるパラメータ数に制限はありません。位置パラメータおよびキーワードパラメータの両者を 1 つのマクロ定義に記述する場合、位置パラメータを先に指定する必要があります。

***option-1* <*...option-n*>**
次に示すオプション引数のうち、いずれか 1 つまたは複数を指定できます。

CMD
マクロがネームスタイル呼び出しとコマンドスタイル呼び出しの両方を受け付けることを指定します。CMD オプション付きで定義されたマクロのことを、*コマンドスタイル マクロ*と呼ぶ場合があります。

CMD オプションは、SAS ウィンドウのコマンドラインから実行する予定のマクロに対してのみ指定します。コマンドスタイル呼び出しを使用するには、SAS システムオプション CMDMAC を有効にする必要があります。CMDMAC システムオプションが有効であり、かつコマンドスタイルマクロを自分のプログラム内で定義している場合、マクロプロセッサは、各 SAS コマンドの先頭ワードをスキャンすることで、それがコマンドスタイルのマクロ呼び出しであるかどうかを判定します。SAS システムオプション NOCMDMAC が有効である場合、マクロプロセッサは、%記号に続くワードのみを潜在的なマクロ呼び出しとして扱います。CMDMAC オプションが有効でない場合でも、CMD オプション付きで定義されたマクロに関しては、ネームスタイル呼び出しを使用できます。

DES='*text*'
マクロカタログ内のマクロエントリに関する説明を指定します。この説明テキストの最大長は 256 文字です。説明は引用符で囲む必要があります。この説明は、ユーザーがコンパイル済みマクロを含むカタログの内容を表示した場合に、**CATALOG** ウィンドウ内に表示されます。DES=オプションは、コンパイル済みマクロ機能を使用する場合に特に役立ちます。

MINDELIMITER='*single character*';
MINDELIMITER=グローバルオプションの値をオーバーライドする値を指定します。この値は、一重引用符で囲まれた単一文字でなければならず、1 つの%MACRO ステートメント内に一度だけ記述できます。

MINOPERATOR | NOMINOPERATOR
マクロの実行時に演算式または論理式を評価する際に、マクロプロセッサがニーモニック **IN** および特殊文字#を論理演算子として認識するよう指定します。この引数の設定は、NOMINOPERATOR グローバルシステムオプションの設定をオーバーライドします。

引数 NOMINOPERATOR を指定すると、マクロの実行時に演算式または論理式を評価する際に、マクロプロセッサがニーモニック **IN** および特殊文字#を論理演算子として認識しなくなります。この引数の設定は、MINOPERATOR グローバルシステムオプションの設定をオーバーライドします。

PARMBUFF
マクロ呼び出しにおけるパラメータ値のリスト全体(ネームスタイル呼び出しにおけるかっこを含む)を、自動マクロ変数 SYSPBUFF の値として割り当てます。PARMBUFF オプションを指定すると、可変数個のパラメータ値を受け付けるマクロを定義できます。

マクロ定義内にパラメータのセットと PARMBUFF オプションの両方が含まれている場合、そのマクロを呼び出すと、これらのパラメータは対応する値を受け取ります。また、これにより、値の呼び出しリスト全体が変数 SYSPBUFF に割り当てられます。

PARMBUFF オプション付きで定義されたマクロを、ウィンドウ環境または対話型ラインモードセッションにおいて値リストを指定せずに呼び出すには、その呼び出しに続いて空のかっこを入力するか、またはその他のプログラムステートメントを入力します。この操作により、マクロ定義のパラメータが含まれていない場合であっても、値リストが存在しないことが指定されます。

SECURE | NOSECURE
マクロをコンパイル済みマクロライブラリに保存する際に、そのマクロの内容を暗号化するよう指定します。この機能を使うと、マクロ自身に含まれている知的所有権を保護するような*セキュア*なマクロを作成できます。これらのマクロは、Encryption Algorithm Manager を使用して保護されます。

NOSECURE オプションは、セキュリティを有効化するためにファイルやライブラリのソースをグローバル編集する場合の使用を前提として提供されているものです。たとえば、保護が必要なマクロをいくつか作成するとします。このようなマクロを作成する場合、NOSECURE オプションを使用します。その後、すべてのマクロが完成し実務で使用できるようになった時点で、グローバル編集を行い、NOSECURE を SECURE に変更します。

マクロで SECURE オプションと SOURCE オプションを指定した場合、%COPY ステートメントの使用時に出力が生成されなくなります。この場合、次の注意が SAS ログに出力されます。

```
NOTE: The macro %name was compiled with the SECURE
option. No output will be produced for this %COPY
statement.
```

STMT
マクロがネームスタイル呼び出しまたはステートメントスタイル呼び出しを受け付けることを指定します。STMT オプション付きで定義されたマクロのことを、*ステートメントスタイルマクロ*と呼ぶ場合があります。

ステートメントスタイル呼び出しを使用するには、IMPLMAC システムオプションを有効にする必要があります。IMPLMAC システムオプションが有効であり、かつステートメントスタイルマクロを自分のプログラム内で定義している場合、マクロプロセッサは、各 SAS コマンドの先頭ワードをスキャンすることで、それがステートメントスタイルのマクロ呼び出しであるかどうかを判定します。NOIMPLMAC オプションが有効である場合、マクロプロセッサは、%記号に続くワードのみを潜在的なマクロ呼び出しとして扱います。IMPLMAC オプションが有効でない場合でも、STMT オプション付きで定義されたマクロに関しては、ネームスタイル呼び出しを使用できます。

SOURCE SRC
コンパイル済みマクロのソースとコンパイル済みマクロのコードを SAS カタログ内の 1 つのエントリとして結合し、永久 SAS ライブラリに保存します。SOURCE オプションを使用する場合、STORE オプションと MSTORED オプションを設定する必要があります。永久 SAS ライブラリを指定するには、SASMSTORE=オプションを使用します。マクロの保存やコンパイル済みマクロの呼び出しを行うには、MSTORED オプションが有効でなければなりません(詳細は、“マクロの保存および再利用” (115 ページ)を参照してください)。

注: SOURCE オプションにより保存されたソースコードは、%MACRO キーワードで始まり、セミコロンが末尾に付いた%MEND ステートメントで終わります。

注意:
SOURCE オプションは、ネストされたマクロ定義(別のマクロ内に含まれているマクロ定義)に関しては適用できません。

STORE
コンパイル済みマクロを、SAS カタログ内の 1 つのエントリとして、永久 SAS ライブラリ内に保存します。永久 SAS ライブラリを指定するには、SAS システムオプション SASMSTORE=を使用します。マクロの保存やコンパイル済みマクロの呼び出しを行うには、SAS システムオプション MSTORED が有効でなければなりません(詳細は、“マクロの保存および再利用” (115 ページ)を参照してください)。

詳細

%MACRO ステートメントは、マクロ定義を開始し、マクロ名を割り当てます。同ステートメントには、マクロパラメータかオプションのリストのいずれかまたは両方を含めることができます。

プログラム内で、マクロ定義は、同マクロの呼び出しよりも前に記述する必要があります。%MACRO ステートメントは、SAS プログラム内の任意の場所(データ行を除く)に記述できます。マクロ定義には、CARDS ステートメント、DATALINES ステートメント、PARMCARDS ステートメント、データ行を含めることはできません。その代わりに、INFILE ステートメントを使用します。

デフォルトでは、定義されたマクロは、SAS カタログ内の 1 つのエントリとして、WORK ライブラリに保存されます。定義したマクロを永久 SAS ライブラリに保存すると、そのマクロを後で使用できるようになります。ただし、SAS バージョン 6 以前では、マクロのコピー、リネーム、トランスポートはサポートされません。

マクロ定義はネストできますが、マクロのネストは多くの場合不要でありしかも非効率的です。あるマクロ定義をネストした場合、そのマクロを含んでいるマクロを呼び出すたびに、ネストされているマクロがコンパイルされます。多くの場合、マクロ定義をネストするのではなく、別のマクロ定義内でマクロ呼び出しをネストするだけで十分です。

例

例 1: %MACRO ステートメントを位置パラメータと共に使用する

次の例では、マクロ PRNT により PROC PRINT ステップを生成します。先頭位置にあるパラメータは VAR であり、これは VAR ステートメント内に記述される SAS 変数を表します。2 番目の位置にあるパラメータは SUM であり、これは SUM ステートメント内に記述される SAS 変数を表します。

```
%macro prnt(var,sum);
proc print data=srhigh;
var &var;
sum &sum;
run;
%mend prnt;
```

このマクロを呼び出すと、カンマまでのすべてのテキストが、パラメータ VAR の値となります。カンマの後に続くテキストは、パラメータ SUM の値となります。

```
%prnt(school district enrollmt, enrollmt)
```

実行時に、マクロ PRNT は次のステートメントを生成します。

```
PROC PRINT DATA=SRHIGH;
VAR SCHOOL DISTRICT ENROLLMT;
SUM ENROLLMT;
RUN;
```

例 2: %MACRO ステートメントをキーワードパラメータと共に使用する

マクロ FINANCE では、%MACRO ステートメントで 2 つのキーワードパラメータ YVAR および XVAR を定義しており、PLOT プロシジャを使用してそれらの値をプロットしています。これらのキーワードパラメータの値としては通常 EXPENSES と DIVISION を使用するため、次の%MACRO ステートメントでは、YVAR および XVAR のデフォルト値として EXPENSES と DIVISION をそれぞれ指定しています。

```
%macro finance(yvar=expenses,xvar=division);
proc plot data=yearend;
plot &yvar*&xvar;
```

```
run;
%mend finance;
```

- デフォルト値を使用する場合、このマクロをパラメータなしで呼び出します。

  ```
  %finance()
  ```

  または

  ```
  %finance;
  ```

  マクロプロセッサは次の SAS コードを生成します。

  ```
  PROC PLOT DATA=YEAREND;
  PLOT EXPENSES*DIVISION;
  RUN;
  ```

- 新しい値を割り当てるには、パラメータに続いて等号とその値を指定します。

  ```
  %finance(xvar=year)
  ```

  YVAR の値は変更されないため、デフォルト値のままになります。このマクロを実行すると、次のコードが生成されます。

  ```
  PROC PLOT DATA=YEAREND;
  PLOT EXPENSES*YEAR;
  RUN;
  ```

例 3: %MACRO ステートメントを PARMBUFF オプションと共に使用する

マクロ PRINTZ では、PARMBUFF オプションを使用することにより、同マクロを呼び出すたびに異なる数の引数を入力できるようにしています。

```
%macro printz/parmbuff;
%let num=1;
%let dsname=%scan(&syspbuff,&num);
%do %while(&dsname ne);
proc print data=&dsname;
run;
%let num=%eval(&num+1);
%let dsname=%scan(&syspbuff,&num);
%end;
%mend printz;
```

PRINTZ のマクロ定義にはパラメータが含まれていないにもかかわらず、次の PRINTZ マクロの呼び出しでは、4 つのパラメータ値 **PURPLE**、**RED**、**BLUE**、**TEAL** を指定しています。

```
%printz(purple,red,blue,teal)
```

結果として、SAS システムは次のステートメントを受け取ります。

```
PROC PRINT DATA=PURPLE;
RUN;
PROC PRINT DATA=RED;
RUN;
PROC PRINT DATA=BLUE;
RUN;
PROC PRINT DATA=TEAL;
RUN;
```

例 4: %MACRO ステートメントを SOURCE オプションと共に使用する

SOURCE オプションは、コンパイル済みマクロのソースとコンパイル済みマクロのコードを結合して保存します。ソースを SAS ログに出力するには、%COPY ステートメントを使用します。保存されたソースの表示や取り出しに関する詳細は、“%COPY ステートメント” (292 ページ)を参照してください。

```
/* commentary */ %macro foobar(arg) /store source
des="This macro does not do much";
%put arg = &arg;
* this is commentary!!!;
%* this is macro commentary;
%mend /* commentary; */; /* Further commentary */
NOTE: The macro FOOBAR completed compilation without errors.
%copy foobar/source;
```

このプログラムを実行すると、次の結果が SAS ログに出力されます。

```
%macro foobar(arg) /store source
des="This macro does not do much";
%put arg = &arg;
* this is commentary!!!;
%* this is macro commentary;
%mend /* commentary; */;
```

例 5: %MACRO ステートメントを STORE オプションと SECURE オプションと共に使用する

SECURE オプションは、STORE オプションと共に使用する必要があります。次の例では、STORE オプションと暗黙の NOSECURE オプションを使用することにより、プレーンテキストで保存されたマクロを作成しています。

```
 options mstored sasmstore=mylib;
libname mylib "mylib";
%macro nonsecure/store; /* This macro is stored in plain text */
data _null_;
x=1;
put "This data step was generated from a non-secure macro.";
run;
%mend nonsecure;
%nonsecure
filename maccat catalog 'mylib.sasmacr.nonsecure.macro';
data _null_;
infile maccat;
input;
list;
run;
```

次の例では、STORE オプションと SECURE オプションを指定することにより、暗号化されたマクロを作成しています。

```
options mstored sasmstore=mylib;
libname mylib "mylib";
%macro secure/store secure; /* This macro is encrypted */
data _null_;
x=1;
put "This data step was generated from a secure macro.";
run;
%mend secure;
%secure
```

```
filename maccat catalog 'mylib.sasmacr.secure.macro';
data _null_;
infile maccat;
input;
list;
run;
```

%MEND ステートメント

マクロ定義を終了します。

種類: マクロステートメント

制限事項: マクロ定義でのみ使用可能

構文

%MEND <*macro-name*> ;

必須引数

macro-name
このステートメントによってそのマクロ定義を終了するマクロの名前を指定します。このマクロ名の指定はオプションですが、マクロ定義の終了時にもマクロ名を指定した方が、マクロ定義の範囲がより明確になります。*macro-name* を指定する場合、%MEND ステートメントに指定するマクロ名は%MACRO ステートメントに指定したマクロ名と一致する必要があります。そうでない場合、SAS システムは警告メッセージを表示します。

例: マクロ定義の終了

```
%macro disc(dsn);
data &dsn;
set perm.dataset;
where month="&dsn";
run;
%mend disc;
```

%PUT ステートメント

テキストやマクロ変数の情報を SAS ログに出力します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードでのみ使用可能

構文

%PUT <*text* | _ALL_ | _AUTOMATIC_ | _GLOBAL_ | _LOCAL_ | _USER_> ;

必須引数

引数なし
空白行を SAS ログに出力します。

text
SAS ログに出力するテキストまたはテキスト式を指定します。*text* の長さがが現在の行サイズよりも大きい場合、同テキストの残りの部分は次の行に出力されます。%PUT ステートメントは、*text* の先頭および末尾にある空白を削除します。これらの空白を残したい場合は、引数のテキストに対してマクロクォーティング関数を使用します。

ALL
すべてのユーザー定義のマクロ変数および自動マクロ変数の値をリストします。

AUTOMATIC
自動マクロ変数の値をリストします。リストされる自動変数は、お使いのオペレーティングシステム上にインストールされている SAS 製品により異なります。スコープは AUTOMATIC として識別されます。

GLOBAL
ユーザー定義のグローバルマクロ変数をリストします。スコープは GLOBAL として識別されます。

LOCAL
ユーザー定義のローカルマクロ変数をリストします。スコープは、現在実行中のマクロの名前になります。

USER
ユーザー定義のグローバル/ローカルマクロ変数をリストします。スコープは GLOBAL として識別されるか、またはこのマクロ変数が定義されているマクロの名前になります。

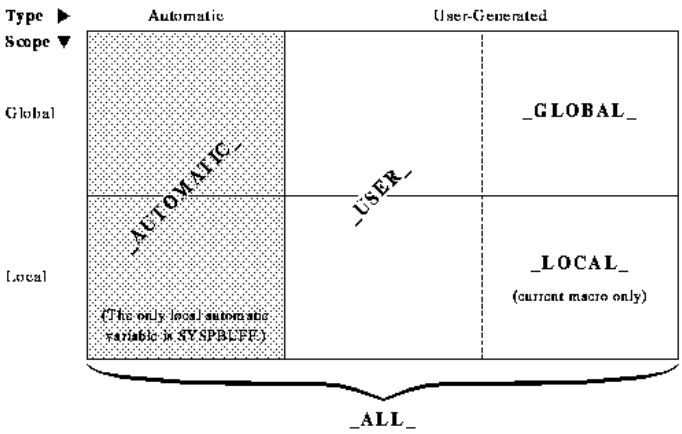
詳細

%PUT ステートメントを使用してマクロ変数の説明をリストする場合、同ステートメントの実行時に存在しているマクロ変数のみが対象となります。この説明には、マクロ変数のスコープ、名前、値が含まれます。ヌル値を含むマクロ変数の場合、その変数のスコープと名前のみが表示されます。マクロクォーティング関数によりクォートされた値の文字列は、クォートされたままになります。現在の行サイズよりも大きい長さを持つ値は、次の行に折り返して表示されます。マクロ変数は、まず現在のローカルマクロ変数、続いて現在のグローバルマクロ変数という順番でリストされます。

注: 特定のスコープ内で、マクロ変数は任意の順番で現れるため、%PUT ステートメントの実行タイミングや実行する SAS セッションが異なれば、マクロ変数の順番も異なる場合があります。このため、変数がリスト内の特定位置に存在することを前提とするようなプログラムは作成しないでください。

%PUT ステートメントの引数の種類とスコープの関係を次の図に示します。

図 19.1 %PUT ステートメントの引数の種類とスコープの関係



%PUT ステートメントは、SAS システムにより生成される ERROR、NOTE、WARNING の各メッセージに似たメッセージを生成する場合、テキストを異なる色で表示します。テキストを異なる色で表示するには、%PUT ステートメントの引数の先頭ワードに ERROR、NOTE、WARNING のいずれかを指定し、その直後に色またはハイフンを指定します。ERROR、NOTE、WARNING の代わりに、それらに相当する自国語のワードを使用することもできます。ハイフンを指定すると、ワード ERROR、NOTE、WARNING が空白となります。

注: %PUT ステートメントで、自動マクロ変数 SYSWARNINGTEXT および SYSERRORTEXT により生成された最終メッセージテキスト(&や%を含んでいるもの)を引数として使用する場合、同テキストに対してマクロクォーティング関数%SUPERQ を適用する必要があります。詳細は、"SYSERRORTEXT 自動マクロ変数" (206 ページ)および"SYSWARNINGTEXT 自動マクロ変数" (223 ページ)を参照してください。

ヒント アンパサンド記号と、直接マクロ変数参照のマクロ変数名の間に等号を配置すると、そのマクロ変数の名前が、同マクロ変数の値と共に SAS ログに表示されます。

```
%let x=1;
%put &=x;
X=1;
```

例

例 1: テキストの表示

%PUT ステートメントを使用してテキストを SAS ログに出力する例を次に示します。

```
%put One line of text.;
%put %str(Use a semicolon(;) to end a SAS statement.);
%put %str(Enter the student%'s address.);
```

これらのステートメントをサブミットすると、次の行が SAS ログに出力されます。

```
One line of text.
Use a semicolon(;) to end a SAS statement.
Enter the student's address.
```

例 2: 自動変数の表示

すべての自動変数を表示するには、次のステートメントをサブミットします。

```
%put _automatic_;
```

結果として、SAS ログには、各自動変数のスコープ、名前、値がリストされます(リストされる変数は、お使いのサイトにインストールされている SAS 製品により異なります)。

```
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 21JUN97
AUTOMATIC SYSDAY Wednesday
AUTOMATIC SYSDEVIC
AUTOMATIC SYSDSN _NULL_
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 0
AUTOMATIC SYSINFO 0
```

例 3: ユーザー定義の変数の表示

次の例は、すべてのスコープにおけるユーザー定義のマクロ変数をリストします。

```
%macro myprint(name);
proc print data=&name;
title "Listing of &name on &sysdate";
footnote "&foot";
run;
%put _user_;
%mend myprint;
%let foot=Preliminary Data;
%myprint(consumer)
```

%PUT ステートメントは、次の行を SAS ログに出力します。

```
MYPRINT NAME consumer
GLOBAL FOOT Preliminary Data
```

SYSDATE は自動マクロ変数であるため、結果には表示されていないことに注意してください。

マクロ MYPRINT の終了後にユーザー定義のマクロ変数を表示するには、同じ%PUT ステートメントをもう一度サブミットします。

```
%put _user_;
```

その結果として、SAS ログには、マクロ変数 NAME がリストされません。マクロ変数 NAME はマクロ MYPRINT のローカルマクロ変数であるため、マクロ MYPRINT の実行が完了した時点で存在しなくなります。

```
GLOBAL FOOT Preliminary Data
```

例 4: ローカル変数の表示

次の例では、マクロ ANALYZE のローカルマクロ変数を表示します。

```
%macro analyze(name,vars);
proc freq data=&name;
tables &vars;
run;
%put FIRST LIST:;
%put _local_;
```

```
%let firstvar=%scan(&vars,1);
proc print data=&name;
where &firstvar ne .;
run;
%put SECOND LIST:;
%put _local_;
%mend analyze;
%analyze(consumer,car house stereo)
```

SAS ログには次の結果が出力されます。最初の%PUT _LOCAL_ステートメントの後に作成されたマクロ変数 FIRSTVAR が、2 番目の%PUT _LOCAL_ステートメントにより表示されています。

```
FIRST LIST:
ANALYZE NAME consumer
ANALYZE VARS car house stereo
SECOND LIST:
ANALYZE NAME consumer
ANALYZE VARS car house stereo
ANALYZE FIRSTVAR car
```

%RETURN ステートメント

現在実行中のマクロを正常終了します。

種類: マクロステートメント

制限事項: マクロ定義でのみ有効

構文

%RETURN;

詳細

%RETURN マクロステートメントは、現在実行中のマクロを正常終了します。

例: %RETURN ステートメントの使用

次の例で、マクロ CHECKIT の引数に 1 を指定して同マクロを呼び出した場合、%RETURN ステートメントにより同マクロの実行が正常終了されるため、同マクロ内の DATA ステップは実行されません。

```
%macro checkit(error);
%if &error = 1 %then %return;
data a;
x=1;
run;
%mend checkit;
%checkit(0)
%checkit(1)
```

%SYMDEL ステートメント

指定された 1 つ以上の変数をマクログローバルシンボルテーブルから削除します。

種類: マクロステートメント

構文

%SYMDEL *macro-variable-1 <...macro-variable-n></option>* ;

必須引数

macro-variable-1 <...macro-variable-n>
1 つ以上のマクロ変数名を指定するか、または 1 つ以上のマクロ変数名を生成するテキスト式を指定します。%SYMDEL ステートメントでは、SAS 変数のリストや、SAS 変数のリストを生成するマクロ式は使用できません。

options

NOWARN
存在しないマクロ変数を削除しようとした場合に、警告メッセージが発行されないようにします。

詳細

%SYMDEL ステートメントは、存在しないマクロ変数を削除しようとした場合に、警告メッセージを発行します。このような警告メッセージが発行されないようにするには、NOWARN オプションを指定します。

%SYSCALL ステートメント

SAS CALL ルーチンを呼び出します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%SYSFUNC 関数と%QSYSFUNC 関数” (273 ページ)

構文

%SYSCALL *call-routine*<(*call-routine-argument-1 <...call-routine-argument-n>*)> ;

必須引数

call-routine
SAS CALL ルーチン、SAS/TOOLKIT ソフトウェアを使用して作成されたユーザー定義の CALL ルーチン、または FCMP プロシジャ(*Base SAS プロシジャガイド*を参照)を使用して作成された CALL ルーチンを指定します。すべての SAS CALL ルーチンは、%SYSCALL ステートメントを使って呼び出すことができます。ただし、LABEL、VNAME、SYMPUT、EXECUTE ルーチンは例外です。

call-routine-argument-1 <...call-routine-argument-n>
1 つ以上のマクロ変数名(先頭のアンパサンドは不要)を指定します。複数指定する場合、マクロ変数名の間をカンマで区切ります。CALL ルーチン引数の一部または全部を生成するテキスト式も指定できます。

詳細

%SYSCALL ステートメントで CALL ルーチンを呼び出すと、各マクロ変数引数の値が取り出され、それらの値は置換されないままの状態で CALL ルーチンに渡されます。CALL ルーチンの完了時に、各引数の値が、それぞれ対応するマクロ変数に書き戻されます。%SYSCALL ステートメントがエラー状態を検出すると、マクロ変数値を更新することなく CALL ルーチンの実行が終了します。続いてエラーメッセージがログに出力された後、マクロ処理が続行されます。

注: %SYSCALL ステートメントの引数は、SAS マクロ言語の規則に従って評価されます。これには、関数名および関数の引数リストの両者が含まれます。特に、引数位置が空である場合、NULL 引数ではなく、長さがゼロの引数が生成されます。

注意:
マクロ変数名の先頭にはアンパサンドを付けないでください。 %SYSCALL マクロステートメントにより呼び出された CALL ルーチンの引数は、実行される前に置換されます。マクロ変数の先頭にアンパサンドを付けた場合、マクロ変数の名前ではなく、そのマクロ変数の値が CALL ルーチンに渡されます。

注意:
マクロ変数は文字データのみを含みます。 関数の引数が数値データか文字データのどちらでもよい場合、%SYSCALL ステートメントは指定されたデータを数値データに変換しようとします。このため、指定されたデータが文字データであった場合には、末尾の空白が削除されます。%SYSCALL ステートメントは、文字データである可能性がある引数は変更しません。末尾の空白を保存するには、関数の引数として渡されるマクロ変数に値を割り当てる際に、%QUOTE 関数を使用します。%QUOTE 関数を使って末尾の空白を保存する必要があるかどうかを判定する場合、対象とする関数のドキュメントを参照し、その引数が数値のみ、文字のみ、数値と文字の両方のうちどれを受け入れるかを確認してください。数値と文字の両方を受け入れると記述されている引数の場合、%QUOTE 関数を使用して同引数に指定された値をクォートします。

```
%let j=1;
%let x=fax;
%let y=fedex;
%let z=phone;
%put j=&j x=&x y=&y z=&z
j=1 x=fax y=fedex z=phone
%syscall allperm(j,x,y,z);
%put j=&j x=&x y=&y z=&z
j=1 x=250 y=65246 z=phone
```

例: RANUNI CALL ルーチンを%SYSCALL と共に使用する

%SYSCALL ステートメントの使用例を次に示します。マクロステートメント%SYSCALL RANUNI(A,B)は、SAS CALL ルーチンである RANUNI を呼び出します。

注: RANUNI の構文は **RANUNI**(**seed**,**x**)です。

```
%let a = 123456;
```

```
%let b = 0;
%syscall ranuni(a,b);
%put &a, &b;
```

%PUT ステートメントは、マクロ変数 A および B の値を次のように SAS ログに出力します。

```
1587033266 0.739019954
```

%SYSEXEC ステートメント

動作環境のコマンドを発行します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “SYSSCP 自動マクロ変数と SYSSCPL 自動マクロ変数” (216 ページ)および“SYSRC 自動マクロ変数” (216 ページ)

構文

%SYSEXEC<*command*> ;

必須引数

引数なし
動作環境モードに移行し、そこで動作環境コマンドを発行した後、元の SAS セッションに戻ります。

command
任意の動作環境コマンドを指定します。*command* にセミコロンが含まれている場合、マクロクォーティング関数を使用します。

詳細

%SYSEXEC ステートメントは、ユーザーが指定したコマンドを動作環境に実行させます。動作環境からのリターンコードは自動マクロ変数 SYSRC に割り当てられます。%SYSEXEC ステートメントと自動マクロ変数 SYSSCP および SYSSCPL を組み合わせて使用することで、複数の動作環境で実行できる可搬性のあるマクロを作成できます。

動作環境の情報
%SYSEXEC ステートメントの使用に関連する次の事項は、動作環境ごとに固有となります。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

- バッチ処理、非対話型モード、対話型ラインモードでの%SYSEXEC ステートメントの利用可能性
- 引数なしで%SYSEXEC ステートメントを実行した場合に、動作環境モードから元の SAS セッションに戻る方法
- %SYSEXEC ステートメントで使用できるコマンド
- 自動マクロ変数 SYSRC に格納されるリターンコード

比較

%SYSEXEC ステートメントによる動作環境コマンドの呼び出しは、X ステートメントによる X ウィンドウ環境コマンドの呼び出しに似ています。ただし、X ステートメントによる X ウィンドウ環境コマンドの呼び出しとは異なり、%SYSEXEC ステートメントによる動作環境コマンドの呼び出しでは、呼び出すホストコマンドを引用符で囲む必要があります。

%SYSLPUT ステートメント

リモートホスト(サーバー)上で新しいマクロ変数を作成するか、またはリモートホスト上に保存されている既存のマクロ変数の値を変更します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

要件 SAS/CONNECT

参照項目: “%LET ステートメント” (307 ページ)および“%SYSRPUT ステートメント” (326 ページ)

構文

%SYSLPUT*macro-variable*=<*value*</REMOTE=*remote-session-identifier*>> ;

必須引数

macro-variable
マクロ変数名か、またはマクロ変数名を生成するマクロテキスト式を指定します。この名前は、リモートホストサーバー上の新規または既存のマクロ変数を参照します。

remote-session-identifier
リモートセッションの名前を指定します。

value
文字列か、または文字列を生成するマクロ式のいずれかを指定します。この値を省略するとヌル(長さがゼロの文字列)が生成されます。先頭および末尾にある空白は無視されます。これらの空白に意味がある場合、値を%STR 関数で囲みます。

詳細

SAS/CONNECT ソフトウェアを使用して、%SYSLPUT ステートメントをローカルホスト(クライアント)からリモートホスト(サーバー)にサブミットすることにより、リモートホスト(サーバー)上での新しい変数の作成や、リモートホスト(サーバー)上での既存のマクロ変数値の変更が行えます。

注: リモートホストおよびローカルホスト上のマクロ変数名の先頭には、アンパサンドを付けません。

リモートホスト上のマクロ変数の値をローカルホスト上のマクロ変数に割り当てるには、%SYSRPUT ステートメントを使用します。

%SYSLPUT ステートメントを使用する場合、SIGNON コマンドまたは SIGNON ステートメントを使って、ローカル SAS セッション(クライアント)とリモート SAS セッション(サー

バー)間のリンクを事前に初期化しておく必要があります。詳細については、SAS/CONNECT ソフトウェアのドキュメントを参照してください。

%SYSMACDELETE ステートメント

WORK.SASMACR カタログからマクロ定義を削除します。

種類: マクロステートメント

制限事項: マクロ定義およびオープンコードで使用可能

構文

%SYSMACDELETE *macro_name* </ *option*>;

必須引数

macro_name
マクロ名を指定するか、またはマクロ名を生成するテキスト式を指定します。

オプション引数

NOWARN
警告診断メッセージを発行しないよう指定します。

詳細

%SYSMACDELETE ステートメントは、指定されたマクロのマクロ定義を、WORK.SASMACR カタログから削除します。指定されたマクロ定義が WORK.SASMACR カタログ内に存在しない場合、WARNING 診断メッセージが発行されます。指定されたマクロが現在実行中である場合、ERROR 診断メッセージが発行されます。

%SYSMSTORECLEAR ステートメント

SASMSTORE=オプションに指定されているライブラリ参照名に関連付けられているコンパイル済みマクロカタログをクローズし、そのライブラリ参照名をクリアします。

種類: マクロステートメント

制限事項: マクロ定義およびオープンコードで使用可能

参照項目: SASMSTORE=システムオプション

構文

%SYSMSTORECLEAR;

詳細

%SYSMSTORECLEAR ステートメントを使うと、コンパイル済みマクロカタログをクローズし、SASMSTORE=ライブラリの切り替え時に以前のライブラリ参照名をクリアできます。

注: SASMSTORE=システムオプションで指定されたライブラリに含まれているコンパイル済みマクロが実行中である場合、次のアクションが実行されます。

- ERROR 診断メッセージが発行されます。
- 指定されたライブラリはクローズされません。
- ライブラリ参照名はクリアされません。

%SYSRPUT ステートメント

リモートホスト上にあるマクロ変数の値を、ローカルホスト上にあるマクロ変数に割り当てます。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

要件 SAS/CONNECT

参照項目: “SYSERR 自動マクロ変数” (204 ページ)、“SYSINFO 自動マクロ変数” (208 ページ)、および“%SYSLPUT ステートメント” (324 ページ)

構文

%SYSRPUT *local-macro-variable=remote-macro-variable*;

必須引数

local-macro-variable
先頭にアンパサンドが付いていないマクロ変数名か、またはそのようなマクロ変数名を生成するテキスト式を指定します。この名前は、ローカルホスト上に保存されているマクロ変数名でなければなりません。

remote-macro-variable
先頭にアンパサンドが付いていないマクロ変数名か、またはそのようなマクロ変数名を生成するテキスト式を指定します。この名前は、リモートホスト上に保存されているマクロ変数名でなければなりません。

詳細

SAS/CONNECT ソフトウェアを使用して%SYSRPUT ステートメントをリモートホストにサブミットすることで、リモートホスト上に保存されているマクロ変数の値を取得できます。%SYSRPUT ステートメントは、その取得した値をローカルホスト上にあるマクロ変数に割り当てます。%SYSRPUT ステートメントは、マクロ変数に値を割り当てるという意味では、%LET マクロステートメントに似ています。ただし、%SYSRPUT ステートメントは、同ステートメントが処理されるリモートホスト上の変数に対してではなく、ローカルホスト上の変数に値を割り当てます。%SYSRPUT ステートメントは、割り当て対象となるマクロ変数を、クライアントセッションにおけるグローバルシンボルテーブルに配置します。

注: リモートホストおよびローカルホスト上のマクロ変数名の先頭には、アンパサンドを付けません。

%SYSRPUT ステートメントは、自動マクロ変数 SYSINFO の値を取得し、その値をローカルホストに渡す場合に使用すると便利です。SYSINFO には、一部の SAS プロシジャが出力したリターンコードの情報が格納されます。SAS/CONNECT ソフトウェアの UPLOAD プロシジャおよび DOWNLOAD プロシジャは、どちらも自動マクロ変数 SYSINFO を更新し、エラーが原因で当該プロシジャが終了した場合には、同変数に

ゼロ以外の値を設定します。リモートホスト上で%SYSRPUT ステートメントを使用すると、SYSINFO マクロ変数の値をローカル SAS セッションに送り返すことができます。このようなジョブをリモートホストに対してサブミットすることにより、リモートホストまたはローカルホスト上で別のステップを開始する前に、PROC UPLOAD ステップまたは PROC DOWNLOAD ステップが正常終了したかどうかをチェックできます。

%SYSRPUT ステートメントの使い方に関する詳細は、SAS/CONNECT ソフトウェアのドキュメントを参照してください。

リモートホスト(サーバー)上で新しいマクロ変数を作成するか、またはリモートホスト上に保存されている既存のマクロ変数の値を変更するには、%SYSLPUT マクロステートメントを使用します。

例: リモートホストのリターンコード値のチェック

次の例では、ファイルをダウンロードし、非対話型のジョブに含まれているステップの成否に関する情報を戻す方法を示します。リモート処理が完了した後、このジョブは変数 RETCODE に保存されているリターンコードの値をチェックします。リモート処理が成功した場合、ローカルホスト上の処理を続行します。

%SYSRPUT ステートメントは、自動マクロ変数 SYSINFO に戻された値を取得し、その値をローカルホストに渡す場合に使用すると便利です。自動マクロ変数 SYSINFO には、SAS プロシジャで生成されるリターンコード情報が含まれています。次の例では、PROC DOWNLOAD ステップに続いて%SYSRPUT ステートメントを実行しています。その後、変数 SYSINFO に戻された値をチェックすることにより、先に実行した PROC DOWNLOAD ステップが成功したかどうかを調べています。

```
rsubmit;
%macro download;
proc download data=remote.mydata out=local.mydata;
run;
%sysrput retcode=&sysinfo;
%mend download;
%download
endrsubmit;
%macro checkit;
%if &retcode = 0 %then %do;
further processing on local host
%end;
%mend checkit;
%checkit
```

SAS/CONNECT のバッチ(非対話型)ジョブは、システム状態コードとして常に 0 を返します。このため、SAS/CONNECT の非対話型ジョブの成否を判定するには、%SYSRPUT マクロステートメントを使用して、自動マクロ変数 SYSERR の値をチェックする必要があります。SAS/CONNECT 会話の接続先がどのリモートシステムであるかを決定するには、次のステートメントをリモートサブミットします。

```
%sysrput rhost=&sysscp;
```

%WINDOW ステートメント

カスタマイズされたウィンドウを定義します。

種類: マクロステートメント

制限事項: マクロ定義またはオープンコードで使用可能

参照項目: “%DISPLAY ステートメント” (293 ページ)および“%INPUT ステートメント” (305 ページ)

構文

%WINDOW*window-name*<*window-option-1 <...window-option-n>*
group-definition-1 <...group-definition-n>> field-definition-1 <...field-definition-n>;

必須引数

window-name
ウィンドウ名を指定します。*Window-name* は SAS 名でなければなりません。

window-option-1 <...window-option-n>
ウィンドウの全般的な特性を指定します。フィールド定義やグループ定義の前に、すべてのウィンドウオプションを指定します。次のウィンドウオプションが使用できます。

COLOR=*color*
ウィンドウの背景色を指定します。ウィンドウの色やウィンドウフィールドの中身の色はデバイスにより異なります。*Color* には次のいずれかを指定できます。

BLACK
BLUE
BROWN
CYAN
GRAY (または GREY)
GREEN
MAGENTA
ORANGE
PINK
RED
WHITE
YELLOW

動作環境の情報
色の表現は、お使いのディスプレイデバイスにより異なる場合があります。また、ディスプレイデバイスによっては、背景色がウィンドウ全体に影響するものもあれば、ウィンドウ境界にのみ影響するものもあります。

COLUMNS=*columns*
境界を含むウィンドウ内のディスプレイ列の数を指定します。ウィンドウは任意の数の列を含むことができます。また、ウィンドウはディスプレイの境界を越えて拡張できます。この機能は、ウィンドウの開発に使用したディスプレイデバイスよりも表示部の広いデバイス上でウィンドウを表示しなければならない場合に便利です。デフォルトでは、ウィンドウはディスプレイ内の残りの列をすべて埋めます。

動作環境の情報
使用できる列の数は、お使いのディスプレイデバイスのタイプにより異なります。また、左境界および右境界は、お使いのデバイスに応じて、それぞれディスプレイの 0～3 列を使用します。様々なタイプのディスプレイデバイス向けのウィンドウを作成する場合、すべてのフィールドが最も狭いウィンドウ内でも表示できることを確認する必要があります。

ICOLUMN=*column*
ウィンドウが表示されるディスプレイ内の開始列を指定します。デフォルトでは、マクロプロセッサはディスプレイの列 1 からウィンドウを開始します。

IROW=*row*
ウィンドウが表示されるディスプレイ内の開始行を指定します。デフォルトでは、マクロプロセッサはディスプレイの行 1 からウィンドウを開始します。

KEYS=<<*libref.*>*catalog.*>*keys-entry*
ウィンドウ用のファンクションキー定義を含んでいる KEYS カタログエントリの名前を指定します。*libref* および *catalog* を省略した場合、SASUSER.PROFILE が使用されます。*keys-entry*.

KEYS=オプションを省略した場合、**KEYS** ウィンドウに定義されている現在のファンクションキー設定が使用されます。

MENU=<<*libref.*>*catalog.*>*pmenu-entry*
PMENU プロシジャを使って作成したメニューの名前を指定します。*libref* および *catalog* を省略した場合、SASUSER.PROFILE が使用されます。*pmenu-entry*.

ROWS=*rows*
境界を含むウィンドウ内のディスプレイ行の数を指定します。ウィンドウは任意の数の行を含むことができます。また、ウィンドウはディスプレイデバイスの境界を越えて拡張できます。この機能は、ウィンドウの開発に使用したディスプレイデバイスよりも表示部の広いデバイス上でウィンドウを表示しなければならない場合に便利です。この値を省略すると、ウィンドウは、ディスプレイデバイス内の残りの行をすべて埋めます。

動作環境の情報
使用できる行の数は、お使いのディスプレイデバイスのタイプにより異なります。

group-definition
グループ名を指定し、そのグループ内にあるすべてのフィールドを定義します。*group definition* は、GROUP=*group field-definition <. . . field-definition-n>* という形式を持ちます。ここで、*group* には、特定ウィンドウ内に表示したいフィールドグループの名前を指定します。ウィンドウには、任意の数のフィールドグループを含めることができます。GROUP=オプションを省略すると、ウィンドウには名前のないフィールドグループが 1 つだけ含められます。*group* は SAS 名でなければなりません。

フィールドをグループにまとめることにより、複数のコンテンツを含む単一のウィンドウを作成できます。特定のグループを参照するには、*window.group* を使用します。

field-definition
ウィンドウ内に表示したいマクロ変数または文字列、およびその説明を指定します。ウィンドウには、任意の数のフィールドを含めることができます。

フィールドを使うことで、表示するマクロ変数値(または定数値)、そのウィンドウ内の位置、その属性を指定できます。定数テキストは引用符で囲んで指定します。フィールドの位置は、開始行と開始列により決定されます。指定できる属性には、色、フィールドに値を入力できるかどうか、強調表示などの特性が含まれます。

マクロ変数を含むフィールド定義の形式は次のとおりです。

<row> <column> macro-variable<field-length> <options>

定数テキストを含むフィールド定義の形式は次のとおりです。

<row> <column>'text' | “text”<options>

フィールド定義の要素には次のものがあります。

row
マクロ変数や定数テキストを表示する行を指定します。各行指定は、ポインタ制御と、通常、数を生成するマクロ式から構成されます。次の行ポインタ制御を使用できます。

#macro-expression
マクロ式により表されるウィンドウ内の行を指定します。マクロ式は正の整数であるか、または正の整数を生成する式でなければなりません。

/ (*forward slash*)
ポインタを次の行の列 1 に移動します。

マクロプロセッサは、ウィンドウの表示時ではなく、ウィンドウの定義時にマクロ式を評価します。このため、フィールドの表示時には、フィールドの行位置は固定となります。

グループ内の最初のフィールド指定で *row* を省略した場合、マクロプロセッサはウィンドウの最初の行を使用します。それ以降のフィールド指定で *row* を省略した場合、マクロプロセッサは直前のフィールドと同じ行を使用します。

マクロプロセッサは、ウィンドウの最初の使用可能な行(ウィンドウ境界、コマンド行、メニューバー、メッセージ行を除く)を行 1 として取り扱います。

フィールド定義の先頭には、*row* または *column* のどちらかを指定する必要があります。

column
マクロ変数や定数テキストを開始する列を指定します。各列指定は、ポインタ制御と、通常、数を生成するマクロ式から構成されます。次の列ポインタ制御を使用できます。

@macro-expression
マクロ式により表されるウィンドウ内の列を指定します。マクロ式は正の整数であるか、または正の整数を生成する式でなければなりません。

+macro-expression
マクロ式により表される番号の列にポインタを移動します。マクロ式は正の整数であるか、または正の整数を生成する式でなければなりません。

マクロプロセッサは、ウィンドウの表示時ではなく、ウィンドウの定義時にマクロ式を評価します。このため、フィールドの表示時には、フィールドの列位置は固定となります。

マクロプロセッサは、左境界に接する列を列 1 として扱います。*column* を省略すると、列 1 が使用されます。

フィールド定義の先頭には *column* または *row* のどちらかを指定する必要があります。

macro-variable
表示するマクロ変数、またはユーザーがその位置に入力した値を受け取るマクロ変数の名前を指定します。この値は、マクロ変数名(マクロ変数参照ではない)であるか、またはマクロ変数名を生成するマクロ式でなければなりません。

デフォルトでは、表示されたウィンドウに値が含まれている場合、対応するマクロ変数値の入力や変更が行えます。ウィンドウに表示されている値をユーザーが変更できないようにするには、PROTECT=オプションを使用します。

注意:
フィールドが重ならないようにしてください。 あるフィールドが、同時に表示される別のフィールドの上に重ならないようにします。フィールドが重なると、マクロ変数値の不正な割り当てなどの、予期せぬ結果が引き起こされる場合

があります(一部のディスプレイデバイスでは、隣接するフィールドが空白で区切られていない場合、それらは重複フィールドとして扱われます)。フィールドが重複している場合、警告メッセージが SAS ログに出力されます。

field-length
マクロ変数値を表示するため、または入力を受け付けるために、現在の行内でどれだけの数の位置が利用できるかを示す整数を指定します。*field-length* の最大値は、行内にある残りの位置数になります。フィールド長は、1 行を超える長さには拡張できません。

注: フィールド長は、マクロ変数に保存されている値の長さには影響しません。フィールド長は、特定のフィールドに表示される文字数、または特定のフィールドで入力を受け付ける文字数にのみ影響します。

フィールドが既存のマクロ変数を含んでいる場合に *field-length* を省略すると、マクロプロセッサは、そのマクロ変数値の現在の長さに等しい値をフィールド長として使用します。この値の最大値は、行内に残されている位置数、または次のフィールドの開始までに残されている位置数になります。

注意:
フィールドがマクロ変数を含んでいる場合には、必ずフィールド長を指定します。
%GLOBAL または%LOCAL 変数で定義されたマクロ変数において、マクロ変数の現在の値がヌルである場合、マクロプロセッサはフィールド長としてゼロを使用します。ユーザーはこのフィールドにはいかなる文字も入力できません。

そのフィールド内でマクロ変数が生成される場合に *field-length* を省略すると、マクロプロセッサはフィールド長としてゼロを使用します。フィールドがマクロ変数を含んでいる場合には、必ずフィールド長を指定します。

'text' | "text"
表示する定数テキストを指定します。このテキストは、一重または二重引用符で囲む必要があります。ユーザーは定数テキストを含むフィールドには値を入力できません。

options
次のいずれかを指定できます。

ATTR=*attribute* | (*attribute-1* <. . . , *attribute-n*>) A=*attribute* | (*attribute-1* <. . . , *attribute-n*>)
フィールドの表示属性を制御します。利用可能な表示属性および同属性の組み合わせは、お使いのディスプレイデバイスにより異なります。

BLINK	フィールドを点滅させます。
HIGHLIGHT	フィールドを強調表示します。
REV_VIDEO	フィールドを反転表示します。
UNDERLINE	フィールドを下線付きで表示します。

AUTOSKIP=YES | NO AUTO=YES | NO
ユーザーがフィールドのすべての位置にデータを入力した場合、現在のウィンドウまたはグループ内にある次の非保護フィールドにカーソルを移動するかどうかを制御します。AUTOSKIP=YES を指定すると、カーソルは自動的に次の非保護フィールドに移動します。AUTOSKIP=NO を指定すると、カーソルは自動的には移動しません。デフォルト値は AUTOSKIP=YES です。

COLOR=*color* C=*color*
フィールドの色を指定します。デフォルトの色は、デバイスにより異なります。*Color* には次のいずれかを指定できます。

BLACK
BLUE
BROWN
CYAN
GRAY (または GREY)
GREEN
MAGENTA
ORANGE
PINK
WHITE
YELLOW

DISPLAY=YES | NO
ユーザーがマクロ変数に値を入力する際に、その入力文字を表示するかどうかを指定します。DISPLAY=YES (デフォルト値)を指定すると、ユーザーが入力した文字が表示されます。DISPLAY=NO を指定すると、ユーザーが入力した文字は表示されません。

ユーザーがパスワードなどの機密情報を入力しなければならないアプリケーションでは、DISPLAY=NO を指定すると便利です。DISPLAY=オプションは、マクロ変数を含んでいるフィールドでのみ使用します。定数テキストは自動的に表示されます。

PROTECT=YES | NO P=YES | NO
マクロ変数を含んでいるフィールドに対してユーザーが情報を入力できるようにするかどうかを指定します。PROTECT=NO (デフォルト値)を指定すると、そのフィールドにはユーザーが情報を入力できます。PROTECT=YES を指定すると、そのフィールドにはユーザーが情報を入力できません。PROTECT=オプションは、マクロ変数を含んでいるフィールドでのみ使用します。定数テキストを含んでいるフィールドは自動的に保護されます。

REQUIRED=YES | NO
フィールドに含まれているマクロ変数にユーザーが値を入力する必要があるかどうかを指定します。REQUIRED=YES を指定すると、そのフィールドに値を入力しない限り、現在のウィンドウの表示が消えなくなります。必須フィールドにはヌル値を指定できません。REQUIRED=NO (デフォルト値)を指定すると、そのフィールドに値を入力しなくても、現在のウィンドウの表示を消すことができます。ウィンドウのコマンド行にコマンドを入力することで、REQUIRED=YES の効果を取り除くことができます。

詳細

%WINDOW ステートメントを使用すると、マクロプロセッサにより制御されるカスタマイズされたウィンドウを定義できます。これらのウィンドウにはコマンド行とメッセージ行があります。これらのウィンドウを使用して、テキストの表示や入力の受け付けが行えます。また、ウィンドウ環境コマンドの呼び出し、ファンクションキーの割り当て、PMENU 機能により作成されたメニューの使用も行えます。

ウィンドウは呼び出す前に定義する必要があります。%WINDOW ステートメントはマクロウィンドウを定義します。一方、%DISPLAY ステートメントは、マクロウィンドウを表示します。マクロウィンドウはいったん定義されると、SAS セッションが終了するまで

存在します。ウィンドウの表示や再定義は、SAS セッションにおける任意の時点で行えます。

マクロ定義内でマクロウィンドウを定義した場合、そのマクロを実行するたびに、同ウィンドウがマクロプロセッサにより再定義されます。定義が変化しないウィンドウを繰り返し表示する場合、次のいずれかを行うと、マクロ処理がより効率的になります。

- マクロの外でウィンドウを定義すること。
- ウィンドウを表示するマクロではなく、1 度だけ実行するマクロ内でウィンドウを定義すること。

%WINDOW ステートメントに新しいマクロ変数の名前が含まれている場合、マクロプロセッサは、その変数を現在のスコープで作成します。%WINDOW ステートメントは、次に示す 2 つの自動マクロ変数を作成します。

SYSCMD
ウィンドウのコマンド行から入力された最後のコマンド(ウィンドウ環境により認識されなかったコマンド)を含んでいます。

SYSMSG
ユーザーがメッセージ行に表示するよう指定したテキストを含んでいます。

注: ウィンドウ環境におけるファイル管理、スクロール、検索、編集の各コマンドは、マクロウィンドウでは利用できません。

例

例 1: アプリケーションの WELCOME ウィンドウの作成

次の%WINDOW ステートメントは、単一のフィールドグループを含むウィンドウを作成します。

```
%window welcome color=white
#5 @28 'Welcome to SAS.' attr=highlight
color=blue
#7 @15
"You are executing Release &sysver on &sysday, &sysdate.."
#12 @29 'Press ENTER to continue.';
```

WELCOME ウィンドウは、ディスプレイデバイス全体を埋めます。このウィンドウの背景色は白、最初のテキスト行の色は青で強調表示されます。それ以降の 2 行の色は黒で、通常表示されます。**WELCOME** ウィンドウはユーザーによる値の入力を必要としません。ただし、このウィンドウの表示を消すには、ENTER キーを押す必要があります。

注: マクロ変数 SYSVER、SYSDAY、SYSDATE を参照するために、区切り文字として 2 つの連続するピリオドが必要となります。

例 2: 入力情報を使ってマクロ変数を作成する

次の例では、ユーザーに情報の入力を求め、その情報を使用してマクロ変数を作成しています。

```
%window info
#5 @5 'Please enter userid:'
#5 @26 id 8 attr=underline
#7 @5 'Please enter password:'
#7 @28 pass 8 attr=underline display=no;
%display info;
```

```
%put userid entered was &id;
%put password entered was &pass;
```

20 章

マクロのシステムオプション

マクロのシステムオプション

マクロ機能に適用される SAS システムオプションが複数存在します。

ディクショナリ

CMDMAC システムオプション

コマンドスタイルのマクロ呼び出しを制御します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOCMDMAC

構文

CMDMAC | NOCMDMAC

必須引数

CMDMAC
マクロプロセッサが、各ウィンドウ環境コマンドの先頭ワードを調べて、それがコマンドスタイルのマクロ呼び出しであるかどうかを判定するようにします。

注: CMDMAC オプションを有効にすると、SAS システムはまずマクロライブラリを検索し、発行されたウィンドウ環境コマンドの先頭ワードと同じ名前のマクロが見つかった場合、そのマクロを実行します。この場合、予期せぬ結果が発生することがあります。

NOCMDMAC
コマンドスタイルのマクロ呼び出しに関して、いかなるチェックも行いません。NOCMDMAC オプションを有効にしている場合にマクロプロセッサがコマンドスタイルのマクロ呼び出しを検出すると、マクロプロセッサはその呼び出しを SAS コマンドとして扱い、そのコマンドが無効な場合や使用法が誤っている場合にはエラーメッセージを出力します。

詳細

CMDMAC システムオプションは、コマンドスタイルマクロとして定義されたマクロがコマンドスタイルで呼び出されるかどうかを制御します。または、そのようなマクロをネームスタイルのマクロ呼び出しとして呼び出す必要があるかどうかを制御します。次の 2 つの例は、コマンドスタイルのマクロ呼び出しとネームスタイルのマクロ呼び出しの例をそれぞれ表しています。

- *macro-name parameter-value-1 parameter-value-2*
- *%macro-name*(*parameter-value-1, parameter-value-2*)

CMDMAC オプションを指定すると、マクロ機能は、コマンドライン上の先頭ワードに対応する名前を見つけようとして、現在のセッション中にコンパイルされたマクロを検索するため、処理時間が増大します。MSTORED オプションが有効である場合、コマンドライン上の先頭ワードに対応する名前を見つけるために、コンパイル済みマクロを含んでいるライブラリが検索されます。MAUTOSOURCE 有効である場合、コマンドライン上の先頭ワードに対応する名前を見つけるために、自動呼び出しライブラリが検索

されます。さらに MRECALL システムオプションも有効である場合、前回の検索でワードが見つからなかった場合でも検索が続行されるため、より多くの処理時間がかかる可能性があります。

どのオプションが有効であるかにかかわらず、ネームスタイルの呼び出しを使用すれば、コマンドスタイルマクロを含むすべてのマクロを呼び出すことができます。

比較

マクロを呼び出す場合には、ネームスタイルのマクロ呼び出しを使用する方が効率的です。ネームスタイルのマクロ呼び出しでは、マクロプロセッサは、パーセント記号に続くワードに対応するマクロ名のみを検索するためです。

IMPLMAC システムオプション

ステートメントスタイルのマクロ呼び出しを制御します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOCMDMAC

構文

IMPLMAC | NOIMPLMAC

必須引数

IMPLMAC
マクロプロセッサが、サブミットされた各ステートメントの先頭ワードを調べて、それがステートメントスタイルのマクロ呼び出しであるかどうかを判定するようにします。

注: IMPLMAC オプションを有効にすると、SAS システムはまずマクロライブラリを検索し、サブミットされた SAS ステートメントの先頭ワードと同じ名前のマクロが見つかった場合、そのマクロを実行します。この場合、予期せぬ結果が発生することがあります。

NOIMPLMAC
ステートメントスタイルのマクロ呼び出しに関して、いかなるチェックも行いません。これがデフォルトの設定です。NOIMPLMAC オプションが有効である場合にマクロプロセッサがステートメントスタイルのマクロ呼び出しを検出すると、マクロプロセッサはその呼び出しを SAS ステートメントとして扱います。そのコマンドが無効な場合や使用法が誤っている場合にはエラーメッセージが出力されます。

詳細

IMPLMAC システムオプションは、ステートメントスタイルマクロとして定義されたマクロがステートメントスタイルで呼び出されるかどうかを制御します。または、そのようなマクロをネームスタイルのマクロ呼び出しとして呼び出す必要があるかどうかを制御します。次の 2 つの例は、ステートメントスタイルのマクロ呼び出しとネームスタイルのマクロ呼び出しの例をそれぞれ表しています。

- *macro-name parameter-value-1 parameter-value-2;*

- *%macro-name*(*parameter-value-1, parameter-value-2*)

IMPLMAC オプションを指定すると、マクロ機能は、各 SAS ステートメントの先頭ワードに対応する名前を見つけようとして、現在のセッション中にコンパイルされたマクロを検索するため、処理時間が増大します。MSTORED オプションが有効である場合、各 SAS ステートメントの先頭ワードに対応する名前を見つけるために、コンパイル済みマクロを含んでいるライブラリが検索されます。MAUTOSOURCE 有効である場合、各 SAS ステートメントの先頭ワードに対応する名前を見つけるために、自動呼び出しライブラリが検索されます。さらに MRECALL システムオプションも有効である場合、前回の検索でワードが見つからなかった場合でも検索が続行されるため、より多くの処理時間がかかる可能性があります。

どのオプションが有効であるかにかかわらず、ネームスタイルの呼び出しを使用すれば、ステートメントスタイルマクロを含むすべてのマクロを呼び出すことができます。

注: 自動呼び出しライブラリやコンパイル済みマクロカタログ内のメンバが既存のウィンドウ環境コマンドと同じ名前を持つ場合、CMDMAC オプションが有効であるならば、SAS システムはまずマクロを検索します。この場合、予期せぬ結果が発生することがあります。

比較

マクロを呼び出す場合には、ネームスタイルのマクロ呼び出しを使用する方が効率的です。ネームスタイルのマクロ呼び出しでは、マクロプロセッサは、パーセント記号に続くワードに対応するマクロ名のみを検索するためです。

MACRO システムオプション

SAS マクロ言語が使用できるかどうかを制御します。

該当要素: 構成ファイル、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: MACRO

構文

MACRO | NOMACRO

必須引数

MACRO
SAS システムがマクロ言語ステートメント、マクロ呼び出し、マクロ変数参照を認識し、それらを処理できるようします。

NOMACRO
SAS システムが、マクロ言語ステートメント、マクロ呼び出し、マクロ変数参照の認識や処理を行わないようにします。通常、項目が SAS システムにより認識されない場合、エラーメッセージが表示されます。マクロ機能を SAS ジョブで使用しない場合、NOMACRO オプションを指定すると、マクロやマクロ変数のチェックに関するオーバーヘッドがなくなるため、性能をわずかに向上させることができます。

MAUTOCOMPLOC システムオプション

自動呼び出しマクロのコンパイル時に、自動呼び出しマクロのソースの場所を SAS ログに表示します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMAUTOCOMPLOC

構文

MAUTOCOMPLOC|NOMAUTOCOMPLOC

必須引数

MAUTOCOMPLOC
自動呼び出しマクロのコンパイル時に、自動呼び出しマクロのソースの場所を SAS ログに表示します。

NOMAUTOCOMPLOC
自動呼び出しマクロのソースの場所が SAS ログに出力されないようにします。

詳細

MAUTOCOMPLOC システムオプションにより作成される、自動呼び出しマクロのソースの場所に関する SAS ログ内の表示は、MAUTOLOCDISPLAY または MLOGIC システムオプションによる影響を受けません。

MAUTOLOCDISPLAY システムオプション

自動呼び出しマクロの呼び出し時に、自動呼び出しマクロのソースの場所をログに表示します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMAUTOLOCDISPLAY

構文

MAUTOLOCDISPLAY | NOMAUTOLOCDISPLAY

必須引数

MAUTOLOCDISPLAY
自動呼び出しマクロの呼び出し時に、自動呼び出しマクロのソースの場所をログに表示します。

NOMAUTOLOCDISPLAY
自動呼び出しマクロの呼び出し時に、自動呼び出しマクロのソースの場所をログに表示しません。デフォルトの設定は NOMAUTOLOCDISPLAY です。

詳細

MAUTOLOCDISPLAY と MLOGIC の両オプションを指定すると、自動呼び出しマクロのソースの場所に関する MLOGIC リストのみがログに表示されます。

MAUTOLOCINDES システムオプション

マクロプロセッサが自動呼び出しソースファイルのフルパス名を、WORK.SASMACR カタログのコンパイル済み自動呼び出しマクロ定義のカタログエントリの説明フィールドに追加するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時
PROC OPTIONS GROUP= MACRO
種類: システムオプション
デフォルト: NOMAUTOLOCINDES
参照項目: SAS log

構文

MAUTOLOCINDES|NOMAUTOLOCINDES

必須引数

MAUTOLOCINDES
マクロプロセッサが自動呼び出しソースファイルのフルパス名を、WORK.SASMACR カタログのコンパイル済み自動呼び出しマクロ定義のカタログエントリの説明フィールドに追加するようにします。

NOMAUTOLOCINDES
自動呼び出しマクロ定義の説明フィールドに対する変更を行いません。

詳細

MAUTOLOCINDES オプションは、自動呼び出しマクロのソースが置かれている場所を決定する場合に便利です。次の例では、フルパス名を含む出力を表示します。

```
options mautolocindes;
%put %lowcase(THIS);

this

proc catalog cat=work.sasmacr;contents;run;

Contents of Catalog WORK.SASMACR

# Name Type Create Date Modified Date Description

1 LOWCASE MACRO 12Sep10:10:36:57 12Sep10:10:36:57 C:\SASv9\sas\dev\
```

```
mva-v930\shell\auto\
```

MAUTOSOURCE システムオプション

自動呼び出し機能が利用できるかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: MAUTOSOURCE

構文

MAUTOSOURCE | NOMAUTOSOURCE

必須引数

MAUTOSOURCE
マクロ名が WORK ライブラリ内に見つからない場合、マクロプロセッサが、自動呼び出しライブラリを検索して、要求された名前を持つマクロを見つけるようにします。

NOMAUTOSOURCE
マクロ名が WORK ライブラリ内に見つからない場合、マクロプロセッサは自動呼び出しライブラリを検索しません。

詳細

マクロ機能がマクロを検索する場合、現在の SAS セッションでコンパイルされたマクロを最初に検索します。MSTORED オプションが有効である場合、マクロ機能は、コンパイル済みマクロを含んでいるライブラリを次に検索します。MAUTOSOURCE オプションが有効である場合、マクロ機能は、自動呼び出しマクロライブラリを次に検索します。

MCOMPILENOTE システムオプション

NOTE (注釈)を SAS ログに出力します。注釈には、マクロのコンパイルを実行するために使用された命令のサイズと数が含まれます。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NONE

構文

MCOMPILENOTE=<NONE | NOAUTOCALL | ALL>

必須引数

NONE
注釈がログに出力されないようにします。

NOAUTOCALL
自動呼び出しマクロに関する注釈がログに出力されないようにします。ただし、それ以外のマクロのコンパイルの実行に関する注釈はログに出力されます。

ALL
注釈をログに出力します。注釈には、マクロのコンパイルを実行するために使用された命令のサイズと数が含まれます。

詳細

注釈を見ることで、マクロのコンパイルが完了したことを確認できます。このオプションを有効にした場合、注釈が出力された時点で、当該マクロのコンパイル済みバージョンが実行できるようになります。マクロが正常にコンパイルされた場合でも、エラーや警告メッセージが注釈に出力されているならば、そのマクロが意図した通りに動作しない可能性があります。

例: MCOMPILENOTE システムオプションの使用

マクロが正常にコンパイルされた場合でも、注釈にエラーが含まれていることがあります。エラーがない場合の注釈の例を次に示します。

```
option mcompilenote=noautocall;
%macro mymacro;
%mend mymacro;
```

これらのステートメントを実行すると、次の行がログに出力されます。

```
NOTE: The macro MYMACRO completed compilation without errors.
```

エラーを含む場合の注釈の例を次に示します。

```
%macro yourmacro;
%end;
%mend yourmacro;
```

これらのステートメントを実行すると、次の行がログに出力されます。

```
ERROR: There is no matching %DO statement for the %END statement.
This statement will be ignored.
NOTE: The macro YOURMACRO completed compilation with errors.
```

MCOMPILE システムオプション

新しいマクロ定義を許可するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: MCOMPILE

構文

MCOMPILE | NOMCOMPILE

必須引数

MCOMPILE
新しいマクロ定義を許可します。

NOMCOMPILE
新しいマクロ定義を許可しません。

詳細

MCOMPILE システムオプションは、新しいマクロ定義を許可します。

NOMCOMPILE システムオプションは、新しいマクロ定義を許可しません。
NOMCOMPILE システムオプションは、既存のコンパイル済みマクロや自動呼び出しマクロの使用は禁止しません。

MCOVERAGE システムオプション

カバレッジ分析データの生成を可能にします。

該当要素:	構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時
PROC OPTIONS GROUP=	マクロ
種類:	システムオプション
デフォルト:	NOMCOVERAGE
要件	MCOVERAGELOC=システムオプションを使用すること

構文

MCOVERAGE|NOMCOVERAGE

必須引数

MCOVERAGE
カバレッジ分析データの生成を有効化します。

NOMCOVERAGE
カバレッジ分析データの生成を行いません。

詳細

MCOVERAGE システムオプションは、*カバレッジ分析データ*の生成を制御します。同データは、SAS ソリューション製品がリリース前に適切なテストが行われていることを保証するために必要となります。

カバレッジ分析データのフォーマットは、空白で区切られたフラットなテキストファイルであり、3 種類のレコードを含んでいます。各レコードは、数値レコード型で始まります。データ内の行番号は、マクロ定義に使用された%MACRO キーワードを基準とする相対的な行番号になります。カバレッジ分析データファイルの保存場所を指定するには、MCOVERAGELOC=システムオプションを使用します。詳細は、“MCOVERAGELOC=システムオプション” (346 ページ)を参照してください。

注: ネストされたマクロ定義は折り畳み改行を含むモデルテキストとして保存されるため、後で実行カバレッジの分析対象とする予定のマクロ定義では、ネストされたマクロ定義を使用しないことを推奨します。

3 つのレコード型の説明を次に示します。

レコード型 1:

1 n n macroname

1
: レコード型

n
: 最初の行番号

n
: 最後の行番号

macroname
: マクロ名

レコード型 1 は、マクロ実行の開始を表します。レコード型 1 は、マクロの実行ごとに一度だけ出力されます。

レコード型 2:

2 n n macroname

2
: レコード型

n
: 最初の行番号

n
: 最後の行番号

macroname
: マクロ名

レコード型 2 は、実行されたマクロの行を表します。マクロの単一行により、複数のレコードが生成される場合があります。

レコード型 3:

3 n n macroname

3
: レコード型

n
: 最初の行番号

n
: 最後の行番号

macroname
: マクロ名

レコード 3 は、マクロの行のうち、その行からはコードが生成されなかったために実行できなかった行を表します。これらの行は、コメント行であるか、またはマクロコードを生成しない行になります。

サンプルのプログラムのログを次に示します。

```
Sample Program Log:
NOTE: Copyright (c) 2002-2008 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software 9.3 (TS1B0)
```

```
Licensed to SAS Institute Inc., Site 1.
NOTE: This session is executing on the XP_PRO platform.



NOTE: SAS initialization used:
real time 0.45 seconds
cpu time 0.20 seconds

1 options source source2;
2
3 options mcoverage mcoverageloc='./foo.dat';
4
5 /* 1 */ %macro
6 /* 2 */ foo (
7 /* 3 */ arg,
8 /* 4 */
9 /* 5 */
10 /* 6 */ arg2
11 /* 7 */
12 /* 8 */
13 /* 9 */ =
14 /* 10 */
15 /* 11 */ This is the default value of arg2)
16 /* 12 */ ;
17 /* 13 */ /* This is a number of lines of comments */
18 /* 14 */ /* which presumably will help the maintainer */
19 /* 15 */ /* of this macro to know what to do to keep */
20 /* 16 */ /* this silly piece of code current */
21 /* 17 */ %if &arg %then %do;
22 /* 18 */ data _null_;
23 /* 19 */ x=1;
24 /* 20 */ %end;
25 /* 21 */ %* this is a macro comment statement
26 /* 22 */ that also can be used to document features
27 /* 23 */ and other stuff about the macro;
28 /* 24 */ %else
29 /* 25 */ %do;
30 /* 26 */ DATA _NULL_;
31 /* 27 */ y=1;
32 /* 28 */ %end;
33 /* 29 */ run;
34 /* 30 */
35 /* 31 */
36 /* 32 */
37 /* 33 */
38 /* 34 */
39 /* 35 */ %mend
40 /* 36 */
41 /* 37 */
42 /* 38 */
43 /* 39 */
44 /* 40 */
45 /* 41 */
46 /* 42 */
47 /* 43 */ foo This is text which should generate a warning! ;
```

```
WARNING: Extraneous information on %MEND statement ignored for macro
definition FOO.
```

MCOVERAGELOC=システムオプション

カバレッジ分析データファイルの場所を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= マクロ

種類: システムオプション

要件 MCOVERAGE システムオプションと共に使用すること

参照項目: “MCOVERAGE システムオプション” (343 ページ)

構文

MCOVERAGELOC=*filеref\file_specification*

必須引数

filеref\file_specification
SAS ファイル参照名か、または引用符で囲んだ外部ファイル名を指定します。

詳細

MCOVERAGELOC=システムオプションは、カバレッジ分析データファイルが出力される場所を指定します。このオプションの値には、SAS ファイル参照名か、または引用符で囲んだ外部ファイル名を指定します。

MERROR システムオプション

マクロ参照が置換できない場合に、マクロプロセッサが警告メッセージを発行するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: MERROR

構文

MERROR | NOMERROR

必須引数

MERROR
マクロプロセッサがマクロ参照をコンパイル済みマクロに対応付けることができない場合に、次の警告メッセージを発行します。

```
WARNING: Apparent invocation of macro %text not resolved.
```

NOMERROR
マクロプロセッサがマクロ参照をコンパイル済みマクロに対応付けることができない場合に、警告メッセージを発行しません。

詳細

マクロ参照が置換されない場合、いくつかの理由が考えられます。たとえば、次の理由が挙げられます。

- マクロ名のスペルが間違っている
- マクロが定義される前に呼び出されている
- パーセント記号を含む文字列が検出された次に例を示します。

```
TITLE Cost Expressed as %Sales;
```

マクロキーワードと間違えられる可能性のあるパーセント記号を含む文字列がプログラム内に存在する場合、NOMERROR オプションを指定します。

MEXECNOTE システムオプション

マクロ呼び出し時に、マクロ実行情報を SAS ログに表示するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMEXECNOTE

参照項目: MEXECSIZE (347 ページ)

構文

MEXECNOTE | NOMEXECNOTE

必須引数

MEXECNOTE
マクロの呼び出し時に、マクロの実行情報をログに表示します。

NOMEXECNOTE
マクロの呼び出し時に、マクロの実行情報をログに表示しません。

詳細

MEXECNOTE オプションは、マクロの実行モードを示す NOTE (注釈)の生成を制御します。

MEXECSIZE システムオプション

メモリ内で実行可能なマクロの最大サイズを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: 65536

参照項目: MEXECNOTE (347 ページ)および MCOMPILENOTE (341 ページ)

構文

MEXECSIZE=*n* | *n*K | *n*M | *n*G | *n*T | *hex*X | MIN | MAX

必須引数

n
メモリ内で実行するマクロの最大サイズをバイト単位で指定します。

***n*K**
メモリ内で実行するマクロの最大サイズをキロバイト単位で指定します。

***n*M**
メモリ内で実行するマクロの最大サイズをメガバイト単位で指定します。

***n*G**
メモリ内で実行するマクロの最大サイズをギガバイト単位で指定します。

***n*T**
メモリ内で実行するマクロの最大サイズをテラバイト単位で指定します。

MIN
メモリ内で実行するマクロの最小サイズを指定します。最小値はゼロです。

MAX
メモリ内で実行するマクロの最大サイズを指定します。

***hex*X**
メモリ内で実行するマクロの最大サイズを 16 進数で指定します。16 進数の末尾には X を付加します。

詳細

MEXECSIZE オプションを使うと、メモリ内で実行可能なマクロの最大サイズを制御できます。これは、ファイルとして実行されるマクロの最大サイズとは異なります。MEXECSIZE オプションで指定する値は、コンパイル済みマクロのサイズです。メモリは、マクロの実行時にのみ割り当てられます。マクロの実行が完了すると、割り当てられていたメモリは解放されます。マクロを実行するメモリが利用できない場合、メモリ不足を示すメッセージが SAS ログに出力されます。コンパイル済みマクロのサイズを SAS ログに出力するには、MCOMPILENOTE オプションを使用します。MEMSIZE オプションは、MEXECSIZE オプションには影響を与えません。

MFILE システムオプション

MPRINT 出力を外部ファイルに送るかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMFILE

要件 MPRINT オプション

参照項目: “MPRINT システムオプション” (355 ページ)

構文

MFILE | NOMFILE

必須引数

MFILE
MPRINT オプションにより生成される出力を外部ファイルに送ります。このオプションはデバッグを行う場合に使うと便利です。

NOMFILE
外部ファイルに MPRINT 出力を送りません。

詳細

MFILE オプションを使用する場合、MPRINT オプションを有効にできる必要があります。また、ファイル参照名 MPRINT を使用して外部ファイルを割り当てる必要があります。マクロの実行時に MPRINT オプションにより SAS ログに表示されるマクロが生成するコードが、ファイル参照名 MPRINT により参照される外部ファイルに出力されます。

MPRINT がファイル参照名として割り当てられていない場合や、ファイルにアクセスできない場合、警告メッセージが SAS ログに出力され、MFILE オプションがオフになります。この機能を利用できるようにするには、MFILE オプションを再度指定し、ファイル参照名 MPRINT をサクセス可能なファイルに割り当てる必要があります。

MINDELIMITER=システムオプション

マクロ演算子 IN で区切り文字として使用する文字を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: 1 個の空白

参照項目: “MINOPERATOR システムオプション” (351 ページ)および“%MACRO ステートメント” (309 ページ)

構文

MINDELIMITER=<”*option*”>

必須引数

option

1 つの文字を二重引用符または一重引用符で囲んで指定します。この文字が、マクロ演算子 IN での区切り文字として使用されます。二重引用符の使用例は次にようになります。

```
mindelimiter=",";
```

一重引用符の使用例は次にようになります。

```
mindelimiter=',';
```

詳細

MINDELIMITER=オプションの値は、大文字小文字を区別します。また、この値の最大長は 1 文字です。MINDELIMITER=オプションのデフォルト値は 1 個の空白です。

演算子 IN の代わりに、文字#を使用できます。

注: 演算子 IN または#をマクロで使用する場合、そのマクロの実行時に使用される区切り文字は、同マクロのコンパイル時に指定されていた MINDELIMITER=オプションの値になります。MINDELIMITER=システムオプションの現在の値ではなく、特定のマクロの実行時にそのマクロに固有の区切り文字を使用したい場合、その区切り文字をマクロ定義ステートメントで指定します。

```
%macro macroname / mindelimiter=',';
```

比較

次の例では、IN 演算子で使用する区切り文字を、デフォルト値の空白からカンマに変更しています。

```
%put %eval(a in d,e,f,a,b,c); /* should print 0 */
%put %eval(a in d e f a b c); /* should print 1 */
option mindelimiter=',';
%put %eval(a in d,e,f,a,b,c); /* should print 1 */
%put %eval(a in d e f a b c); /* should print 0 */
```

これらのステートメントを実行すると、次の行が SAS ログに出力されます。

```
NOTE: Copyright (c) 2007-2008 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version 9.2 (TS A0)
Licensed to SAS Institute Inc., Site 0000000001.
NOTE: This session is executing on the WIN_NT platform.
NOTE: SAS initialization used:
real time 1.02 seconds
cpu time 0.63 seconds
%put %eval(a in d,e,f,a,b,c); /* should print 0 */
0
%put %eval(a in d e f a b c); /* should print 1 */
1
option mindelimiter=',';
%put %eval(a in d,e,f,a,b,c); /* should print 1 */
1
%put %eval(a in d e f a b c); /* should print 0 */
0
```

MINOPERATOR システムオプション

マクロプロセッサが論理演算子 IN (#)を認識し評価するかどうかを制御します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMINOPERATOR

構文

MINOPERATOR | NOMINOPERATOR

必須引数

MINOPERATOR
マクロプロセッサが、ニーモニック演算子 **IN** および特殊文字**#**の両者を式における論理演算子として認識し、それらを評価するようにします。

NOMINOPERATOR
マクロプロセッサが、ニーモニック演算子 **IN** および特殊文字**#**の両者を通常の文字として認識するようにします。

詳細

IN (**#**)を式の中で演算子として使用したい場合、MINOPERATOR システムオプションを使用するか、または%MACRO ステートメント内で同オプションを使用します。

```
options minoperator;
```

特定のマクロの実行時に評価される式の中で IN または#を演算子として使用したい場合、そのマクロ定義内で MINOPERATOR キーワードを指定します。

```
%macro macroname / minoperator;
```

マクロ演算子 IN は DATA ステップの IN 演算子に似ていますが、両者は同じものではありません。両者の違いを次に示します。

- マクロ演算子 IN では、数値配列を検索できません。
- マクロ演算子 IN では、文字配列を検索できません。
- コロン(**:**)は、範囲を指定する簡略表記(たとえば、1～10 までの範囲を表す場合 **1:10** と表記する)としては認識されません。範囲を表すには、マクロ内で次のように指定する必要があります。

  ```
  %eval(3 in 1 2 3 4 5 6 7 8 9 10);
  ```

- リスト要素のデフォルトの区切り文字は空白になります。詳細は “MINDELIMITER=システムオプション” (349 ページ)を参照してください。
- IN 演算子にはその前後に 2 つのオペランドがありますが、これら両方のオペランドに値を含める必要があります。

  ```
  %put %eval(a IN a b c d); /*Both operands are present. */
  ```

  どちらかのオペランドにヌル値が含まれている場合、エラーが生成されます。

```
%put %eval( IN a b c d); /*Missing first operand. */
```

または

```
%put %eval(a IN); /*Missing second operand. */
```

IN 演算子の前後に指定するオペランドのどちらにヌル値が含まれていた場合でも、同じエラーが SAS ログに出力されます。

```
ERROR: Operand missing for IN operator in argument to %EVAL function.
```

次の例では、マクロ演算子 IN を使用して文字列を検索しています。

```
%if &state in (NY NJ PA) %then %let &region = %eval(&region + 1);
```

詳細は“演算式と論理式の定義 ” (74 ページ)を参照してください。

MLOGIC システムオプション

マクロプロセッサがデバッグ用にマクロの実行をトレースするかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO
LOGCONTROL

種類: システムオプション

デフォルト: NOMLOGIC

参照項目: “SAS ログ” (*SAS 言語リファレンス: 解説編* 9 章)

構文

MLOGIC | NOMLOGIC

必須引数

MLOGIC
マクロプロセッサがマクロの実行をトレースし、そのトレース情報を SAS ログに出力するようにします。このオプションはデバッグを行う場合に使うと便利です。

NOMLOGIC
マクロの実行をトレースしません。マクロのデバッグを行わない場合は、このオプションを使用します。

詳細

マクロをデバッグする場合、MLOGIC オプションを使用します。MLOGIC オプションにより生成される各行には、プレフィックスとして MLOGIC(*macro-name*):が出力されます。MLOGIC オプションが有効である場合にマクロプロセッサがマクロ呼び出しを検出すると、マクロプロセッサは次のものを特定するメッセージを表示します。

- マクロ実行の開始
- 呼び出し時のマクロパラメータの値
- 各マクロプログラムステートメントの実行
- 各%IF 条件の true または false
- マクロ実行の終了

注: MLOGIC オプションを指定すると、大量の出力が生成されます。

マクロのデバッグに関する詳細は、“マクロ機能のエラーメッセージとデバッグ” (121 ページ)を参照してください。

例: マクロの実行のトレース

次の例では、MLOGIC オプションを使用することにより、マクロ MKTITLE および RUNPLOT の実行をトレースしています。

```
%macro mktitle(proc,data);
title "%upcase(&proc) of %upcase(&data)";
%mend mktitle;
%macro runplot(ds);
%if %sysprod(graph)=1 %then
%do;
%mktitle (gplot,&ds)
proc gplot data=&ds;
plot style*price
/ haxis=0 to 150000 by 50000;
run;
quit;
%end;
%else
%do;
%mktitle (plot,&ds)
proc plot data=&ds;
plot style*price;
run;
quit;
%end;
%mend runplot;
options mlogic;
%runplot(sasuser.houses)
```

このプログラムを実行すると、次のような MLOGIC 出力が SAS ログに表示されます。

```
MLOGIC(RUNPLOT): Beginning execution.
MLOGIC(RUNPLOT): Parameter DS has value sasuser.houses
MLOGIC(RUNPLOT): %IF condition %sysprod(graph)=1 is TRUE
MLOGIC(MKTITLE): Beginning execution.
MLOGIC(MKTITLE): Parameter PROC has value gplot
MLOGIC(MKTITLE): Parameter DATA has value sasuser.houses
MLOGIC(MKTITLE): Ending execution.
MLOGIC(RUNPLOT): Ending execution.
```

MLOGICNEST システムオプション

マクロのネスト情報を MLOGIC 出力として SAS ログに表示するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO
LOGCONTROL

種類: システムオプション

デフォルト: NOMLOGICNEST

参照項目: “SAS ログ” (*SAS 言語リファレンス: 解説編* 9 章)

構文

MLOGICNEST | NOMLOGICNEST

必須引数

MLOGICNEST
マクロのネスト情報を MLOGIC 出力として SAS ログに表示します。

NOMLOGICNEST
マクロのネスト情報を MLOGIC 出力として SAS ログに表示しません。

詳細

MLOGICNEST オプションは、マクロのネスト情報が MLOGIC 出力として SAS ログに表示されるようにします。

MLOGICNEST オプションの設定は、現在実行中のマクロの出力には影響しません。

MLOGICNEST を設定しても、MLOGIC を設定したことにはなりません。ネスト情報を含む出力を SAS ログに表示するには、MLOGIC および MLOGICNEST の両システムオプションを設定する必要があります。

例: MLOGICNEST システムオプションの使用

最初の例では、MLOGIC オプションと MLOGICNEST オプションの両方を使用しています。

```
%macro outer;
%put THIS IS OUTER;
%inner;
%mend outer;
%macro inner;
%put THIS IS INNER;
%inrmost;
%mend inner;
%macro inrmost;
%put THIS IS INRMOST;
%mend;
options mlogic mlogicnest;
%outer
```

MLOGICNEST オプションを使用した場合、SAS ログに表示される MLOGIC 出力は次のようになります。

```
MLOGIC(OUTER): Beginning execution.
MLOGIC(OUTER): %PUT THIS IS OUTER
THIS IS OUTER
MLOGIC(OUTER.INNER): Beginning execution.
MLOGIC(OUTER.INNER): %PUT THIS IS INNER
THIS IS INNER
MLOGIC(OUTER.INNER.INRMOST): Beginning execution.
MLOGIC(OUTER.INNER.INRMOST): %PUT THIS IS INRMOST
THIS IS INRMOST
```

```
MLOGIC(OUTER.INNER.INRMOST): Ending execution.
MLOGIC(OUTER.INNER): Ending execution.
MLOGIC(OUTER): Ending execution.
```

2 番目の例では、NOMLOGICNEST オプションのみを使用しています。

```
 %macro outer;
%put THIS IS OUTER;
%inner;
%mend outer;
%macro inner;
%put THIS IS INNER;
%inrmost;
%mend inner;
%macro inrmost;
%put THIS IS INRMOST;
%mend;
options nomlogicnest;
%outer
```

NOMLOGICNEST オプションを使用した場合、SAS ログに表示される MLOGIC 出力は次のようになります。

```
MLOGIC(OUTER): Beginning execution.
MLOGIC(OUTER): %PUT THIS IS OUTER
THIS IS OUTER
MLOGIC(INNER): Beginning execution.
MLOGIC(INNER): %PUT THIS IS INNER
THIS IS INNER
MLOGIC(INRMOST): Beginning execution.
MLOGIC(INRMOST): %PUT THIS IS INRMOST
THIS IS INRMOST
MLOGIC(INRMOST): Ending execution.
MLOGIC(INNER): Ending execution.
MLOGIC(OUTER): Ending execution.
```

MPRINT システムオプション

マクロの実行により生成される SAS ステートメントをデバッグ用にトレースするかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO
LOGCONTROL

種類: システムオプション

デフォルト: NOMPRINT

参照項目: “MFILE システムオプション” (348 ページ)および“SAS ログ” (*SAS 言語リファレンス: 解説編* 9 章)

構文

MPRINT | NOMPRINT

必須引数

MPRINT
マクロの実行により生成される SAS ステートメントを表示します。マクロをデバッグする際には、SAS ステートメントを表示すると便利です。

NOMPRINT
マクロの実行により生成される SAS ステートメントを表示しません。

詳細

MPRINT オプションは、マクロの実行により生成されるテキストを表示します。各ステートメントの先頭には改行が付けられます。MPRINT 出力の各行には、プレフィックスとして MPRINT(*macro-name*):が出力されるため、各ステートメントを生成したマクロを特定できます。複数の空白で区切られているトークンは、単一の空白で区切られた状態で出力されます。

MPRINT 出力を外部ファイルに送るには、MPRINT オプションと MFILE オプションの両方を指定した上で、その外部ファイルにファイル参照名 MPRINT を割り当てます。詳細は“MFILE システムオプション” (348 ページ)を参照してください。

例

例 1: SAS ステートメントの生成をトレースする

次の例では、MPRINT オプションを使用することで、マクロ MKTITLE および RUNPLOT の実行時に生成される SAS ステートメントをトレースしています。

```
%macro mktitle(proc,data);
title "%upcase(&proc) of %upcase(&data)";
%mend mktitle;
%macro runplot(ds);
%if %sysprod(graph)=1 %then
%do;
%mktitle (gplot,&ds)
proc gplot data=&ds;
plot style*price
/ haxis=0 to 150000 by 50000;
run;
quit;
%end;
%else
%do;
%mktitle (plot,&ds)
proc plot data=&ds;
plot style*price;
run;
quit;
%end;
%mend runplot;
options mprint;
%runplot(sasuser.houses)
```

このプログラムを実行すると、次のような MPRINT 出力が SAS ログに表示されます。

```
MPRINT(MKTITLE): TITLE "GPLOT of SASUSER.HOUSES";
MPRINT(RUNPLOT): PROC GPLOT DATA=SASUSER.HOUSES;
MPRINT(RUNPLOT): PLOT STYLE*PRICE / HAXIS=0 TO 150000 BY 50000;
```

```
MPRINT(RUNPLOT): RUN;
MPRINT(RUNPLOT): QUIT;
```

例 2: 外部ファイルに MPRINT 出力を送る

先のプログラムにおけるマクロ呼び出しの前に次に示すステートメントを追加すると、SAS セッションの終了時に、MPRINT 出力がファイル DEBUGMAC に送られます。

```
options mfile mprint;
filename mprint 'debugmac';
```

MPRINTNEST システムオプション

マクロのネスト情報を MPRINT 出力として SAS ログに表示するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMPRINTNEST

構文

MPRINTNEST | NOMPRINTNEST

必須引数

MPRINTNEST
マクロのネスト情報を MPRINT 出力として SAS ログに表示します。

NOMPRINTNEST
マクロのネスト情報を MPRINT 出力として SAS ログに表示しません。

詳細

MLOGICNEST オプションは、マクロのネスト情報が MPRINT 出力として SAS ログに表示されるようにします。MPRINTNEST 出力は、外部ファイルに送られる MPRINT 出力に対しては影響を与えません。詳細については、MFILE システムオプションの説明を参照してください。

MPRINTNEST を設定しても、MPRINT を設定したことにはなりません。ネスト情報を含む出力を SAS ログに表示するには、MPRINT および MPRINTNEST の両システムオプションを設定する必要があります。

例: MPRINTNEST システムオプションの使用

次の例では、MPRINT および MPRINTNEST の両システムオプションを使用しています。

```
%macro outer;
data _null_;
%inner
run;
%mend outer;
%macro inner;
```

```
put %inrmost;
%mend inner;
%macro inrmost;
'This is the text of the PUT statement'
%mend inrmost;
options mprint mprintnest;
%outer
```

これらのステートメントを実行すると、次の出力が SAS ログに表示されます。

```
MPRINT(OUTER): data _null_;
MPRINT(OUTER.INNER): put
MPRINT(OUTER.INNER.INRMOST): 'This is the text of the PUT statement'
MPRINT(OUTER.INNER): ;
MPRINT(OUTER): run;
This is the text of the PUT statement
NOTE: DATA statement used (Total process time):
real time 0.10 seconds
cpu time 0.06 seconds
```

次の例では、NOMPRINTNEST オプションを使用しています。

```
%macro outer;
data _null_;
%inner
run;
%mend outer;
%macro inner;
put %inrmost;
%mend inner;
%macro inrmost;
'This is the text of the PUT statement'
%mend inrmost;
options nomprintnest;
%outer
```

これらのステートメントを実行すると、次の出力が SAS ログに表示されます。

```
MPRINT(OUTER): data _null_;
MPRINT(INNER): put
MPRINT(INRMOST): 'This is the text of the PUT statement'
MPRINT(INNER): ;
MPRINT(OUTER): run;
This is the text of the PUT statement
NOTE: DATA statement used (Total process time):
real time 0.00 seconds
cpu time 0.01 seconds
```

MRECALL システムオプション

前回検索時に見つからなかったメンバを見つけるために自動呼び出しライブラリを検索するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMRECALL

構文

MRECALL | NOMRECALL

必須引数

MRECALL
未定義のマクロ名の呼び出しが試みられるたびに、そのマクロ名を見つけるために自動呼び出しライブラリを検索します。未定義のマクロを見つけるために自動呼び出しライブラリを繰り返し検索するのは非効率的です。通常、このオプションは、自動呼び出しマクロを呼び出すプログラムの開発やデバッグを行う場合に使用します。

NOMRECALL
要求されたマクロ名を見つけるために自動呼び出しライブラリを 1 度だけ検索します。

詳細

MRECALL オプションは、主として次の場合に使用します。

- 自動呼び出しライブラリ内のマクロを必要とするシステムを開発する場合。
- 使用できないライブラリ内にあるマクロに対する自動呼び出しにより引き起こされたエラーから回復する場合。この場合、そのライブラリを利用可能にした後、同マクロを再度呼び出す際に MRECALL オプションを使用します。通常、MRECALL オプションは、自動呼び出しマクロの開発やデバッグを行う場合以外には使用しません。

MREPLACE システムオプション

既存のマクロを再定義できるかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: MREPLACE

構文

MREPLACE | NOMREPLACE

必須引数

MREPLACE
WORK ライブラリ内のカタログに保存されている既存のマクロ定義を再定義できるようにします。

NOMREPLACE
WORK ライブラリ内のカタログに保存されている既存のマクロ定義を再定義することを禁止します。

詳細

MREPLACE システムオプションを指定すると、同じ名前を持つ既存のマクロを上書きすることが可能になります。

NOMREPLACE システムオプションを指定すると、同じ名前を持つマクロがコンパイル済みであった場合でも、既存のマクロを上書きできなくなります。

MSTORED システムオプション

マクロ機能がコンパイル済みマクロを見つけるために特定のカタログを検索するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

デフォルト: NOMSTORED

構文

MSTORED | NOMSTORED

必須引数

MSTORED
コンパイル済みマクロを見つけるために、SASMSTORE=オプションにより参照されている SAS ライブラリを検索します。

NOMSTORED
コンパイル済みマクロの検索を行いません。

詳細

MSTORED オプションの設定にかかわらず、マクロ機能がマクロを検索する場合、現在の SAS セッションでコンパイルされたマクロを最初に検索します。MSTORED オプションが有効である場合、マクロ機能は、コンパイル済みマクロを含んでいるライブラリを次に検索します。MAUTOSOURCE オプションが有効である場合、マクロ機能は、自動呼び出しマクロライブラリを次に検索します。その後、マクロ機能は、SASHELP ライブラリ内の SASMACR カタログを検索します。

MSYMTABMAX=システムオプション

マクロ変数シンボルテーブルで利用可能なメモリの最大量を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

構文

MSYMTABMAX= *n* | *n*K | *n*M | *n*G | MAX

必須引数

n
利用可能な最大メモリ量をバイト単位で指定します。

***n*K**
利用可能な最大メモリ量をキロバイト単位で指定します。

***n*M**
利用可能な最大メモリ量をメガバイト単位で指定します。

***n*G**
利用可能な最大メモリ量をギガバイト単位で指定します。

MAX
メモリの最大量(65534)を指定します。

詳細

最大値に達した場合、それ以降に追加されるマクロ変数はディスクに書き出されます。

MSYMTABMAX=システムオプションで指定できる値は、ゼロからお使いの動作環境で表すことができる非負整数の最大値の範囲になります。デフォルト値はホストにより異なります。値としてゼロを指定すると、すべてのマクロシンボルテーブルはメモリではなくディスクに書き出されます。

MSYMTABMAX=の値は、システム性能に影響を与える可能性があります。このオプション値が小さすぎる場合に、アプリケーションが指定のメモリ限界に頻繁に到達すると、結果としてディスク入出力が増加します。逆に、このオプション値が大きすぎる場合に、アプリケーションが指定のメモリ限界に頻繁に到達すると、アプリケーションで利用可能なメモリ量が少なくなり、CPU の使用率が増加します。この値を実務ジョブ用に指定する前に、テストを実行して最適値を決定してください。

MVARSIZE=システムオプション

メモリに保存されるマクロ変数値の最大サイズを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

構文

MVARSIZE= *n* | *n*K | *n*M | *n*G | MAX

必須引数

n
利用可能な最大メモリ量をバイト単位で指定します。

***n*K**
利用可能な最大メモリ量をキロバイト単位で指定します。

***n*M**
利用可能な最大メモリ量をメガバイト単位で指定します。

***n*G**
利用可能な最大メモリ量をギガバイト単位で指定します。

MAX
メモリの最大量(65534)を指定します。

詳細

マクロ変数値に必要なメモリ量が MVARSIZE=値よりも大きい場合、その変数値はディスク上の一時カタログに書き出されます。マクロ変数名はメンバ名として使用され、すべてのメンバがタイプ MSYMTAB を持ちます。

MVARSIZE=システムオプションに指定できる値の範囲は、0～65534 です。値 0 を指定すると、すべてのマクロ変数値がディスクに書き出されます。

MVARSIZE=の値は、システム性能に影響を与える可能性があります。このオプション値が小さすぎる場合に、アプリケーションが限界値を超える大きさの変数を頻繁に作成すると、結果としてディスク入出力が増加します。この値を実務ジョブ用に指定する前に、テストを実行して最適値を決定してください。

注: MVARSIZE=オプションは、マクロ変数の値の最大長には影響しません。詳細は“マクロ変数” (21 ページ)を参照してください。

SASAUTOS=システムオプション

自動呼び出しライブラリの保存場所を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= ENVFILES
MACRO

種類: システムオプション

構文

SASAUTOS= *library-specification* |
(*library-specification-1* . . . , *library-specification-n*)

必須引数

library-specification
SAS マクロ定義を含んでいるライブラリメンバを含んでいるライブラリの保存場所を指定します。場所は、SAS ファイル参照名を指定するか、またはホスト固有の場所名を引用符で囲んで指定します。指定されたライブラリの各メンバは SAS マクロ定義を含んでいます。

(*library-specification-1 . . . , library-specification-n*)
SAS マクロ定義を含んでいるライブラリメンバを含んでいるライブラリの保存場所を 2 つ以上指定します。場所は、SAS ファイル参照名を指定するか、またはホスト固有の場所名を引用符で囲んで指定します。2 つ以上の自動呼び出しライブラリを指定する場合、場所の指定を丸かっこで囲み、それらをカンマまたは空白で区切る必要があります。

詳細

SAS システムが自動呼び出しマクロ定義を検索する場合、SASAUTOS オプションに指定された順番と同じ順番で各場所をオープンし検索を実施します。SAS システムが指定の場所をオープンできない場合、警告メッセージが生成され、NOMAUTOSOURCE システムオプションが有効になります。同じ SAS セッションで自動呼び出し機能を再度使用するには、MAUTOSOURCE オプションを再度指定する必要があります。

動作環境の情報
ソースライブラリを指定するには、ファイル参照名を使用するか、またはホスト固有の場所名を引用符で囲んで指定します。有効なライブラリ指定とその構文はホストにより異なります。ライブラリ指定の構文は、通常、お使いの動作環境のコマンドライン構文に一致しますが、句読点が追加または変更されている場合があります。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

z/OS 固有
library-specification を追加するには、APPEND システムオプションか INSERT システムオプションを使用します。詳細については、UNIX および z/OS 環境における APPEND システムオプションと INSERT システムオプションのドキュメントを参照してください。

SASMSTORE=システムオプション

コンパイル済みマクロのカタログを含んでいる(または含む予定の)SAS ライブラリのライブラリ参照名を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

構文

SASMSTORE=*libref*

必須引数

libref
コンパイル済みマクロのカタログを含んでいる(または含む予定の)SAS ライブラリのライブラリ参照名を指定します。このライブラリ参照名として、WORK は指定できません。

SERROR システムオプション

マクロ変数参照がマクロ変数に一致しない場合に、マクロプロセッサが警告メッセージを発行するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO

種類: システムオプション

別名: SERR | NOSERR

デフォルト: SERROR

構文

SERROR | NOSERROR

必須引数

SERROR
マクロプロセッサがマクロ参照を既存のマクロ変数に対応付けることができない場合に、警告メッセージを発行します。

NOSERROR
マクロプロセッサがマクロ参照を既存のマクロ変数に対応付けることができない場合に、警告メッセージを発行しません。

詳細

マクロ変数参照が置換されない場合、いくつかの理由が考えられます。たとえば、次に示す条件が 1 つ以上当てはまる場合、マクロ変数参照は置換されません。

- マクロ変数参照内で名前の綴りが誤っている場合。
- 変数が定義される前に参照されている場合。
- アンパサンド(&)で始まる文字列がプログラム内に含まれており、アンパサンドと文字列の間が空白で区切られていない場合。次に例を示します。

```
if x&y then do;
if buyer="Smith&Jones, Inc." then do;
```

アンパサンドを含んでいるテキスト文字列がプログラムに含まれてる場合、このプログラムの実行時に警告メッセージが発行されないようするには、NOSERROR オプションを指定します。

SYMBOLGEN システムオプション

デバッグ用に、マクロ変数参照の置換結果を SAS ログに出力するかどうかを指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

PROC OPTIONS GROUP= MACRO
LOGCONTROL

種類: システムオプション

別名: SGEN | NOSGEN

デフォルト: NOSYMBOLGEN

参照項目: “SAS ログ” (*SAS 言語リファレンス: 解説編* 9 章)

構文

SYMBOLGEN | NOSYMBOLGEN

必須引数

SYMBOLGEN
マクロ変数参照の置換結果を表示します。このオプションはデバッグを行う場合に使うと便利です。

NOSYMBOLGEN
マクロ変数参照の置換結果を表示しません。

詳細

SYMBOLGEN オプションは結果を次の形式で表示します。

```
SYMBOLGEN: Macro variable name resolves to value
```

SYMBOLGEN オプションは、二重のアンパサンド(**&&**)が単一のアンパサンド(**&**)に置換された場合にも置換結果を表示します。

例: マクロ変数の参照の置換をトレースする

次の例では、SYMBOLGEN オプションを使用して、マクロ MKTITLE および RUNPLOT の実行時に発生するマクロ変数参照の置換をトレースします。

```
%macro mktitle(proc,data);
title "%upcase(&proc) of %upcase(&data)";
%mend mktitle;
%macro runplot(ds);
%if %sysprod(graph)=1 %then
%do;
%mktitle (gplot,&ds)
proc gplot data=&ds;
plot style*price
/ haxis=0 to 150000 by 50000;
run;
quit;
%end;
%else
%do;
%mktitle (plot,&ds)
proc plot data=&ds;
plot style*price;
run;
quit;
%end;
%mend runplot;
%runplot(sasuser.houses)
```

このプログラムを実行すると、次の SYMBOLGEN 出力が SAS ログに表示されます。

```
SYMBOLGEN: Macro variable DS resolves to sasuser.houses
SYMBOLGEN: Macro variable PROC resolves to gplot
SYMBOLGEN: Macro variable DATA resolves to sasuser.houses
SYMBOLGEN: Macro variable DS resolves to sasuser.houses
```

SYSPARM=システムオプション

SAS プログラムに渡すことのできる文字列を指定します。

該当要素: 構成ファイル、OPTIONS ウィンドウ、OPTIONS ステートメント、SAS 起動時

種類: システムオプション

構文

SYSPARM='*character-string*'

必須引数

character-string
文字列を引用符で囲んで指定します。文字列の最大長は 200 です。

詳細

DATA ステップ内では SYSPARM()関数を使用することにより、指定した文字列にアクセスできます。また、マクロ変数参照&SYSPARM を使用すれば、SAS プログラムの任意の場所で同文字列にアクセスできます。

動作環境の情報
前述の構文は OPTIONS ステートメントに適用されます。コマンドラインまたは構成ファイルで呼び出す場合、構文はホストにより異なります。詳細については、お使いの動作環境向けの SAS ドキュメントを参照してください。

例: プログラムにユーザー ID を渡す

次の例では、SYSPARM オプションを使用して、ユーザー ID をプログラムに渡しています。

```
options sysparm='usr1';
data a;
length z $100;
if sysparm()='usr1' then z="&sysparm";
run;
```

3 部

付録

付録 1

マクロ機能の予約語

マクロ機能のワード規則

マクロ機能には次の規則が適用されます。

- 予約語は、マクロ名、マクロ変数名、マクロラベル名として使用できません。予約語には、マクロ機能により予約されている語と動作環境により予約されている語の両方が含まれます。マクロ名がマクロ機能の予約語である場合、マクロプロセッサは警告を発行します。そのマクロはコンパイルされないため、実行できません。マクロ機能が内部で使用するために予約している語の一覧については、“予約語” (369 ページ) を参照してください。
- マクロ言語要素には SYS で始まる名前を付けてはいけません。SAS システムは SYS で始まる名前を、SAS ソフトウェアが提供するマクロ言語要素の名前として予約しているためです。
- マクロ名の衝突を避けるために、マクロ変数には、SYS、AF、DMS で始まる名前を付けないでください。

予約語

マクロ機能における予約語の一覧を次の表に示します。

表 A1.1 *マクロ機能の予約語*

ABEND	END	LENGTH	QKUPCASE	SYSEVALF
ABORT	EVAL	LET	QSCAN	SYSEXEC
ACT	FILE	LIST	QSUBSTR	SYSFUNC
ACTIVATE	GLOBAL	LISTM	QSYSFUNC	SYSGET
BQUOTE	GO	LOCAL	QUOTE	SYSRPUT

BY	GOTO	MACRO	QUPCASE	THEN
CLEAR	IF	MEND	RESOLVE	TO
CLOSE	INC	PAUSE	RETURN	TSO
CMS	INCLUDE	NRSTR	RUN	UNQUOTE
COMANDR	INDEX	ON	SAVE	UNSTR
COPY	INFILE	OPEN	SCAN	UNTIL
DEACT	INPUT	PUT	STOP	UPCASE
DEL	KCMPRES	NRBQUOTE	STR	WHILE
DELETE	KINDEX	NRQUOTE	SYSCALL	WINDOW
DISPLAY	KLEFT	METASYM	SUBSTR	
DMIDSPLY	KLENGTH	QKCMPRES	SUPERQ	
DMISPLIT	KSCAN	QKLEFT	SYMDEL	
DO	KSUBSTR	QKSCAN	SYMEXIST	
EDIT	KTRIM	QKSUBSTR	SYMGLOBL	
ELSE	KUPCASE	QKTRIM	SYMLOCAL	

付録 2

SAS トークン

SAS トークン

SAS システムがプログラムを処理する場合、ワードスキャナ—と呼ばれるコンポーネントがプログラムを 1 文字ずつ読み取り、それらの文字をワードにグループ化します。このようなワードのことをトークンと呼びます。

トークンのリスト

SAS システムは、次に示す 4 つのタイプのトークンを認識します。

リテラル
: 一重または二重引用符で囲まれた 1 つ以上の文字です。リテラルの例としては次のものがあります。

表 A2.1 *リテラルの例*

'CARY'	"2008"
'Dr. Kemple-Long'	'<entry align="center">'

名前
: 文字または下線で始まる 1 つ以上の文字です。先頭以外には、文字、下線、数字を使用できます。

表 A2.2 *名前の例*

data	_test	linesleft
f25	univariate	otherwise

year_2008	descending

数
数値です。数値トークンには次のものが含まれます。

- 整数。整数とは、小数部や指数部を含まない数のことです。整数の例としては、1、72、5000 などが挙げられます。SAS 日付、時刻、日付および時刻などの定数(例: '24AUG2008'D)も整数であり、同様に 16 進定数(例: 0C4X)も整数です。
- 実数(浮動小数点数)。浮動小数点数とは、小数点や指数部を含んでいる数のことです。浮動小数点数の例としては、2.35、5.、2.3E1、5.4E− 1 などが挙げられます。

特殊文字
文字、数、下線以外の任意の文字です。次の文字は特殊文字です。

= + − % & ; ( ) #

トークンの最大長は、どのタイプであれ、32767 文字になります。トークンは、トークナイザが次のいずれかを検出した時点で終了します。

- 新しいトークンの開始。
- 名前トークンまたは数トークンの後の空白。
- リテラルトークンの場合、トークンを開始する同じタイプの引用符。これには例外があります。同じタイプの引用符が後に続いている引用符は、単一の引用符として解釈され、リテラルトークンの一部となります。たとえば、**'Mary''s'**という文字列の場合、4 番目の引用符が検出された時点で、このリテラルトークンは終了します。2 番目および 3 番目の引用符は、単一の引用符として解釈され、リテラルトークンの一部となります。

付録 3

%SYSFUNC 関数で使用する関数の構文

概要と構文

この付録では、%SYSFUNC 関数で使用できる関数(一部)の概要と構文を示します。

%SYSFUNC の関数と引数

%SYSFUNC 関数で使用できる関数(一部)の説明と構文を次の表に示します。この表は、%SYSFUNC 関数で使用できる関数の完全な一覧ではないことに注意してください。%SYSFUNC 関数で使用できない関数の一覧については、表 17.2 (274 ページ)を参照してください。

表 A3.1 *%SYSFUNC の関数と引数*

関数	説明と構文
ATTRC	SAS データセットの文字属性の値を返します。 **%SYSFUNC(ATTRC(*data-set-id,attr-name*))**
ATTRN	SAS データセットの数値属性の値を返します。 **%SYSFUNC(ATTRN(*data-set_id,attr-name*))**
CEXIST	SAS カタログまたは SAS カタログエントリが存在するかどうかを調べます。**%SYSFUNC(CEXIST(*entry <, U>*))**
CLOSE	SAS データセットをクローズします。**%SYSFUNC(CLOSE(*data-set-id*))**
CUROBS	現在のオブザベーションの番号を返します。 **%SYSFUNC(CUROBS(*data-set-id*))**

関数	説明と構文
DCLOSE	ディレクトリをクローズします。**%SYSFUNC(DCLOSE(*directory-id*))**
DINFO	ディレクトリに関する指定の情報項目を返します。**%SYSFUNC(DINFO(*directory-id,info-items*))**
DNUM	ディレクトリ内のメンバの数を返します。**%SYSFUNC(DNUM(*directory-id*))**
DOPEN	ディレクトリをオープンします。**%SYSFUNC(DOPEN(*fileref*))**
DOPTNAME	指定されたディレクトリ属性を返します。**%SYSFUNC(DOPTNAME(*directory-id,nval*))**
DOPTNUM	ディレクトリで利用可能な情報項目の数を返します。**%SYSFUNC(DOPTNUM(*directory-id*))**
DREAD	ディレクトリメンバの名前を返します。**%SYSFUNC(DREAD(*directory-id,nval*))**
DROPNOTE	SAS データセットや外部ファイルから注釈マーカーを削除します。**%SYSFUNC(DROPNOTE(*data-set-id*|*file-id,note-id*))**
DSNAME	データセット ID に関連付けられているデータセット名を返します。**%SYSFUNC(DSNAME(<*data-set-id*>))**
EXIST	SAS ライブラリメンバが存在するかどうかを調べます。**%SYSFUNC(EXIST(*member-name*<,*member-type*>))**
FAPPEND	外部ファイルの末尾にレコードを追加します。**%SYSFUNC(FAPPEND(*file-id*<,*cc*>))**
FCLOSE	外部ファイル、ディレクトリ、ディレクトリメンバをクローズします。**%SYSFUNC(FCLOSE(*file-id*))**
FCOL	ファイルデータバッファ(FDB)内の現在の列位置を返します。**%SYSFUNC(FCOL(*file-id*))**
FDELETE	外部ファイルを削除します。**%SYSFUNC(FDELETE(*fileref*))**
FETCH	SAS データセット内にある次の削除されていないオブザベーションをデータセットデータベクトル(DDV)に読み込みます。**%SYSFUNC(FETCH(*data-set-id*<,NOSET>))**
FETCHOBS	SAS データセット内にある指定されたオブザベーションを DDV に読み込みます。**%SYSFUNC(FETCHOBS(*data-set-id,obs-number*<,*options*>))**
FEXIST	ファイル参照名に関連付けられている外部ファイルが存在するかどうかを調べます。**%SYSFUNC(FEXIST(*fileref*))**

関数	説明と構文
FGET	FDB 内にあるデータをコピーします。**%SYSFUNC(FGET(*file-id*,*cval*<,*length*>))**
FILEEXIST	外部ファイルの物理名を使って同ファイルが存在するかどうかを調べます。**%SYSFUNC(FILEEXIST(*file-name*))**
FILENAME	外部ファイル、ディレクトリ、出力デバイスにファイル参照名を割り当てます。または割り当てを解除します。**%SYSFUNC(FILENAME(*fileref*,*file-name*<,*device*<,*host-options*<,*dir-ref*>>>))**
FILEREF	ファイル参照名が現在の SAS セッションで割り当て済みであるかどうかを調べます。**%SYSFUNC(FILEREF(*fileref*))**
FINFO	ファイルに関する指定の情報項目を返します。**%SYSFUNC(FINFO(*file-id*,*info-item*))**
FNOTE	読み取られた最後のレコードを特定します。**%SYSFUNC(FNOTE(*file-id*))**
FOPEN	外部ファイルをオープンします。**%SYSFUNC(FOPEN(*fileref*<,*open-mode*<,*record-length*<,*record-format*>>>))**
FOPTNAME	外部ファイルに関する情報項目の名前を返します。**%SYSFUNC(FOPTNAME(*file-id*,*nval*))**
FOPTNUM	外部ファイルで利用可能な情報項目の数を返します。**%SYSFUNC(FOPTNUM(*file-id*))**
FPOINT	読み取りポインタを、次に読み取るレコード上に配置します。**%SYSFUNC(FPOINT(*file-id*,*note-id*))**
FPOS	FDB 内の列ポインタの位置を設定します。**%SYSFUNC(FPOS(*file-id*,*nval*))**
FPUT	現在の列位置から始まるデータを、外部ファイルの FDB に移動します。**%SYSFUNC(FPUT(*file-id*,*cval*))**
FREAD	外部ファイル内のレコードを FDB に読み込みます。**%SYSFUNC(FREAD(*file-id*))**
FREWIND	ファイルポインタを先頭レコードに配置します。**%SYSFUNC(FREWIND(*file-id*))**
FRLEN	読み取った最終レコードのサイズ、または出力用にオープンしたファイルの現在のレコードサイズを返します。**%SYSFUNC(FRLEN(*file-id*))**
FSEP	FGET 関数で使用するトークン区切り文字を設定します。**%SYSFUNC(FSEP(*file-id*,*cval*))**

関数	説明と構文
FWRITE	レコードを外部ファイルに書き出します。 **%SYSFUNC(FWRITE(*file-id*<,*cc*>))**
GETOPTION	SAS システムオプションまたはグラフィックオプションの値を返します。 **%SYSFUNC(GETOPTION(*option-name*<,*reporting-options*<,...>>))**
GETVARC	SAS データセット変数の値を、DATA ステップ数値変数またはマクロ文字変数に割り当てます。**%SYSFUNC(GETVARC(*data-set-id*,*var-num*))**
GETVARN	SAS データセット変数の値を、DATA ステップ数値変数またマクロ数値変数に割り当てます。**%SYSFUNC(GETVARN(*data-set-id*,*var-num*))**
LIBNAME	ライブラリ参照名を SAS ライブラリに割り当てます。または割り当てを解除します。**%SYSFUNC(LIBNAME(*libref*<,*SAS-data-library*<,*engine*<,*options*>>>))**
LIBREF	ライブラリ参照名が割り当て済みであるかどうかを調べます。 **%SYSFUNC(LIBREF(*libref*))**
MOPEN	ディレクトリメンバファイルをオープンします。 **%SYSFUNC(MOPEN(*directory-id*,*member-name*<*open-mode*<,*record-length*<,*record-format*>>>))**
NOTE	SAS データセットの現在のオブザベーションのオブザベーション ID を返します。**%SYSFUNC(NOTE(*data-set-id*))**
OPEN	SAS データファイルをオープンします。**%SYSFUNC(OPEN(<*data-file-name*<,*mode*>>))**
PATHNAME	SAS ライブラリや外部ファイルの物理名を返します。 **%SYSFUNC(PATHNAME(*fileref*))**
POINT	NOTE 関数により特定されるオブザベーションを見つけます。 **%SYSFUNC(POINT(*data-set-id*,*note-id*))**
REWIND	データセットポインタを SAS データセットの先頭に配置します。 **%SYSFUNC(REWIND(*data-set-id*))**
SPEDIS	WHERE 句内にある不正なキーワードを正しいキーワードに変更するのに必要となる操作の数を返します。 **%SYSFUNC(SPEDIS(*query*,*keyword*))**
SYSGET	指定されたホストの環境変数の値を返します。 **%SYSFUNC(sysget(*host-variable*))**
SYSMSG	データセットや外部ファイルへのアクセスを試みた最後の関数により生成されたエラーメッセージや警告メッセージを返します。 **%SYSFUNC(SYSMSG())**

関数	説明と構文
SYSRC	直近に呼び出されたエントリのシステムエラー番号または終了ステータスを返します。**%SYSFUNC(SYSRC())**
VARFMT	データセット変数に割り当てられている出力形式を返します。**%SYSFUNC(VARFMT(*data-set-id*,*var-num*))**
VARINFMT	データセット変数に割り当てられている入力形式を返します。**%SYSFUNC(VARINFMT(*data-set-id*,*var-num*))**
VARLABEL	データセット変数に割り当てられているラベルを返します。**%SYSFUNC(VARLABEL(*data-set-id*,*var-num*))**
VARLEN	データセット変数の長さを返します。**%SYSFUNC(VARLEN(*data-set-id*,*var-num*))**
VARNAME	データセット変数の名前を返します。**%SYSFUNC(VARNAME(*data-set-id*,*var-num*))**
VARNUM	データセット変数の番号を返します。**%SYSFUNC(VARNUM(*data-set-id*,*var-name*))**
VARTYPE	データセット変数のデータ型を返します。**%SYSFUNC(VARTYPE(*data-set-id*,*var-num*))**

用語集

演算式
算術演算子とオペランドの並びから構成されるタイプのマクロ式。演算式は、実行時に数値を返します。

自動呼び出し機能
マクロを定義したソースステートメントを保存しておき、そのマクロ定義をプログラムに含めることなく、必要に応じて同マクロを呼び出せるようにする SAS 機能。

自動呼び出しマクロ
コンパイルされていないソースコードとテキストが自動呼び出しマクロライブラリ内に保存されているマクロ。コンパイル済みマクロとは異なり、自動呼び出しマクロは、それが初めて呼び出される際にコンパイルされます。

コマンドスタイルマクロ
%MACRO ステートメントの CMD オプションで定義されるマクロ。

定数テキスト
マクロの一部またはオープンコード内のマクロ変数の値として保存されている文字列。マクロプロセッサは、この文字列から、SAS ステートメント、ディスプレイマネージャコマンド、その他のマクロプログラムステートメントとして使用するテキストを生成します。定数テキストは、マクロテキストとも呼ばれます。

ダミーマクロ
マクロプロセッサによりコンパイルされるが保存されないマクロ。

グローバルマクロ変数
SAS プログラム内のグローバルスコープまたはローカルスコープの両方で参照されるマクロ変数。ただし、プログラム内に同じ名前のローカルマクロ変数が存在する場合は除きます。グローバルマクロ変数は、当該セッションまたはプログラムの終了まで存在します。

グローバルスコープ
SAS マクロプログラミングでは、グローバルマクロ変数を参照するための広いコンテキスト境界を指します。すなわち、現在の SAS セッションまたは SAS バッチプログラムを意味します。

入力スタック
SAS プログラムの入力から直近に読み取られた行、およびワードスキャナにより処理されるのを待っている、マクロプロセッサにより生成された任意のテキスト。

キーワードパラメータ
名前の後に等号が付くタイプのマクロパラメータ。複数のキーワードパラメータは任意の順番で指定できますが、その場合、任意の位置パラメータの後に指定する必要があります。

ローカルマクロ変数
それが定義されたマクロ内と、同マクロ内から呼び出されたマクロ内でのみ利用できるマクロ変数。ローカルマクロ変数は、それを作成したマクロが実行を停止すると存在しなくなります。

ローカルスコープ
SAS マクロプログラミングでは、ローカルマクロ変数を参照するための狭いコンテキスト境界を指します。すなわち、現在のマクロを意味します。

論理式
論理演算子とオペランドの並びから構成されるタイプのマクロ式。論理式は、実行時に true または false のいずれかの値を返します。

マクロ
一群のコンパイル済みのプログラムステートメントと保存済みテキストを含む SAS カタログエントリ。

マクロ呼び出し
SAS プログラム内で、コンパイル済みマクロプログラムをステートメントで呼び出すこと。マクロを呼び出すには、構文%<user-sup-val>macro-name</user-sup-val>; を使用します。

マクロのコンパイル
ユーザーが入力したステートメント内のマクロ定義を、マクロプロセッサが実行可能な形式に変換する処理。コンパイル済みのマクロを保存すると、それ以降の SAS プログラムやセッションで同マクロを使用できます。

マクロの実行
コンパイル済みのマクロプログラムステートメントにより与えられる命令に従って、テキストの生成、SAS ログへのメッセージ出力、入力の受け入れ、マクロ変数値の作成や変更、その他のアクティビティの実行などを行うこと。生成されるテキストは、SAS ステートメント、SAS コマンド、または別のマクロプログラムステートメントのいずれかになります。

マクロ式
実行時に値を返す記号の任意の有効な組み合わせ。マクロ式のタイプには、テキスト式、論理式、演算式の 3 つがあります。テキスト式は、テキスト、マクロ変数、マクロ関数、マクロ呼び出しの任意の組み合わせから構成され、解決(実行)されるとテキストを生成します。論理式は、論理演算子とオペランドから構成され、true または false のいずれかの値を返します。演算式は算術演算子とオペランドから構成され、数値を返します。

マクロ機能
SAS プログラムの拡張やカスタマイズに使用できる Base SAS ソフトウェアのコンポーネント。マクロ機能を使用すると、一般的なタスクを実行するのに入力する必要のあるテキスト量を削減できます。マクロ機能は、マクロプロセッサとマクロプログラミング言語から構成されます。

マクロ関数
マクロ機能により定義される関数。各マクロ関数は、1 つ以上の引数を処理することで結果を生成します。

マクロ起動
マクロ呼び出しに同じ。

マクロ言語
マクロプロセッサとの対話に使用されるプログラミング言語。

マクロパラメータ
%MACRO ステートメントの丸かっこ内に指定するローカルマクロ変数。マクロパラメータには、マクロの呼び出し時に、ユーザーが値を指定する必要があります。

マクロプロセッサ
マクロとマクロプログラムステートメントをコンパイルし実行する SAS ソフトウェアのコンポーネント。

マクロクォーティング
特殊文字やニーモニックをマクロ言語の一部としてではなくテキストとして解釈するようマクロプロセッサに命令する機能。

マクロ変数
SAS マクロプログラミング言語の一部である変数。マクロ変数の値は文字列であり、ユーザーが変更しない限り同じ値のままになります。マクロ変数はシンボリック変数とも呼ばれます。

マクロ変数参照
前にアンパサンドが付けられたマクロ変数名(&<user-sup-val>name</user-sup-val>)。マクロプロセッサは、マクロ変数参照を、指定されたマクロ変数の値で置き換えます。

モデルテキスト
定数テキストに同じ。

ネームスタイルマクロ
%MACRO ステートメントを使って指定され定義されるマクロ。

ヌル値
SAS マクロ言語では、ゼロ個の文字からなる値のこと。

オープンコード
マクロ定義の外側にある SAS プログラムの部分。

位置パラメータ
呼び出し時に、%MACRO ステートメントで(カンマ区切り文字を使って)指定されるタイプのマクロパラメータ。マクロ実行ステートメントでは、(同じくカンマ区切り文字を使って)対応する位置により定義されます。

クォーティング
特定の項目を、マクロ言語のシンボルとしてではなく、テキストとしてマクロプロセッサに読み取らせる処理。言い換えれば、クォーティングとは項目から意味を取り去り、その項目をテキストとして扱うことです。

クォーティング関数
その引数に関してクォーティングを実行するマクロ言語関数。

予約語
ソフトウェアアプリケーションの内部コンポーネントによる使用のために予約されているため、そのアプリケーションのユーザーによってはいかなるタイプのデータオブジェクトにも割り当てることができない名前。

戻り値
マクロ関数の実行結果となる文字列。

SAS コンパイル
SAS 言語の各種ステートメントを、ユーザーが入力した形式から SAS システムで利用可能な形式に変換する処理。

SAS 変数
SAS データセットまたは SAS データビュー内の列。各変数のデータ値は、すべてのオブザベーション(行)の単一の特性を表します。

スコープ
プログラミングにおいて、関連する値や式を取り囲むコンテキストのこと。SAS マクロプログラミングの場合、スコープはグローバルまたはローカルのいずれかになります。グローバルスコープとローカルスコープでは、マクロ変数に値を割り当てる方法や、マクロプロセッサによるマクロ変数参照の置換方法が異なります。

セッションコンパイル済みマクロ
マクロプロセッサによりコンパイルされ、WORK ライブラリ内の SAS カタログに保存されるマクロ。セッションコンパイル済みマクロは、現在の SAS セッション中にのみ存在します。コンパイル済みマクロとは異なり、セッションコンパイル済みマクロは、他の SAS セッションでは呼び出すことができません。

ステートメントスタイルマクロ
%MACRO ステートメントの STMT オプションで定義されるマクロ。

コンパイル済みマクロ
以前のセッションでコンパイルされ、永久ディレクトリ内に保存されたマクロプログラム。セッションコンパイル済みマクロとは異なり、コンパイル済みマクロは任意の SAS プログラムで呼び出すことができます。

文字列
SAS マクロ言語における、連続する複数の文字からなる任意のグループ。

シンボルテーブル
マクロプロセッサが特定のスコープ向けのすべてのマクロ変数およびマクロステートメントのラベルを格納する領域。

シンボリック置換
マクロ変数参照(&<user-sup-val>variable-name</user-sup-val>)をその値に置換する処理。

シンボリック変数
マクロ変数に同じ。

テキスト式
解決(実行)時にテキストを生成するタイプのマクロ式。テキスト式には、テキスト、マクロ変数、マクロ関数、マクロ呼び出しの任意の組み合わせを含めることができます。

トークン
SAS システムで入力を処理できるように、SAS 言語や SAS マクロ言語で入力を分割するのに使われる単位。トークン(またはワードとも呼ぶ)は、英語の単語のように見える項目(変数名など)や、それ以外の項目(算術演算子やセミコロンなど)を含んでいます。

トークナイザ
ワードスキャナの一部であり、入力をトークン(またはワード)に分割します。

アンクォーティング
クォーティングされた項目の意味を復元する処理。

ワード
トークンに同じ。

ワードスキャナ
SAS プログラム内のすべてのトークン(ワード)を検査し、それらのトークンを処理するために SAS システムの適切なコンポーネントに同トークンを移動する SAS コンポーネント。

キーワード

や

ら

わ