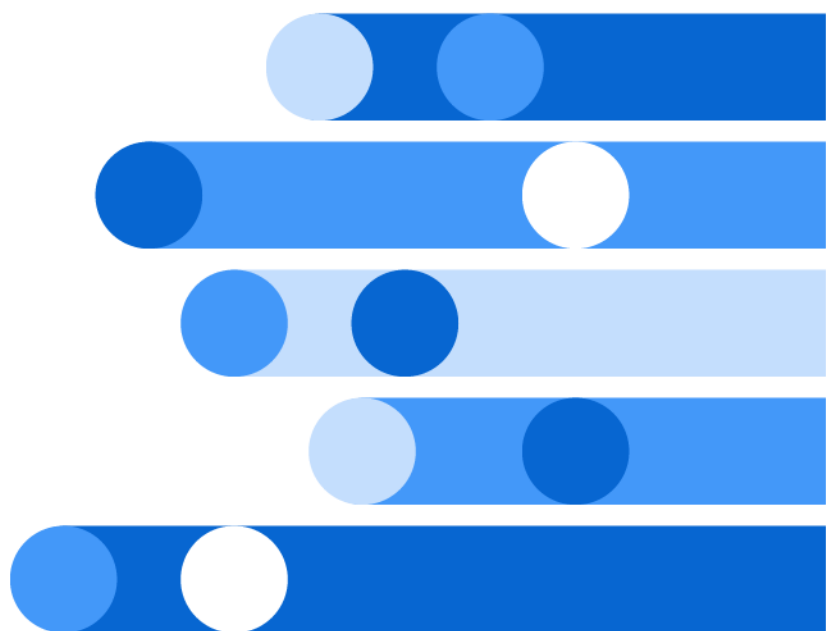




SAS[®] DATA ステップステートメント: リファレンス

2021.1.6 - 2026.01*



* このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを [SAS Help Center](#) で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。



SAS[®] ドキュメント
2026 年 3 月 20 日

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® DATA ステップステートメント: リファレンス*. Cary, NC: SAS Institute Inc.

SAS® DATA ステップステートメント: リファレンス

Copyright © 2020, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_001-P1:lestmtsref

目次

SAS 言語の構文規則	v
1 章 / SAS ステートメントの実行	1
定義	1
処理環境	8
CAS で DATA ステップを実行する理由	10
CAS DATA ステップでの実行がサポートされているステートメント	10
CAS DATA ステップでの実行がサポートされていないステートメント	15
SAS Viya プラットフォームで非推奨となったステートメント	16
2 章 / SAS DATA ステップステートメントのディクショナリ	19
カテゴリ別の DATA ステップステートメント	21
ディクショナリ	31
3 章 / SAS ステートメント環境変数のディクショナリ	399
ディクショナリ	399

SAS 言語の構文規則

SAS 言語の構文規則の概要

SAS 言語要素の構文のドキュメントでは、標準の規則が使用されています。これらの規則により、SAS 構文の構成要素を簡単に識別できます。規則は、次の項目に分類されます。

- 構文の構成要素
- 書体に関する規則
- 特殊文字
- SAS ライブラリや外部ファイルへの参照

構文の構成要素

ほとんどの言語要素の構文のコンポーネントには、キーワードと引数があります。キーワードのみ必要な言語要素もあります。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。複数の引数を含む構文で区切り記号を使用する場合と使用しない場合を説明するために、引数の構文の形式が複数示されています。

keyword

プログラムを記述するときに使用する SAS 言語要素の名前を指定します。キーワードはリテラルであり、通常、構文の先頭の単語です。CALL ルーチンでは、最初の 2 つの単語がキーワードです。

これらの例の SAS 構文では、キーワードには太字が使用されています。

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

この例では、CALL ルーチンの最初の 2 つの単語がキーワードです。

CALL RANBIN(*seed, n, p, x*)

一部の SAS ステートメントの構文は、引数のない 1 つのキーワードで構成されます。

DO;

... SAS code ...

END;

一部のシステムオプションでは、2 つあるキーワード値から 1 つを指定する必要があります。

DUPLEX | NODUPLEX

プロシジャステートメントによっては、ステートメント構文中に複数のキーワードが含まれます。

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <,
column-2, ...>)

引数

数値または文字の定数、変数、式のいずれかを指定します。引数は、キーワードまたはキーワードの後の等号記号の後に続きます。SAS では、言語要素を処理するために引数を使用されます。引数は必須の場合と、省略可能な場合があります。構文では、オプションの引数は山かっこ (<>) で囲みます。

この例では、*string* と *position* がキーワード CHAR に続きます。これらの引数は、CHAR 関数の必須引数です。

CHAR (*string, position*)

各引数には値があります。この例の SAS コードでは、引数 *string* の値は 'summer'、引数 *position* の値は 4 です。

```
x=char('summer', 4);
```

この例では、*string* および *substring* は必須引数ですが、*modifiers* と *startpos* はオプションです。

FIND(*string, substring* <,*modifiers*> <,*startpos*>

argument(s)

引数は必ず 1 つ必要であり、複数の引数が許可されます。引数の間はスペースで区切ります。カンマ(,)などの区切り記号は、引数間に必要ありません。

たとえば、MISSING ステートメントは、この形式で複数の引数を含みます。

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >

引数は必ず 1 つ必要であり、リテラル引数がこの引数に関連付けられます。リテラルと引数のペアは複数指定できます。リテラルと引数の間に区切り記号は必要ありません。省略記号(...)は、追加のリテラルと引数が許可されることを示します。

たとえば、BY ステートメントはこの引数を含みます。

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...>;

argument-1 <options> <argument-2 <options> ...>

引数は必ず 1 つ必要であり、1 つ以上のオプションがこの引数に関連付けられます。複数の引数と関連するオプションを指定できます。引数とオプションの間に区切り記号は必要ありません。省略記号(...)は、追加の引数と関連するオプションが許可されることを示します。

たとえば、FORMAT プロシジャの PICTURE ステートメントは、この形式で複数の引数を含みます。

```
PICTURE name <(format-options)>
<value-range-set-1 <(picture-1-options)>
<value-range-set-2 <(picture-2-options)> ...>;
```

argument-1=value-1 <argument-2=value-2 ...>

引数には値を割り当てる必要があり、複数の引数を指定できます。省略記号(...)は、追加の引数が許可されることを示します。引数間に区切り記号は必要ありません。

たとえば、LABEL ステートメントは、この形式で複数の引数を含みます。

```
LABEL variable-1=label-1 <variable-2=label-2 ...>;
```

argument-1 <, argument-2, ...>

引数は必ず 1 つ必要であり、カンマまたは別の区切り記号で区切って複数の引数を指定できます。省略記号(...)は、カンマで区切られた引数が続くことを示します。SAS ドキュメントでは両方の形式が使用されます。

次に、この形式で指定された複数の引数の例を示します。

```
AUTHPROVIDERDOMAIN (provider-1:domain-1 <, provider-2:domain-2, ...>
```

```
INTO :macro-variable-specification-1 <, :macro-variable-specification-2, ...>
```

注: 通常、SAS ドキュメントのサンプルコードは、小文字の固定幅フォントを使用して表記されます。コードの作成には、大文字も、小文字も、大文字と小文字の両方も使用できます。

書体に関する規則

SAS 構文の説明に使用されるスタイル規則には、大文字太字、大文字、斜体の規則も含まれます。

UPPERCASE BOLD

関数またはステートメントの名前など、SAS キーワードを示します。この例では、キーワード ERROR の表記には大文字太字が使用されています。

```
ERROR <message>;
```

UPPERCASE

リテラルである引数を表します。

この CMPMODEL=システムオプションの例では、BOTH、CATALOG、XML がリテラルです。

CMPMODEL=BOTH | CATALOG | XML |

italic

ユーザー指定の引数または値を示します。斜体表記の項目は、ユーザー指定値であり、次のいずれかを表します。

- 非リテラル引数。この LINK ステートメントの例では、引数 *label* はユーザー指定値のため、斜体で表示されます。

LINK *label*;

- 引数に割り当てられる非リテラル値。

この FORMAT ステートメントの例では、引数 DEFAULT に変数の *default-format* が割り当てられます。

FORMAT *variable(s)* <format> <DEFAULT = *default-format*>;

特殊文字

SAS 言語要素の構文には、次の特殊文字を使用できます。

=

等号記号は、システムオプションなどの一部の言語要素内のリテラルの値を示します。

この MAPS システムオプションの例では、等号により MAPS の値が設定されます。

MAPS=*location-of-maps*

<>

山かっこは省略可能な引数を示します。必須引数は山かっこで囲みません。

この CAT 関数の例では、少なくとも項目が 1 つが必要です。

CAT (*item-1* <, *item-2*, ...>)

|

縦棒は、値グループから 1 つの値を選択できることを示します。縦棒で区切られている値は相互に排他的です。

この CMPMODEL=システムオプションの例では、引数を 1 つのみ選択できます。

CMPMODEL=BOTH | CATALOG | XML

...

省略記号は、引数の繰り返しが可能なことを示します。引数と省略記号が山かっこで囲まれている場合、その引数はオプションです。繰り返される引数には、その引数の前や後ろに、区切り記号を入れる必要があります。

この CAT 関数の例では、複数の *item* 引数が許可され、カンマで区切る必要があります。

CAT (*item-1* <, *item-2*, ...>)

'value' or "value"

単一引用符や二重引用符付きの引数は、その値にも単一引用符または二重引用符を付ける必要があることを示します。

この FOOTNOTE ステートメントの例では、引数 *text* に引用符が付けられています。

FOOTNOTE <*n*> <*ods-format-options* 'text' | "text">;

;

セミコロンは、ステートメントまたは CALL ルーチンの終了を示します。

この例では、各ステートメントがセミコロンで終了しています。

```
data namegame;
  length color name $8;
  color = 'black';
  name = 'jack';
  game = trim(color) || name;
run;
```

SAS ライブラリと外部ファイルへの参照

SAS ステートメントおよびその他の言語要素の多くは、SAS ライブラリと外部ファイルを参照します。論理名(ライブラリ参照名またはファイル参照名)から参照を作成するのか、引用符付きの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、参照の作成に SAS ステートメント(LIBNAME または FILENAME)を使用するのか、動作環境のコントロール言語を使用するのかを選択します。SAS ライブラリと外部ファイルを参照する方法は複数あり、一部の方法は動作環境によって異なります。

SAS ドキュメント中の外部ファイルを使用する例では、*file-specification* という語句を斜体で使用しています。また、SAS ライブラリを使用する例には斜体フレーズ *SAS-library* を引用符で囲んで使用します。

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


SAS ステートメントの実行

定義	1
ステートメント	1
DATA ステップステートメント	2
コードブロックステートメント	5
処理環境	8
SAS Compute Server と CAS	8
CAS で DATA ステップを実行する理由	10
CAS DATA ステップでの実行がサポートされているステートメント	10
CAS DATA ステップでの実行がサポートされていないステートメント	15
SAS Viya プラットフォームで非推奨となったステートメント	16

定義

ステートメント

定義

SAS ステートメントとは、SAS キーワード、SAS 名、特殊文字、または演算子で構成される文字列です。SAS ステートメントは、特定の処理を実行するよう SAS に指示するほか、必要な情報を SAS に提供します。SAS ステートメントの最後には、必ずセミコロン(;)を追加します。SAS 言語のステートメントの種類の一部を次に示します。

- DATA ステップステートメント
- PROC ステップステートメント
- マクロステートメント

■ グローバルステートメント

このドキュメントは、SAS Viya プラットフォームでの DATA ステッププログラミングで使用する SAS ステートメントについて説明します。

これらステートメントの完全なリストについては、“[他の種類の SAS ステートメントのドキュメント](#)”を参照してください。

他の種類の SAS ステートメントのドキュメント

一部のステートメントは、他の SAS ドキュメントの関連する題材で説明されています。

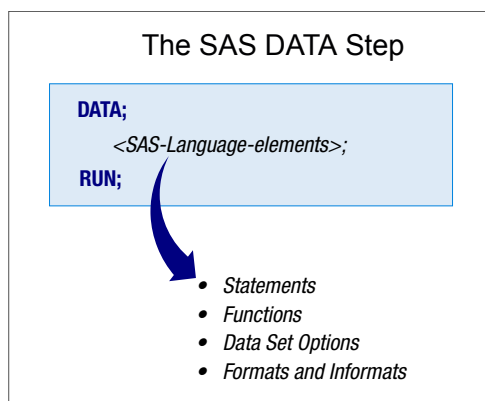
- [SAS グローバルステートメント: リファレンス](#)
- [Base SAS プロシジャガイド](#)
- [SAS DS2 Language Reference](#)
- [Application Messaging with SAS](#)
- [SAS Language Interfaces to Metadata](#)
- [SAS マクロ言語: リファレンス](#)
- [SAS Output Delivery System: ユーザーガイド](#)
- [SAS XML LIBNAME Engine: ユーザーガイド](#)
- [SAS/ACCESS for Relational Databases: リファレンス](#)
- [SAS/CONNECT User's Guide](#)
- [SAS/SHARE User's Guide](#)

DATA ステップステートメント

定義

SAS DATA ステップステートメントは、SAS DATA ステップ内で実行される SAS 言語要素の一種であり、SAS DATA ステッププログラミング言語の一部です。

SAS DATA ステップは、DATA ステートメントで始まり、RUN ステートメントで終わる SAS 言語要素のグループです。DATA ステートメントの後には、さらに多くの DATA ステップステートメント、[関数](#)、[出力形式](#)、および [入力形式](#)などの他のプログラミング言語要素が続きます。これらの言語要素は、データの読み込み、変換、および書き込みに使用されます。



DATA ステップ機能の概要

- データを読み込み、変換し、出力します。
- SAS ではシングルスレッドで実行されます。
- SAS Cloud Analytic Services (CAS)ではマルチスレッドで実行されます。CAS は、大規模な並列実行を可能にするクラウドベースのランタイム環境です。
- ファイルの終わりに達するまで、外部ファイルから自動的にレコードを読み込むループ機能を内蔵しています。

他の DATA ステップ言語要素のドキュメント

- [SAS 関数と CALL ルーチン: リファレンス](#)
- [SAS データセットオプション: リファレンス](#)
- [SAS 出力形式と入力形式: リファレンス](#)

実行ステートメントと宣言ステートメント

DATA ステップステートメントは、DATA ステップ内に現れ、SAS で実行される実行可能または宣言ステートメントです。この 2 種類のステートメントの違いは、ステートメントが有効になるタイミングが、コンパイル時かプログラム実行時かという点にあります。

DATA ステップ処理のフェーズについては、“[アクションのフロー](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

Declarative statements

宣言ステートメントは、SAS に情報を供給し、DATA ステップ処理の[コンパイルフェーズ](#)中に有効になります。宣言ステートメントは、しばしば“コンパイル時専用”のステートメントと呼ばれます。これらのステートメントは、出力に書き込まれる変数に関する情報をプログラムデータベクター(PDV)に提供し、変数の属性を決定します。

宣言ステートメントは、SAS 実行可能ステートメントが反復中にどの行を読み込むかを認識するのにも役立ちます。この変数情報がコンパイル時に決定される

と、属性が設定され、変更できなくなるため、これらのステートメントは非常に重要です。これが、コンパイル時専用ステートメントが条件付きで実行できない理由でもあります。これらのステートメントは、DATA ステップが実行される前のコンパイル時に有効になります。

例えば、WHERE ステートメントは DATA ステップ内で指定され、そのステートメントが有効であるように見えるかもしれませんが、実際にはコンパイル時に有効になる宣言ステートメントです。WHERE ステートメントは、SET、MERGE、MODIFY などの実行可能ステートメントが必要とする情報を SAS に提供するため、宣言的です。

Executable statements

実行ステートメントは、DATA ステップの各反復において、何らかのアクションを実施します。これらは、プログラムがすでにコンパイルされた後の[実行フェーズ](#)で有効になります。実行可能ステートメントは、プログラムのすべてのループとフローを制御します。

次の表に、SAS DATA ステップで使用可能な実行ステートメントと宣言ステートメントを示します。

表 1.1 DATA ステップの実行ステートメント

実行ステートメント		
ABORT	IF-THEN/ELSE	PUT, フォーマット
配列参照	INFILE	PUT, リスト
割り当て	INPUT	PUT, 名前付き
CALL	GOTO	PUT
CONTINUE	INPUT, 列	PUT, ODS
DELETE	INPUT, フォーマット	PUTLOG
DESCRIBE	INPUT, リスト	REDIRECT
DISPLAY	INPUT, 名前付き	REMOVE
DO	LEAVE	REPLACE
DO, 反復	LINK	RESETLINE
DO UNTIL	LIST	RETURN
DO WHILE	LOSTCARD	SELECT
ERROR	MERGE	SET
EXECUTE	MODIFY	STOP

実行ステートメント		
FILE	null	合計
FILE, ODS	OUTPUT	UPDATE
IF, サブセット化	PUT, 列	

表 1.2 DATA ステップの宣言ステートメント

宣言ステートメント		
ARRAY	DATALINES4	ラベル、ステートメント
ATTRIB	DROP	LENGTH
BY	END	RENAME
CARDS	FORMAT	RETAIN
CARDS4	INFORMAT	WHERE
DATA	KEEP	WINDOW
DATALINES	LABEL	n/a

DATA ステップステートメントは、関数カテゴリに分類することができます。カテゴリ別の DATA ステップステートメントの一覧については、“[カテゴリ別の DATA ステップステートメント](#)” (21 ページ)を参照してください。

コードブロックステートメント

定義

コードブロックはソースコードの語彙構造であり、1 つのステートメントとして実行するためにグループ化されます。コードブロックを形成する SAS ステートメントは、**ブロックステートメント**と呼ばれることもあります。コードブロックステートメントは通常、開始と終了の SAS ステートメントを指定することによって形成されます。次に、SAS のコードブロックを形成するステートメントの例を示します。

- **DO ... END**
- **DATA ... RUN;**
- **SELECT ... END**

さまざまな種類のコードブロックの実行は、*グループ処理*として知られています。SAS におけるこのタイプの処理の例としては、*DO グループ処理*、*RUN グループ処理*、*BY グループ処理*などがあります。

他の DATA ステップ言語要素のドキュメント

- [SAS 関数と CALL ルーチン: リファレンス](#)
- [SAS データセットオプション: リファレンス](#)
- [SAS 出力形式と入力形式: リファレンス](#)

実行ステートメントと宣言ステートメント

DATA ステップステートメントは、DATA ステップ内に現れ、SAS で実行される実行可能または宣言ステートメントです。これら 2 種類のステートメントの違いは、ステートメントが有効になるタイミングが、コンパイル時かプログラム実行時かという点にあります。

DATA ステップ処理のフェーズについては、“[アクションのフロー](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

Declarative statements

宣言ステートメントは、SAS に情報を供給し、DATA ステップ処理の[コンパイルフェーズ](#)中に有効になります。宣言ステートメントは、しばしば“コンパイル時専用”のステートメントと呼ばれます。これらのステートメントは、出力に書き込まれる変数に関する情報をプログラムデータベクター(PDV)に提供し、変数の属性を決定します。

宣言ステートメントは、SAS 実行可能ステートメントが反復中にどの行を読み込むかを認識するのにも役立ちます。したがって、この変数情報がコンパイル時に決定されると、属性が設定され、変更できなくなるため、これらのステートメントは非常に重要です。これが、コンパイル時専用ステートメントが条件付きで実行できない理由でもあります。これらのステートメントは、DATA ステップが実行される前のコンパイル時に有効になります。

たとえば、WHERE ステートメントは DATA ステップ内で指定され、そのステートメントが有効であるように見えるかもしれませんが、実際にはコンパイル時に有効になる宣言ステートメントです。WHERE ステートメントは、SET、MERGE、MODIFY などの実行可能ステートメントが必要とする情報を SAS に提供するため、宣言的です。

Executable statements

実行ステートメントは、DATA ステップの各反復において、何らかのアクションを実施します。これらは、プログラムがすでにコンパイルされた後の[実行フェーズ](#)で有効になります。実行可能ステートメントは、プログラムのすべてのループとフローを制御します。

次の表に、SAS DATA ステップで使用可能な実行ステートメントと宣言ステートメントを示します。

表 1.3 DATA ステップの実行ステートメント

実行ステートメント		
ABORT	IF-THEN/ELSE	PUT, フォーマット
配列参照	INFILE	PUT, リスト
割り当て	INPUT	PUT, 名前付き
CALL	GOTO	PUT
CONTINUE	INPUT, 列	PUT, ODS
DELETE	INPUT, フォーマット	PUTLOG
DESCRIBE	INPUT, リスト	REDIRECT
DISPLAY	INPUT, 名前付き	REMOVE
DO	LEAVE	REPLACE
DO, 反復	LINK	RESETLINE
DO UNTIL	LIST	RETURN
DO WHILE	LOSTCARD	SELECT
ERROR	MERGE	SET
EXECUTE	MODIFY	STOP
FILE	null	合計
FILE, ODS	OUTPUT	UPDATE
IF, サブセット化	PUT, 列	

表 1.4 DATA ステップの宣言ステートメント

宣言ステートメント		
ARRAY	DATALINES4	ラベル、ステートメント
ATTRIB	DROP	LENGTH
BY	END	RENAME

宣言ステートメント		
CARDS	FORMAT	RETAIN
CARDS4	INFORMAT	WHERE
DATA	KEEP	WINDOW
DATALINES	LABEL	n/a

DATA ステップステートメントは、関数カテゴリに分類することができます。カテゴリ別の DATA ステップステートメントの一覧については、[“カテゴリ別の DATA ステップステートメント” \(21 ページ\)](#)を参照してください。

処理環境

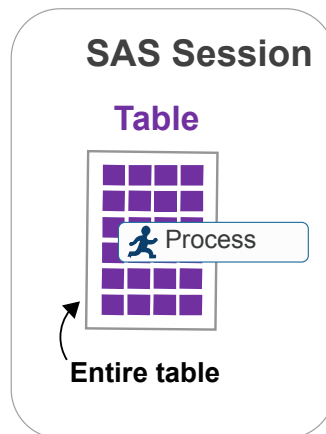
SAS Compute Server と CAS

SAS Viya では、DATA ステップステートメントは CAS または SAS のいずれかで実行されます。CAS は、マルチスレッドの並列 DATA ステップを実行できるクラウドベースのランタイム環境です。DATA ステップが CAS で実行される意味についての詳細は、[“分散 CAS 環境における分散データのマルチスレッド処理” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)を参照してください。

DATA ステップが SAS で実行される場合、SAS Compute Server 上の SAS クライアントセッションで実行されます。Compute Server を使用すると、クライアントは SAS 言語を使用して処理するジョブ形式で SAS プログラムとストアードプロシジャを送信できるようになります。

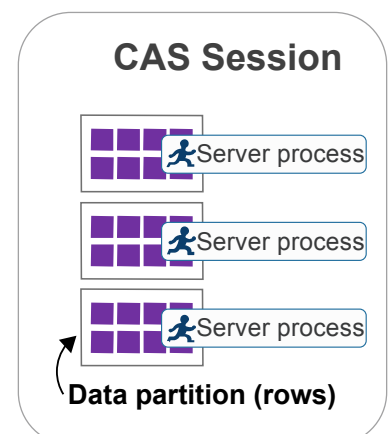
SAS Viya Platform Processing Environments

SAS Compute Server



The program always runs in a single thread on the whole SAS data set.

Cloud Analytic Services



The program runs in multiple threads in parallel on partitioned data.

CAS で DATA ステップを実行するための構文は、Compute Server で実行するための構文とほぼ同じです。つまり、CAS は同じ言語要素のほとんどをサポートし、多くの同じ機能を提供します。ただし、すべての DATA ステップ言語要素が CAS での処理でサポートされているわけではありません。

SAS 言語要素と DATA ステップの概念的な情報については、[SAS プログラマガイド: エッセンシャル](#)を参照してください。

CAS での DATA ステッププログラミングのサポートについては、次のドキュメントを参照してください。

- “CAS DATA ステップでの実行がサポートされているステートメント” (10 ページ).
- “CAS DATA ステップでの実行がサポートされていないステートメント” (15 ページ).
- “CAS でサポートされていない SAS CALL ルーチンと関数” (SAS 関数と CALL ルーチン: リファレンス)および “CAS でサポートされていない SAS CALL ルーチンと関数” (SAS 関数と CALL ルーチン: リファレンス)。
- CAS でサポートされている SAS 関数のリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (SAS 関数と CALL ルーチン: リファレンス)の“CAS”カテゴリーを参照してください。
- CAS でサポートされている SAS データセットオプションのリストについては、“カテゴリ別のデータセットオプション” (SAS データセットオプション: リファレンス)の“CAS”カテゴリーを参照してください。

- CAS でサポートされている SAS 出力形式のリストについては、“[カテゴリ別の出力形式](#)” ([SAS 出力形式と入力形式 リファレンス](#))の“CAS”カテゴリーを参照してください。
- CAS でサポートされている SAS 入力形式のリストについては、“[カテゴリ別の入力形式](#)” ([SAS 出力形式と入力形式 リファレンス](#))の“CAS”カテゴリーを参照してください。

CAS で DATA ステップを実行する理由

単一の実行スレッドで実行されるビッグデータは低速です。

DATA ステップは、CAS では、複数スレッドで並列実行できるため、より高速で実行されます。CAS では、複数のマシンで数百のスレッドを使用できます。DATA ステップを CAS で実行するようにプログラムすると、それらのスレッドがすべて使用可能になり、超並列処理でパフォーマンスを向上させることができます。並列処理では、複数のマシンで DATA ステップを開始し、処理ワークロードをこれらのマシンのスレッドに分割することで、ビッグデータの実行を高速化します。

CAS での DATA ステッププログラミングの詳細については、“[DATA ステップ処理モード](#)” ([SAS Cloud Analytic Services: DATA ステッププログラミング](#))を参照してください。

CAS DATA ステップでの実行がサポートされているステートメント

SAS Viya プラットフォーム上の SAS DATA ステップの機能強化により、DATA ステップ言語要素を CAS または SAS Compute Server で実行できるようになりました。DATA ステップが CAS で実行されると、処理が複数のスレッドに分散されるため、処理がより高速かつ効率的になります。

次の SAS ステートメントは、SAS Viya プラットフォーム上の CAS サーバーでの処理でサポートされています。

表 1.5 CAS DATA ステップでの実行がサポートされているステートメント

言語要素	説明
ABORT ステートメント	現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。

言語要素	説明
ARRAY ステートメント	配列の要素を定義します。
配列参照ステートメント	処理する配列要素を記述します。
“割り当て ステートメント”	式を評価し、その結果を変数に格納します。
“ATTRIB ステートメント”	1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。
“BY ステートメント”	DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。
“CALL ステートメント”	SAS CALL ルーチンを呼び出します。
“CONTINUE ステートメント”	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。
“DATA ステートメント”	DATA ステップを開始し、SAS データセット、ビュー、プログラムなどの出力の名前を提供します。
“DELETE ステートメント”	現在のオブザベーションの処理を中止します。
“DO ステートメント”	1 単位として実行するステートメントのグループを指定します。
“DO ステートメント: 反復”	インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
“DO UNTIL ステートメント”	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。

言語要素	説明
“DO WHILE ステートメント”	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
“DROP ステートメント”	出力 SAS データセットから変数を除外します。
“END ステートメント”	DO グループまたは SELECT グループの処理を終了します。
“ERROR ステートメント”	_ERROR_ を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
“FILE ステートメント”	PUT ステートメントの現在の出力ファイルを指定します。
“FORMAT ステートメント”	変数に出力形式を関連付けます。
“GOTO ステートメント”	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。
“IF ステートメント: サブセット化”	指定した式の条件に一致するオブザベーションの処理を続けます。
“IF-THEN/ELSE ステートメント”	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
“INFORMAT ステートメント”	変数に入力形式を関連付けます。
“KEEP ステートメント”	出力 SAS データセットに含める変数を指定します。
“LABEL ステートメント”	説明を示すラベルを変数に割り当てます。
“IF ステートメント: サブセット化”	他のステートメントから参照されるステートメントを識別します。

言語要素	説明
“LEAVE ステートメント”	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。
“LENGTH ステートメント”	文字変数や数値変数を格納する場合はバイト数を、VARCHAR 変数を格納する場合は文字数を指定します。
“LINK ステートメント”	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。
“LIST ステートメント”	現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。
“LOSTCARD ステートメント”	1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。
“MERGE ステートメント”	複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。
“OUTPUT ステートメント”	現在のオブザベーションを SAS データセットに書き込みます。
“PUT ステートメント”	SAS ログ、SAS 出力ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を書き込みます。
“PUT ステートメント: 列”	変数の値を出力行の指定した列に書き込みます。
“PUT ステートメント: フォーマット”	指定した出力形式で変数の値を出力行に書き込みます。
“PUT ステートメント: リスト”	変数値と指定した文字列を出力行に書き込みます。

言語要素	説明
“PUT ステートメント: 名前付き”	変数名と等号の後に変数値を書き込みます。
“PUTLOG ステートメント”	SAS ログにメッセージを書き込みます。
“RENAME ステートメント”	出力 SAS データセットの変数に新しい名前を指定します。
“RETAIN ステートメント”	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
“RETURN ステートメント”	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
“SELECT ステートメント”	複数のステートメントまたはステートメントグループからその1つを実行します。
“SET ステートメント”	1 つ以上の SAS データセットからオブザベーションを読み込みます。
“STOP ステートメント”	現在の DATA ステップの実行を中止します。
“合計 ステートメント”	式の計算結果を合計変数に加算します。
“WHERE ステートメント”	SAS データセットから特定の条件を満たすオブザベーションを選択します。

CAS での DATA ステップの実行方法の詳細については、[CAS での DATA ステップの役割](#)および [“DATA ステップ処理モード” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)を参照してください。

CAS DATA ステップでの実行がサポートされていないステートメント

次のステートメントは、CAS で実行されている DATA ステップでの処理ではサポートされていません。

表 1.6 CAS DATA ステップでの実行がサポートされていないステートメント

言語要素	説明
CARDS ステートメント	ステートメントの後にデータ行が続くことを指定します。
CARDS4 ステートメント	ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。
DATALINES ステートメント	ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。
DESCRIBE ステートメント	コンパイル済み DATA ステッププログラムまたは DATA ステップビューからソースコードを読み込みます。
EXECUTE ステートメント	コンパイル済み DATA ステッププログラムを実行します。
FILE ステートメント	PUT ステートメントの現在の出力ファイルを指定します。
INFILE ステートメント	INPUT ステートメントで読み込む外部ファイルを指定します。
INPUT ステートメント	入力データレコードでの値の位置を指定し、読み込んだ値を対応する SAS 変数に割り当てます。
INPUT ステートメント: 列	指定した列から入力値を読み込み、読み込んだ値を対応する SAS 変数に代入します。

言語要素	説明
INPUT ステートメント: フォーマット出力	指定した入力形式で入力値を読み込み、読み込んだ値を対応する SAS 変数に代入します。
INPUT ステートメント: リスト	入力データレコードをスキャンして入力値を取得し、その値を対応する SAS 変数に割り当てます。
INPUT ステートメント	条件が true である間、DO ループ内のステートメントを繰り返して実行します。
INPUT ステートメント: 名前付き	変数名と等号の後ろに続くデータ値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。
MODIFY ステートメント	既存のデータセットにあるオブザベーションの置き換え、削除、追加を実行します。ただし、オブザベーションのコピーは作成しません。
REMOVE ステートメント	SAS データセットからオブザベーションを削除します。
REPLACE ステートメント	同じ場所にあるオブザベーションを置換します。
UPDATE ステートメント	トランザクションを適用してマスターファイルを更新します。

SAS Viya プラットフォームで非推奨となったステートメント

次のグローバルステートメントと DATA ステップステートメントと機能は、SAS Viya プラットフォームではサポートされていません。

表 1.7 SAS Viya プラットフォームでサポートされていない SAS 9.4 ステートメントとステートメントオプション

言語要素
DISPLAY ステートメント
DM ステートメント(グローバル)
PUT ステートメントの_ODS_=オプション
FILE ステートメントの ODS=オプション
WINDOW ステートメント

SAS DATA ステップステートメントのディクショナリ

カテゴリ別の DATA ステップステートメント	21
ディクショナリ	31
ABORT ステートメント	31
ARRAY ステートメント	40
配列参照 ステートメント	46
割り当て ステートメント	51
ATTRIB ステートメント	52
BY ステートメント	57
CALL ステートメント	64
CARDS ステートメント	65
CARDS4 ステートメント	65
CATNAME ステートメント	66
CHECKPOINT EXECUTE_ALWAYS ステートメント	66
コメント ステートメント	66
CONTINUE ステートメント	66
DATA ステートメント	68
DATALES ステートメント	81
DATALES4 ステートメント	84
DELETE ステートメント	85
DESCRIBE ステートメント	86
DO ステートメント	88
DO ステートメント: 反復	90
DO UNTIL ステートメント	95
DO WHILE ステートメント	97
DROP ステートメント	98
END ステートメント	100
ENDSAS ステートメント	101
ERROR ステートメント	102
EXECUTE ステートメント	103
FILE ステートメント	104
FILENAME ステートメント	128
FILENAME ステートメント: Azure アクセスメソッド	128
FILENAME ステートメント: CATALOG アクセスメソッド	128

FILENAME ステートメント: CLIPBOARD アクセスメソッド	128
FILENAME ステートメント: DATAURL アクセスメソッド	129
FILENAME ステートメント: EMAIL(SMTP)アクセスメソッド	129
FILENAME ステートメント: FILESRVC アクセスメソッド	129
FILENAME ステートメント: FTP アクセスメソッド	129
FILENAME ステートメント: Hadoop アクセスメソッド	130
FILENAME ステートメント: S3 アクセスメソッド	130
FILENAME ステートメント: SOCKET アクセスメソッド	130
FILENAME ステートメント: URL アクセスメソッド	130
FILENAME ステートメント: WebDAV アクセスメソッド	130
FILENAME ステートメント: ZIP アクセスメソッド	131
FOOTNOTE ステートメント	131
FORMAT ステートメント	131
GOTO ステートメント	135
IF ステートメント: サブセット化	137
IF-THEN/ELSE ステートメント	140
%INCLUDE ステートメント	142
INFILE ステートメント	143
INFORMAT ステートメント	182
INPUT ステートメント	186
INPUT ステートメント: 列	205
INPUT ステートメント: フォーマット	210
INPUT ステートメント: リスト	215
INPUT ステートメント: 名前付き	223
KEEP ステートメント	228
LABEL ステートメント	230
Label: ステートメント	234
LEAVE ステートメント	236
LENGTH ステートメント	237
LIBNAME ステートメント	241
LIBNAME ステートメント: CVP エンジン	241
LIBNAME ステートメント: JMP エンジン	241
LIBNAME ステートメント: JSON エンジン	242
LIBNAME ステートメント: WebDAV サーバーアクセス	242
LINK ステートメント	242
LIST ステートメント	244
%LIST ステートメント	248
LOCK ステートメント	248
LOCKDOWN ステートメント	249
LOSTCARD ステートメント	249
MERGE ステートメント	252
MISSING ステートメント	261
MODIFY ステートメント	261
null ステートメント	284
OPTIONS ステートメント	284
OUTPUT ステートメント	285
PAGE ステートメント	288
PUT ステートメント	289
PUT ステートメント: 列	313
PUT ステートメント: フォーマット	316
PUT ステートメント: リスト	321
PUT ステートメント: 名前付き	326
PUTLOG ステートメント	329
REDIRECT ステートメント	335

REMOVE ステートメント	337
RENAME ステートメント	339
REPLACE ステートメント	341
RESETLINE ステートメント	344
RETAIN ステートメント	344
RETURN ステートメント	350
RUN ステートメント	351
%RUN ステートメント	352
SASFILE ステートメント	352
SELECT ステートメント	352
SET ステートメント	356
SKIP ステートメント	372
STOP ステートメント	372
合計 ステートメント	374
SYSECHO ステートメント	376
SYSTASK ステートメント	376
TITLE ステートメント	380
UPDATE ステートメント	380
WHERE ステートメント	389
X ステートメント	398

カテゴリ別の DATA ステップステートメント

実行ステートメントと宣言ステートメントの他に、SAS DATA ステップステートメントは次の関数カテゴリに分類できます。

表 2.1 DATA ステップステートメントのカテゴリ

ステートメントのカテゴリ	機能
アクション	■ 変数の作成や変更を行います。 ■ DATA ステップで処理する特定のオブザベーションのみを選択します。 ■ 入力データのエラーを調べます。 ■ 作成されるオブザベーションを操作します。 ステートメントのリストは、アクションを参照してください。
CAS	■ CAS サーバーで実行されるステートメントを指定します。 ステートメントのリストは、CASを参照してください。

ステートメントのカテゴリ	機能
制御	■ 特定のオブザベーションに対して、ステートメントをスキップします。 ■ ステートメントの実行順序を変更します。 ■ ある箇所から別の箇所にプログラムの制御を移動します。 ステートメントのリストは、制御を参照してください。
ファイル操作	■ データセットへの入力として使用するファイルを操作します。 ■ DATA ステップで書き込むファイルを操作します。 ステートメントのリストは、ファイル操作を参照してください。
In-Database 処理	■ In-database DATA ステップ処理でサポートされるステートメントを指定します。 ステートメントのリストは、In-Database 処理を参照してください。
情報	■ プログラムデータベクトルに関する追加情報を提供します。 ■ 作成する 1 つまたは複数のデータセットに関する追加情報を提供します。 ステートメントのリストは、情報を参照してください。

一部のステートメントは SAS 内でのみ実行され、一部のステートメントは SAS 内および CAS サーバー上で実行されます。ステートメントカテゴリに対して CAS を指定した場合、ステートメントは SAS 内および CAS サーバー上で実行されます。ステートメントカテゴリに対して CAS を指定していない場合、ステートメントは SAS 内でのみ実行されます。たとえば、DO ステートメントは SAS 内および CAS サーバー上で実行されるため、カテゴリとして CAS を指定します。ABORT ステートメントは SAS でしか実行されないため、CAS はカテゴリとして指定されません。

この表に、カテゴリ別の DATA ステップステートメントの一覧と簡単な説明を示します。

カテゴリ	言語要素	説明
CAS	ABORT ステートメント (p. 31)	現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。
	ARRAY ステートメント (p. 40)	配列の要素を定義します。

カテゴリ	言語要素	説明
	配列参照 ステートメント (p. 46)	処理する配列要素を記述します。
	割り当て ステートメント (p. 51)	式を評価し、その結果を変数に格納します。
	ATTRIB ステートメント (p. 52)	1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。
	BY ステートメント (p. 57)	DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。
	CALL ステートメント (p. 64)	SAS CALL ルーチンを呼び出します。
	CONTINUE ステートメント (p. 66)	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。
	DATA ステートメント (p. 68)	DATA ステップを開始し、SAS データセット、ビュー、プログラムなどの出力の名前を提供します。
	DELETE ステートメント (p. 85)	現在のオブザベーションの処理を中止します。
	DO ステートメント (p. 88)	1 単位として実行するステートメントのグループを指定します。
	DO ステートメント: 反復 (p. 90)	インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
	DO UNTIL ステートメント (p. 95)	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	DO WHILE ステートメント (p. 97)	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
	DROP ステートメント (p. 98)	出力 SAS データセットから変数を除外します。
	END ステートメント (p. 100)	DO グループまたは SELECT グループの処理を終了します。
	ERROR ステートメント (p. 102)	_ERROR を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
	FILE ステートメント (p. 104)	PUT ステートメントの現在の出力ファイルを指定します。
	FORMAT ステートメント (p. 131)	変数に出力形式を関連付けます。

カテゴリ	言語要素	説明
	GOTO ステートメント (p. 135)	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。
	IF ステートメント: サブセット化 (p. 137)	指定した式の条件に一致するオブザベーションの処理を継続します。
	IF-THEN/ELSE ステートメント (p. 140)	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
	INFORMAT ステートメント (p. 182)	変数に入力形式を関連付けます。
	KEEP ステートメント (p. 228)	出力 SAS データセットに含める変数を指定します。
	LABEL ステートメント (p. 230)	説明を示すラベルを変数に割り当てます。
	Label: ステートメント (p. 234)	他のステートメントから参照されるステートメントを識別します。
	LEAVE ステートメント (p. 236)	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。
	LENGTH ステートメント (p. 237)	文字変数や数値変数を格納する場合はバイト数を、VARCHAR 変数を格納する場合は文字数を指定します。
	LINK ステートメント (p. 242)	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。
	LIST ステートメント (p. 244)	現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。
	LOSTCARD ステートメント (p. 249)	1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。
	MERGE ステートメント (p. 252)	複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。
	OUTPUT ステートメント (p. 285)	現在のオブザベーションを SAS データセットに書き込みます。
	PUT ステートメント (p. 289)	SAS ログ、SAS 出力ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を書き込みます。
	PUT ステートメント: 列 (p. 313)	変数の値を出力行の指定した列に書き込みます。
	PUT ステートメント: フォーマット (p. 316)	指定した出力形式で変数の値を出力行に書き込みます。

カテゴリ	言語要素	説明
	PUT ステートメント: リスト (p. 321)	変数値と指定した文字列を出力行に書き込みます。
	PUT ステートメント: 名前付き (p. 326)	変数名と等号の後に変数値を書き込みます。
	PUTLOG ステートメント (p. 329)	SAS ログにメッセージを書き込みます。
	RENAME ステートメント (p. 339)	出力 SAS データセットの変数に新しい名前を指定します。
	RETAIN ステートメント (p. 344)	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
	RETURN ステートメント (p. 350)	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
	SELECT ステートメント (p. 352)	複数のステートメントまたはステートメントグループからその 1 つを実行します。
	SET ステートメント (p. 356)	1 つ以上の SAS データセットからオブザベーションを読み込みます。
	STOP ステートメント (p. 372)	現在の DATA ステップの実行を中止します。
	合計 ステートメント (p. 374)	式の計算結果を合計変数に加算します。
In-Database 処理	WHERE ステートメント (p. 389)	SAS データセットから特定の条件を満たすオブザベーションを選択します。
	ABORT ステートメント (p. 31)	現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。
	ARRAY ステートメント (p. 40)	配列の要素を定義します。
	配列参照 ステートメント (p. 46)	処理する配列要素を記述します。
	割り当て ステートメント (p. 51)	式を評価し、その結果を変数に格納します。
	ATTRIB ステートメント (p. 52)	1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。
	CALL ステートメント (p. 64)	SAS CALL ルーチンを呼び出します。
	CONTINUE ステートメント (p. 66)	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。

カテゴリ	言語要素	説明
	DATA ステートメント (p. 68)	DATA ステップを開始し、SAS データセット、ビュー、プログラムなどの出力の名前を提供します。
	DELETE ステートメント (p. 85)	現在のオブザベーションの処理を中止します。
	DO ステートメント (p. 88)	1 単位として実行するステートメントのグループを指定します。
	DO ステートメント: 反復 (p. 90)	インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
	DO UNTIL ステートメント (p. 95)	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	DO WHILE ステートメント (p. 97)	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
	DROP ステートメント (p. 98)	出力 SAS データセットから変数を除外します。
	END ステートメント (p. 100)	DO グループまたは SELECT グループの処理を終了します。
	ERROR ステートメント (p. 102)	_ERROR_ を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
	FORMAT ステートメント (p. 131)	変数に出力形式を関連付けます。
	GOTO ステートメント (p. 135)	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。
	IF ステートメント: サブセット化 (p. 137)	指定した式の条件に一致するオブザベーションの処理を継続します。
	IF-THEN/ELSE ステートメント (p. 140)	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
	KEEP ステートメント (p. 228)	出力 SAS データセットに含める変数を指定します。
	LABEL ステートメント (p. 230)	説明を示すラベルを変数に割り当てます。
	Label: ステートメント (p. 234)	他のステートメントから参照されるステートメントを識別します。
	LEAVE ステートメント (p. 236)	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。

カテゴリ	言語要素	説明
	LENGTH ステートメント (p. 237)	文字変数や数値変数を格納する場合はバイト数を、VARCHAR 変数を格納する場合は文字数を指定します。
	LINK ステートメント (p. 242)	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。
	OUTPUT ステートメント (p. 285)	現在のオブザベーションを SAS データセットに書き込みます。
	PUTLOG ステートメント (p. 329)	SAS ログにメッセージを書き込みます。
	RETAIN ステートメント (p. 344)	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
	RETURN ステートメント (p. 350)	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
	SELECT ステートメント (p. 352)	複数のステートメントまたはステートメントグループからその 1 つを実行します。
	SET ステートメント (p. 356)	1 つ以上の SAS データセットからオブザベーションを読み込みます。
	STOP ステートメント (p. 372)	現在の DATA ステップの実行を中止します。
	合計 ステートメント (p. 374)	式の計算結果を合計変数に加算します。
	WHERE ステートメント (p. 389)	SAS データセットから特定の条件を満たすオブザベーションを選択します。
アクション	ABORT ステートメント (p. 31)	現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。
	割り当て ステートメント (p. 51)	式を評価し、その結果を変数に格納します。
	CALL ステートメント (p. 64)	SAS CALL ルーチンを呼び出します。
	DELETE ステートメント (p. 85)	現在のオブザベーションの処理を中止します。
	DESCRIBE ステートメント (p. 86)	コンパイル済み DATA ステッププログラムまたは DATA ステップビューからソースコードを読み込みます。
	ERROR ステートメント (p. 102)	_ERROR を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。
	EXECUTE ステートメント (p. 103)	コンパイル済み DATA ステッププログラムを実行します。

カテゴリ	言語要素	説明
	IF ステートメント: サブセット化 (p. 137)	指定した式の条件に一致するオブザベーションの処理を継続します。
	LIST ステートメント (p. 244)	現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。
	LOSTCARD ステートメント (p. 249)	1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。
	OUTPUT ステートメント (p. 285)	現在のオブザベーションを SAS データセットに書き込みます。
	PUTLOG ステートメント (p. 329)	SAS ログにメッセージを書き込みます。
	REDIRECT ステートメント (p. 335)	ストアドプログラムを実行するときに、別の入力 SAS データセットまたは出力 SAS データセットを参照します。
	REMOVE ステートメント (p. 337)	SAS データセットからオブザベーションを削除します。
	REPLACE ステートメント (p. 341)	同じ場所にあるオブザベーションを置換します。
	STOP ステートメント (p. 372)	現在の DATA ステップの実行を中止します。
	合計 ステートメント (p. 374)	式の計算結果を合計変数に加算します。
	WHERE ステートメント (p. 389)	SAS データセットから特定の条件を満たすオブザベーションを選択します。
情報	ARRAY ステートメント (p. 40)	配列の要素を定義します。
	配列参照 ステートメント (p. 46)	処理する配列要素を記述します。
	ATTRIB ステートメント (p. 52)	1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。
	DROP ステートメント (p. 98)	出力 SAS データセットから変数を除外します。
	FORMAT ステートメント (p. 131)	変数に出力形式を関連付けます。
	INFORMAT ステートメント (p. 182)	変数に入力形式を関連付けます。
	KEEP ステートメント (p. 228)	出力 SAS データセットに含める変数を指定します。

カテゴリ	言語要素	説明
	LABEL ステートメント (p. 230)	説明を示すラベルを変数に割り当てます。
	LENGTH ステートメント (p. 237)	文字変数や数値変数を格納する場合はバイト数を、VARCHAR 変数を格納する場合は文字数を指定します。
	RENAME ステートメント (p. 339)	出力 SAS データセットの変数に新しい名前を指定します。
	RETAIN ステートメント (p. 344)	DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。
	CONTINUE ステートメント (p. 66)	DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。
	DO ステートメント (p. 88)	1 単位として実行するステートメントのグループを指定します。
	DO ステートメント: 反復 (p. 90)	インデックス変数の値に基づいて、DO ステートメントと END ステートメントにはさまれている各ステートメントを実行します。
	DO UNTIL ステートメント (p. 95)	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
	DO WHILE ステートメント (p. 97)	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
	END ステートメント (p. 100)	DO グループまたは SELECT グループの処理を終了します。
制御	GOTO ステートメント (p. 135)	指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。
	IF-THEN/ELSE ステートメント (p. 140)	特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。
	Label: ステートメント (p. 234)	他のステートメントから参照されるステートメントを識別します。
	LEAVE ステートメント (p. 236)	現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。
	LINK ステートメント (p. 242)	指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。
	RETURN ステートメント (p. 350)	DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。
	SELECT ステートメント (p. 352)	複数のステートメントまたはステートメントグループからその 1 つを実行します。

カテゴリ	言語要素	説明
ファイル操作	BY ステートメント (p. 57)	DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。
	CARDS ステートメント (p. 65)	ステートメントの後にデータ行が続くことを指定します。
	CARDS4 ステートメント (p. 65)	ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。
	DATA ステートメント (p. 68)	DATA ステップを開始し、SAS データセット、ビュー、プログラムなどの出力の名前を提供します。
	DATALINES ステートメント (p. 81)	ステートメントの後にデータ行が続くことを指定します。
	DATALINES4 ステートメント (p. 84)	ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。
	FILE ステートメント (p. 104)	PUT ステートメントの現在の出力ファイルを指定します。
	INFILE ステートメント (p. 143)	INPUT ステートメントで読み込む外部ファイルを指定します。
	INPUT ステートメント (p. 186)	入力データレコードでの値の位置を指定し、読み込んだ値を対応する SAS 変数に割り当てます。
	INPUT ステートメント: 列 (p. 205)	指定した列から入力値を読み込み、読み込んだ値を対応する SAS 変数に代入します。
	INPUT ステートメント: フォーマット (p. 210)	指定した形式で入力値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。
	INPUT ステートメント: リスト (p. 215)	入力データレコードの入力値を走査し、対応する SAS 変数に割り当てます。
	INPUT ステートメント: 名前付き (p. 223)	変数名と等号の後ろに続くデータ値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。
	MERGE ステートメント (p. 252)	複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。
	MODIFY ステートメント (p. 261)	既存のデータセットにあるオブザベーションの置き換え、削除、追加を実行します。ただし、オブザベーションのコピーは作成しません。
	PUT ステートメント (p. 289)	SAS ログ、SAS 出力ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を書き込みます。
	PUT ステートメント: 列 (p. 313)	変数の値を出力行の指定した列に書き込みます。
	PUT ステートメント: フォーマット (p. 316)	指定した出力形式で変数の値を出力行に書き込みます。

カテゴリ	言語要素	説明
	PUT ステートメント: リスト (p. 321)	変数値と指定した文字列を出力行に書き込みます。
	PUT ステートメント: 名前付き (p. 326)	変数名と等号の後に変数値を書き込みます。
	SET ステートメント (p. 356)	1 つ以上の SAS データセットからオブザベーションを読み込みます。
	UPDATE ステートメント (p. 380)	トランザクションを適用してファイルを更新します。

ディクショナリ

ABORT ステートメント

現在の DATA ステップ、SAS ジョブまたは SAS セッションの実行を中止します。

該当要素:	DATA ステップ
カテゴリ:	アクション CAS In-Database 処理
種類:	実行
制限事項:	In-database DATA ステップの場合、ABORT ステートメントは引数を受け入れません。詳細については、“ DATA Step Acceleration ” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。
Linux 固有:	n の値
参照項目:	“ SAS Compute Server と CAS ” (8 ページ) “ 処理環境 ” (8 ページ)

構文

形式 1: SAS Compute Server、バッチモード、または非対話型モードで使用する ABORT ステートメント

ABORT <[ABEND](#) | [CANCEL](#) <[FILE](#)> | [EXIT](#)< n > | [RETURN](#)< n > | [NOLIST](#)>;

形式 2: CAS サーバー上で使用できされる ABORT ステートメント

ABORT;**引数なし**

ABORT ステートメントの形式 1 をオプションなしで使用すると、次のアクションが発生します。

- バッチモードおよび非対話型モード
 - 現在の DATA ステップの処理を終了し、SAS ログにエラーメッセージを書き込みます。ABORT ステートメントを実行するタイミングによって、データセットの一部または全部のオブザベーションが含まれない場合があります。
 - OBS=システムオプションを 0 に設定します。
 - SAS ジョブの残りの部分に関して、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などの限られた処理を続けます。
 - 後続の DATA ステップおよび PROC ステップには、オブザベーションを含まない出力データセットを作成します。
- SAS Compute Server
 - 現在の DATA ステップを終了し、SAS ログにエラーメッセージを書き込みます。ABORT ステートメントを実行するタイミングによって、データセットの一部または全部のオブザベーションが含まれない場合があります。
 - OBS=システムオプションを 0 に設定します。
 - SAS ジョブの残りの部分に関して、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などの限られた処理を続けます。
 - 後続の DATA ステップおよび PROC ステップには、オブザベーションを含まない出力データセットを作成します。
 - 計算サーバーの jobConditionCode を異常なエラーコードに設定します。
 - 計算サーバージョブとセッションの状態を'エラー'に設定します。
- 対話型ラインモード
 - 現在処理中の DATA ステップを終了します。後続の DATA ステップまたは PROC ステートメントは、通常と同様に処理されます。

ABORT ステートメントの形式 2 は、オプションなしで指定された場合、CAS での処理でサポートされます。オプションを指定せずに ABORT ステートメントを指定すると、次のアクションが発生します。

- CAS サーバーセッション
 - CAS で実行中の現在の DATA ステップの処理を停止します。
 - ABORT ステートメントの実行前に処理された行を含む CAS テーブルを作成します。
 - ABORT ステートメントによる DATA ステップの実行を終了を示すメッセージを SAS ログに書き込みます。
 - ABORT ステートメントの後続の DATA ステップおよび PROC ステートメントの処理を続けます。

引数

ABEND

現在実行中の SAS ジョブまたは SAS セッションを異常終了させます。アクションは実行モードによって異なります。

- バッチモードおよび非対話型モード
 - 処理を即座に停止します。
 - ABORT ステートメントの ABEND オプションにより実行が終了させられたことを示すエラーメッセージを SAS ログに送信します。
 - 後続のステートメントや構文チェックを実行しません。
 - 動作環境に制御を戻します。それ以降の動作は、動作環境と各サイトでの異常終了時のジョブの処理方法によって異なります。
- SAS Compute Server セッション
 - SAS Compute Server セッションが終了します。
 - 異常なエラーコードは“sessionConditionCode”に設定されます。
 - SAS Compute Server セッションの状態は“キャンセル”に設定されます。

CANCEL <FILE> <n>

サブミットされたステートメントの実行をキャンセルします。アクションは実行モードによって異なります。

- バッチモードおよび非対話型モード
 - SAS プログラム全体を終了させます。
 - SAS ログにエラーメッセージを書き込みます。
- SAS Compute Server セッション
 - 現在サブミットされているプログラムのみをクリアします。
 - その後の他のサブミットコールには影響しません。
 - SAS ログにエラーメッセージを書き込みます。
- SAS IntrNet Application Server
 - 要求ごとに別々の実行を生成し、対応するリクエストコードをサブミットします。要求コード内に CANCEL 引数が含まれている場合、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 セッションは終了しません。

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、ABORT ステートメントを実行すると、その autoexec ファイルまたは%INCLUDE ファイルの内容のみがクリアされます。その他のサブミットされたソースステートメントは、その 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 を使用してサブミットできません。

注 %INCLUDE ファイル内で ABORT CANCEL FILE オプションを実行すると、開いているマクロがすべて閉じられ、ソースコードの次の行から実行が再開されます。

EXIT

これにより、SAS Compute Server セッションで次のアクションが発生します。

- 残りの SAS Compute Server セッションは実行されません。
- NOTE のメッセージが表示され、SAS Compute Server セッションが終了します。
- SAS Compute Server ジョブの 'jobConditionCode' は 0 に設定されます。
- SAS Compute Server セッションの状態は“完了”に設定されます。

<n>

SAS Compute Server セッションで EXIT 引数とともに使用する場合に、条件コードを指定できる整数値です。n を指定すると、次のアクションが発生します。

- 残りの SAS Compute Server セッションは実行されません。
- ERROR のメッセージが表示され、SAS Compute Server セッションが終了します。
- SAS Compute Server セッションの SYSCC 自動マクロ変数は n に設定されます。
- SAS Compute Server ジョブの 'jobConditionCode' は n に設定されます。
- SAS Compute Server セッションの状態は“エラー”に設定されます。

範囲 4000<n<20000、および n!= 9999

制限事項 ABORT EXIT <n> は、SAS Compute Server でのみサポートされています。SAS Compute Server 以外の実行モードで ABORT EXIT n を実行すると、次のエラーが発生します。

ERROR: ABORT EXIT is supported only in the SAS Compute Server.

SAS 用に予約されている既存のコードとの衝突を避けるため、n の値は次の範囲に制限されます。4000< n< 20000 および n !=9999

注 現在の SAS Compute Server セッションに設定されている終了コードは、以前のジョブで設定されている以前の SYSCC コードをオーバーライドします。

参照項目: “%ABORT マクロステートメント” (SAS マクロ言語: リファレンス)

RETURN

現在実行中の SAS ジョブまたはセッションをすぐに通常終了します。結果は動作モードによって異なります。

- バッチモードおよび非対話型モード:
 - 処理を即座に停止します。
 - ABORT ステートメントの RETURN オプションにより実行が終了させられたことを示すエラーメッセージを SAS ログに送信します。

- 後続のステートメントや構文チェックを実行しません。
- エラーを示す条件コードが表示され、動作環境に制御を戻します。
- SAS Compute Server セッション
- SAS Compute Server セッションが終了します。
- 異常なエラーコードは“sessionConditionCode”に設定されます。
- SAS Compute Server セッションの状態は“キャンセル”に設定されます。

n

条件コードの指定を可能にする整数値です。CANCEL 引数とともに *n* を使用すると、値 *n* が SYSINFO 自動マクロ変数に配置されます。CANCEL 引数と一緒に使用しない場合、*n* の値は、SAS によって終了コードとして返されるエラーコードです。*n* の値は 0 - 255 の範囲で指定できます。通常、リターンコード 0 は、プログラムがエラーなしで実行されたことを示します。正值であれば、数値が大きいほど重大なエラーであることを表します。リターンコード 0～6 は、SAS による使用のために予約されています。

NOLIST

すべての変数を SAS ログへ出力しないようにします。

要件 ABORT ステートメントでは NOLIST を最後のオプションとして指定する必要があります。

詳細

一般情報

ABORT ステートメントは、現在の DATA ステップの処理を終了させます。この後の展開は、以下の情報によって決まります。

- SAS ステートメントをサブミットするために使用した方法
- ABORT ステートメントで使った引数
- ユーザーの動作環境

通常、ABORT ステートメントは IF-THEN ステートメントや SELECT ステートメントの句の中で使用され、エラー条件が発生した場合に処理を終了させます。

DATA ステップで ABORT ステートメントを実行する場合、このステップで作成された DATA ステップで同じ名前の既存のデータセットが置き換えられることはありません。

注: システムオプション ERRORABEND が適用されている場合、SAS では ABORT ステートメントで生成されるリターンコードは無視されます。

動作環境の情報: ABEND オプションと RETURN オプションには 1 つだけ違いがあります。ABEND オプションを使用した場合、それ以降の処理は、お使いの動作環境と各サイトでの異常終了ジョブの処理方法によって異なります。RETURN オプションを使用した場合、エラーを示す条件コードが返されます。

比較

- バッチモードまたは非対話型モードでは、ABORT ステートメントと STOP ステートメントの動作が異なります。ABORT ステートメントは処理を停止し、自動変数_ERROR_の値を 1 に設定します。STOP ステートメントは現在実行中の DATA ステップの処理を停止し、次のステップから処理を続けます。

例

例 1: SAS の DATA ステップ処理を終了する

この例では、**IF-THEN/ELSE** ステートメント内で **ABORT** ステートメントを使用しており、ゼロ除算の条件が発生するデータ値が検出された場合に DATA ステップ処理を停止します。DATA ステップは、SAS Compute Server 上の SAS で実行されます。

入力データは 4 行で構成されますが、DATA ステップは 2 行のみを含む出力データセットを作成します。これは、行 3 でゼロ除算エラーが発生すると DATA ステップが終了しますが、部分的な出力データセットが作成されるためです。

```
data test;
  input mass volume;
  density=mass/volume;
  if volume=0 then abort;
datalines;
100 20
200 20
300 0
400 20
;
proc print data=test;
run;
```

Partial Log Output

```
ERROR: Execution terminated by an ABORT statement at line 85 column 21.
. . .
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.
        When this step was stopped there were 2 observations and 3 variables.
. . .
NOTE: There were 2 observations read from the data set WORK.TEST.
NOTE: The PROCEDURE PRINT printed page 10.
```

アウトプット 2.1 ABORT ステートメントを使用しての DATA ステップ処理の終了の PROC PRINT 出力

Obs	mass	volume	density
1	100	20	5
2	200	20	10

DATA ステップで ABORT ステートメントを実行する場合、SAS は既存のデータセットを同じ名前のデータセットに置き換えません。

たとえば、次のプログラムでは、最初の DATA ステップで、2 番目の DATA ステップの入力として使用されるデータセットが作成されます。2 番目の DATA ステップには ABORT ステートメントが含まれており、入力データセットと出力データセットの名前は同じです(test)。したがって、2 番目の DATA ステップでは結果が生成されません。PROC PRINT は、最初の DATA ステップで作成されたデータセットのみを表示します。

```
data test;
input mass volume;
datalines;
100 20
200 20
300 0
400 20
;
data test;
set test;
if volume=0 then abort;
density=mass/volume;
run;
proc print data=test; run;
```

アウトプット 2.2 最初の DATA ステップの結果のみを示す PROC PRINT 出力

Obs	mass	volume
1	100	20
2	200	20
3	300	0
4	400	20

例 2: 条件コードを使用して SAS の DATA ステップ処理を終了する

この例では、IF-THEN ステートメント内で ABORT n ステートメントを使用しており、ゼロ除算の条件が発生するデータ値が検出された場合に DATA ステップ処理を停止します。

```
data test;
mass=100;
volume=0;
if volume=0 then abort 255;
density=mass/volume;
run;
```

```
Partial Log Output
ERROR: Execution terminated by an ABORT statement at line 86 column 21.
. . .
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.TEST may be incomplete.
       When this step was stopped there were 0 observations and 3 variables.
NOTE: DATA statement used (Total process time):
```

ABORT ステートメントが実行されると、SAS は条件コード 255 を動作環境に返します。

例 3: SAS セッションを終了する

この例では、**ABORT EXIT** ステートメントと条件付き **IF-THEN** ステートメントを使用して、ゼロ除算条件が発生したときに SAS セッションを終了します。SAS セッションは SAS Compute Server 上で実行されています。

最初の DATA ステップでは、例の入力データを作成します。2 番目の DATA ステップでは、**ABORT EXIT** ステートメントを指定します。

```
data test;
  input mass volume;
datalines;
100 20
200 20
300 0
400 20
;

data test2;
  set test;
  if volume=0 then abort exit;
  density=mass/volume;
run;
proc print data=test2; run;
```

Partial Log Output

```
NOTE: The data set WORK.TEST has 4 observations and 2 variables.
NOTE: Execution terminated by an ABORT EXIT statement at line 94 column 21.
NOTE: There were 3 observations read from the data set WORK.TEST.
WARNING: The data set WORK.TEST2 may be incomplete.
         When this step was stopped there were 2 observations and 3 variables.
WARNING: Data set WORK.TEST2 was not replaced because this step was stopped.
```

例 4: CAS の DATA ステップ処理を終了する

この例では、**IF-THEN/ELSE** ステートメントとともに **ABORT** ステートメントの形式 2 を使用して、SAS が name 変数で文字値 J を検出したときに CAS サーバー上の DATA ステップ処理を停止します。

注: CAS で実行中の DATA ステップで指定された場合、ABORT ステートメントは DATA ステップの処理のみを終了します。ABORT ステートメントは、CAS サーバーセッションを終了しません。

```
cas casauto;
libname mycas cas;

/* Create a CAS input table for the example */
data mycas.class;
  set sashelp.class;
```



```

run;
proc print data=mycas.class; run;

data mycas.class2;
  set mycas.class;
  Jname=find(name,"J");
  if Jname then abort;
run;
proc print data=mycas.class2; run;

```

アウトプット 2.3 例の CAS 入力テーブル作成の PROC Print 出力

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

アウトプット 2.4 ABORT ステートメントを使用して CAS サーバー上の DATA ステップ処理を終了した PROC Print 出力

Obs	Name	Sex	Age	Height	Weight	Jname
1	Alfred	M	14	69.0	112.5	0
2	Alice	F	13	56.5	84.0	0
3	Barbara	F	13	65.3	98.0	0
4	Carol	F	14	62.8	102.5	0
5	Henry	M	14	63.5	102.5	0

アウトプット 2.5 ABORT ステートメントを使用して CAS サーバー上の DATA ステップ処理を終了した部分的なログ出力

```

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
ERROR: Execution terminated by an ABORT statement at line 92 column 15.
Name=James Sex=M Age=12 Height=57.3 Weight=83 Jname=1 _ERROR_=1 _N_=6
NOTE: There were 6 observations read from the table CLASS in caslib
CASUSER(sasdemo).
NOTE: The table class2 in caslib CASUSER(sasdemo) has 5 observations and 6
variables.
NOTE: The SAS System stopped processing this step because of errors.

```

関連項目

ステートメント:

- “STOP ステートメント” (372 ページ)
- “定義” (1 ページ)

ARRAY ステートメント

配列の要素を定義します。

該当要素: DATA ステップ

カテゴリ: CAS
In-Database 処理
情報

種類: 宣言

制限事項: 配列の初期化時に、VARCHAR データタイプの変数をサポートします。

要件 DATA ステップで変数を読み込んだり書き出したりする際に、変数を VARCHAR データ型として保持したい場合は、CAS エンジンが必要となります。

構文

```
ARRAY array-name { subscript }  
<$ length | length | VARCHAR(length) | VARCHAR(*)>  
<array-elements> <(initial-value-list)>;
```

引数

array-name

配列の名前を指定します。

制限事項 *Array-name* には、同じ DATA ステップ内の SAS 変数名とは異なる SAS 名を指定してください。

array-name 引数には、IF、SELECT、SUBSTR、INPUT、PUT、WHEN、WHILE、UNTIL、RETURN のいずれの名前も使用できません。

注意 SAS 関数の名前を配列名として使用すると、予期しない結果が生じることがあります。関数名を誤って配列名として使用すると、DATA ステップの実行時に、指定した名前を含むかっこ付きの参照名は関数の呼び出しではなく配列参照として解釈されます。この結果、警告メッセージが SAS ログに書き込まれます。

{subscript}

アスタリスク、数値、または数値の範囲を使用して、配列内の要素数と配置順序を指定します。*subscript* は、次のいずれかの形式になります。

{dimension-size(s)}

配列の各次元の要素数を指定します。*dimension-size* は、1 次元配列の要素数または多次元配列の各次元の要素数を示す数値です。

ヒ subscript は中かっこ({})、大かっこ([])、丸かっこ(())で囲むことができ
ン ます。
ト

例 次の ARRAY ステートメントは、SIMPLE という名前の 1 次元配列を定義
します。SIMPLE 配列には、RED、GREEN、YELLOW という 3 つの変数
がまとめられています。

```
array simple{3} red green yellow;
```

次元が複数ある配列は、多次元配列と呼ばれます。多次元配列の次元数に制限はありません。たとえば、2 次元配列は、配列要素の行列配置を提供します。SAS では変数を 2 次元配列に配置する場合、配列の上部左端から開始して、すべての行を順番に書き込みます(これを「行順」と呼びます)。次のステートメントで、5 行 3 列で構成される 2 次元配列を定義します。

```
array x{5,3} score1-score15;
```

{<lower :>upper<, ...<lower :> upper>}

配列の各次元の境界で、*lower* はその次元の下界、*upper* は上界です。

範 多くの明示的な配列では、配列の各次元の subscript は 1 から *n* までに
囲 なります。*n* は、この次元の要素数になります。

ヒ ほとんどの配列では、1 が便利な下限値となります。したがって、下限
ン 値と上限値を指定する必要はありません。ただし、配列次元を 1 以外の
ト 数値から開始する場合、上限値と下限値の両方を指定することをお勧め
します。

subscript の評価に必要な時間を短縮するには、下限として 0 を指定します。

例 デフォルトでは、この例の各次元の値はその次元の上限値となっています。

```
array x{5,3} score1-score15;
```

次の ARRAY ステートメントでは、もう 1 つの方法として前の例を長い形式で示します。

```
array x{1:5,1:3} score1-score15;
```

{*}

配列内の変数の数から subscript を算出するように指定します。アスタリスクを指定する場合、*array-elements* も指定する必要があります。

制限事 アスタリスクは、_TEMPORARY_ 配列では使用できないほか、多次
項 元配列の定義時にも使用できません。

\$

配列内の要素が文字要素であることを指定します。

ヒント 配列要素が文字要素であることがすでに定義されている場合、ドル記号を指定する必要はありません。

length

配列要素の長さが設定されていない場合に配列要素の長さを指定します。数値変数と文字変数の場合は、変数に格納されている最大バイト数です。

範囲 数値変数の場合、動作環境に応じて、2 から 8 バイト、または 3 から 8 バイトまでの値を指定できます。文字変数の場合、どの動作環境でも 1 から 32767 バイトまでの値を指定できます。

VARCHAR

直前にある変数が、VARCHAR データタイプの文字変数であることを指定します。

length

VARCHAR 変数に格納される最大文字数を指定します。

範囲 VARCHAR 変数の場合、1 文字から 536,870,911 文字まで

制限事項 VARCHAR 変数に文字定数を割り当てる場合、その文字定数は 32767 バイトまでに制限されます。

*

VARCHAR 変数に許可される最大長の 536,870,911 文字をサポートするように指定します。

array-elements

配列を構成する要素の名前を指定します。*array-elements* は、すべて数値で指定するか、すべて文字で指定します。任意の順序で指定できます。要素には、名前付きの変数や一時的なデータ要素があります。

variables

変数名をリストします。

範囲 この名前は、ARRAY ステートメントにて定義した変数か、配列名と数値を組み合わせる SAS によって作成された変数のいずれかになります。たとえば、subscript がアスタリスクではなく数値である場合、配列内にある各変数の名前を指定する必要はありません。かわりに、配列名と数値 1、2、3、...*n* を組み合わせた変数名が SAS によって作成されます。

制限事項 _ALL_ を使用する場合、過去に定義済みの変数はすべて同一の種類である必要があります。

ヒント このような SAS 変数のリストにより、同一の DATA ステップで過去に定義済みの変数を参照できます。

NUMERIC は、すべての数値変数を示します。

CHARACTER は、すべての文字変数を示します。

ALL を使用すると、すべての変数を示します。

名前の範囲リストでは、数値変数や文字変数の範囲を、それらの変数を定義した順番に基づいて定義することができます。

参照 “変数リスト” (SAS プログラマガイド: エッセンシャル).
項目:

例 “例 1: 配列を定義する” (45 ページ)

TEMPORARY

一時データ要素で構成されるリストを作成します。

範囲 一時データ要素は、数値または文字列になります。

ヒント 一時データ要素は DATA ステップの変数と同じように動作しますが、次の例外があります。

一時データ要素には名前がありません。一時データ要素の参照には、配列名と次元が使用されます。

出力データセットには書き出されません。

すべての要素を参照するために、特殊な subscript であるアスタリスク (*) は使用できません。

一時データ要素の値は常に自動的に保持されます。この値は、DATA ステップの次の繰り返しの先頭に戻った際に欠損値にリセットされます。

配列を作成する目的が演算の実行のみにある場合、一時要素の配列を使用すると便利です。演算結果を保存するには、その結果を変数に割り当てます。一時データ要素を使用すると処理時間を短縮することができます。

(initial-value-list)

配列内の対応する要素に初期値を指定します。要素の値には、数値または文字列を指定します。文字列はすべて引用符で囲む必要があります。1 つまたは複数の初期値を直接指定するには、次の形式を使用します。

(initial-value(s))

繰り返しの回数とネスト形式のサブリストを初期値に指定するには、次の形式を使用します。

`<constant-iter-value*> <(>constant value | constant-sublist<)>`

制限事項 *initial-value-list* と *array-elements* の両方を指定する場合、ARRAY ステートメントでは *array-elements* を *initial-value-list* より前に指定する必要があります。

ヒント 変数と一時データ要素の両方に初期値を割り当てることができます。

要素と値は位置を基準にして照合されます。指定した初期値の数よりも配列数が多い場合、初期値が割り当てられなかった配列要素には欠損値が割り当てられます。また、警告メッセージが発行されます。

初期値のリストはカンマまたはブランクを使用して区切ることができます。

連続する整数値の範囲を指定する場合は、簡略化した表記を使用できます。増分は常に+1 になります。

配列要素の属性(長さや種類など)を指定していない場合は、指定した初期値の属性が対応する配列要素に自動的に割り当てられます。初期値は、配列要素に新しい値が割り当てられるまで保持されます。

一部またはすべての要素に初期値を割り当てると、すべての要素は RETAIN ステートメントに指定した場合と同じように動作します。

例 次の例では、反復係数とネストされたサブリストを使用する方法を示します。これらの ARRAY ステートメントはすべて、同じ初期値リストを含んでいます。

```
ARRAY x{10} x1-x10 (10*5);
ARRAY x{10} x1-x10 (5*(5 5));
ARRAY x{10} x1-x10 (5 5 3*(5 5) 5 5);
ARRAY x{10} x1-x10 (2*(5 5) 5 5 2*(5 5));
ARRAY x{10} x1-x10 (2*(5 2*(5 5)));
```

例 “例 2: 数値の初期値を割り当てる” (45 ページ)

“例 3: 文字列の初期値を割り当てる” (45 ページ)

詳細

ARRAY ステートメントは、グループとして処理する要素のまとまりを定義します。配列の要素は、配列名と subscript を基準にして参照されます。1 つの配列内で複数の要素を処理する場合が多いため、配列は DO グループ内で頻繁に参照されます。

比較

- SAS 言語で使用する配列は、他の多くの言語で使用する配列とは異なります。SAS の配列は、変数のグループを一時的に識別するための便利な方法として使用します。この配列はデータ構造ではありません。また、*array-name* は変数ではありません。
- ARRAY ステートメントは、配列を定義します。配列参照は、プログラムステートメントにて配列要素を使用します。

例

例 1: 配列を定義する

以下の例では、ARRAY ステートメントを使って配列を定義する方法を紹介しています。

- `array rain {5} janr febr marr aprr mayr;`
- `array days{7} d1-d7;`
- `array month{*} jan feb jul oct nov;`
- `array x{*} _NUMERIC_;`
- `array qbx{10};`
- `array meal{3};`

例 2: 数値の初期値を割り当てる

これらの例では、配列を定義し、配列の要素に数値の初期値を割り当てる方法を示しています。

- `array test{4} t1 t2 t3 t4 (90 80 70 70);`
- `array test{4} t1-t4 (90 80 2*70);`
- `array test{4} _TEMPORARY_ (90 80 70 70);`

例 3: 文字列の初期値を割り当てる

この例では、配列を定義し、配列の要素に文字列の初期値を割り当てる方法を紹介します。

- `array test2{*} $ a1 a2 a3 ('a','b','c');`

例 4: 高度な配列を定義する

- `array new{2:5} green jacobs denato fetzer;`
- `array x{5,3} score1-score15;`
- `array test{3:4,3:7} test1-test10;`
- `array temp{0:999} _TEMPORARY_;`
- `array x{10} (2*1:5);`

例 5: 先頭に 0 を持つ変数名の範囲を作成する

この例では、先頭に 0 を持つ変数名の範囲を作成します。各変数名の長さは 3 文字です。この変数名は A01、A02、...A10 の順に正しく並べ替えられます。先頭に 0 を付けないと、変数名は A1、A10、A2、...A9 の順序で並べ替えられます。

```
data test (drop=i);
  array a{10} A01-A10;
  do i=1 to 10;
    a{i}=i;
  end;
run;
```

```
proc print noobs data=test;
run;
```

アウトプット 2.6 先頭に 0 を持つ配列名

The SAS System									
A01	A02	A03	A04	A05	A06	A07	A08	A09	A10
1	2	3	4	5	6	7	8	9	10

関連項目

- [“配列の使用” \(SAS プログラマガイド: エッセンシャル\)](#)

ステートメント:

- [“配列参照 ステートメント” \(46 ページ\)](#)

配列参照 ステートメント

処理する配列要素を記述します。

該当要素:	DATA ステップ
カテゴリ:	CAS In-Database 処理 情報
種類:	宣言
制限事項:	VARCHAR データタイプの変数はサポートされていません。

構文

```
array-name {subscript};
```

引数

- array-name**
同一の DATA ステップ内で ARRAY ステートメントによってすでに定義されている配列名です。
- {subscript}**
subscript(添字)を指定します。次のいずれかの形式を使用できます。

{variable-1< , ...variable-n>}

通常、DO ループの処理で使用する変数または変数のリストを指定します。DO ループの実行時ごとに、この変数の現在の値が処理される配列要素の subscript になります。

ヒント subscript は中かっこ({})、大かっこ([])、丸かっこ(())で囲むことができます。

例 [“例 1: 反復 DO ループ処理を使用する” \(48 ページ\)](#)

{*}

配列内の要素を変数リストとして強制的に処理するように指定します。

制限 一時配列要素を含む配列を定義する際、アスタリスクを使用して配列要素を参照することはできません。

ヒント アスタリスクは、INPUT ステートメント、PUT ステートメント、および一部の SAS 関数で使用できます。

この構文は処理を容易にするために提供されていますが、通常の配列処理では使用できません。

例 [“例 4: 変数リストとしてアスタリスクを使用する” \(50 ページ\)](#)

expression-1< , ...expression-n>

SAS 式を指定します。

範囲 配列参照を含むステートメントの実行時、この式の値が subscript 値になります。また、この式の結果は、配列数の下限値と上限値の範囲内の整数になります。

例 [“例 3: subscript を指定する” \(49 ページ\)](#)

詳細

- プログラムステートメント内で配列を参照するには、配列参照を使用します。配列を定義する ARRAY ステートメントは、DATA ステップ内でこの配列の参照箇所よりも前の位置に指定する必要があります。配列定義は DATA ステップの実行時のみ有効です。複数の DATA ステップで同じ配列を使用する場合、ステップごとに配列を再定義します。

注意

SAS 関数の名前を配列名として使用すると、予期しない結果が生じることがあります。 関数名を誤って配列名として使用すると、DATA ステップの実行時に、指定した名前を含むかっこ付きの参照名は関数の呼び出しではなく配列参照として解釈されます。この結果、警告メッセージが SAS ログに書き込まれます。

- 配列参照は、SAS 関数や SAS ステートメントなどの SAS 式を記述できる場所であればどこにでも指定できます。
 - 割り当てステートメント

- 合計ステートメント
- DO UNTIL(*expression*)
- DO WHILE(*expression*)
- IF
- INPUT
- PUT
- SELECT
- WINDOW
- 配列次元の下限值が 1 の場合、配列次元の要素数を返すために、DIM 関数を反復 DO ステートメントと組み合わせてよく使用します。DIM 関数を使用すると、DO ステートメントの上限値を変更せずに配列要素の数を変更することができます。例えば、DIM(NEW)の結果として 4 が返されるため、これらのステートメントでは配列内のすべての要素を処理します。

```
array new{*} score1-score4;
do i=1 to dim(new);
  new{i}=new{i}+10;
end;
```

- 配列の中に値が存在するかどうかは、IN 演算子を使って判断することができます。次に例を示します。

```
data test;
array v[3] (1 2 3);
if 2 in v then flag = 5;
run;
```

比較

ARRAY ステートメントでは配列を定義しますが、配列参照では処理対象となる配列メンバーを定義します。

例

例 1: 反復 DO ループ処理を使用する

この例では、DO ループを繰り返す際、変数 I の値を配列参照の subscript として使用して、配列の各要素を処理します。配列要素の値が 99 の場合、その値は IF-THEN ステートメントによって 100 に変更されます。

```
data test;
  input d1 d2 d3 d4 d5 d6 d7;
  datalines;
1 2 99 3 99 0 99
;
run;
```

```

data test;
  set test;
  array days{7} d1-d7;
  do i=1 to 7;
    if days{i}=99 then days{i}=100;
  end;
  put d1= d2= d3= d4= d5= d6= d7=;
run;

```

SAS は次の出力をログに書き込みます。

```
d1=1 d2=2 d3=100 d4=3 d5=100 d6=0 d7=100
```

例 2: 1 つのステートメントで複数の配列を参照する

1 つの SAS ステートメントで複数の配列を参照することができます。この例では、DAYS と HOURS という 2 つの配列を作成します。DO ループ内のステートメントでは、この 2 つの配列にある配列要素を参照して変数 I の現在の値を置き換えます。

```

data test;
  input d1 d2 d3 d4 d5 d6 d7;
  datalines;
1 2 99 3 99 0 99
;
run;

data test;
  set test;
  array days{7} d1-d7;
  array hours{7} h1-h7;
  do i=1 to 7;
    if days{i}=99 then days{i}=100;
    hours{i}=days{i}*24;
  end;
  put d1= d2= d3= d4= d5= d6= d7=;
  put h1= h2= h3= h4= h5= h6= h7=;
run;

```

SAS は次の出力をログに書き込みます。

```
d1=1 d2=2 d3=100 d4=3 d5=100 d6=0 d7=100
h1=24 h2=48 h3=2400 h4=72 h5=2400 h6=0 h7=2400
```

例 3: subscript を指定する

この例では、INPUT ステートメントを使用して、変数 A1 と変数 A2、配列 ARR1 の 3 番目の要素(A3)を読み取ります。

```

data test;
  array arr1{*} a1-a3;
  x=1;
  input a1 a2 arr1{x+2};

  datalines;
1 2 3

```

```
;
run;
```

例 4: 変数リストとしてアスタリスクを使用する

```
■ data test;
  input cost1 cost2 cost3 cost4 cost5 cost6 cost7 cost8 cost9 cost10;
  datalines;
10 20 30 40 50 60 70 80 90 100
;
run;
```

```
data test;
  set test;
  array cost{10} cost1-cost10;
  totcost=sum(of cost {*});
  put totcost=;
run;
```

```
■ data test;
  array days{7} d1-d7;
  input days {*};
  datalines;
1 2 3 4 5 6 7
;
run;
```

```
■ data test;
  input h1 h2 h3 h4 h5 h6 h7;
  datalines;
1 2 3 4 5 6 7
;
run;
```

```
data test;
  set test;
  array hours{7} h1-h7;
  put hours {*};
run;
```

関連項目

- [“Array Processing” \(SAS Language Reference: Concepts\)](#)

関数:

- [“DIM 関数” \(SAS 関数と CALL ルーチン: リファレンス\)](#)

ステートメント:

- [“ARRAY ステートメント” \(40 ページ\)](#)
- [“DO ステートメント: 反復” \(90 ページ\)](#)

割り当て ステートメント

式を評価し、その結果を変数に格納します。

該当要素:	DATA ステップ
カテゴリ:	アクション CAS In-Database 処理
種類:	実行

構文

variable=*expression*;

引数

variable

新しい変数名または既存の変数名を指定します。

範囲 *variable* には、変数名、配列参照、SUBSTR 関数を指定できます。

ヒント 割り当てステートメントで作成された変数は自動的に保持されません。

expression

SAS 式を指定します。

ヒント *expression* には、等号の左側で使用する変数を指定します。変数がステートメントの両側にある場合、右側の元の値が式の評価に使われます。結果は、等号の左側の変数に格納されます。詳細については、“[式](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

詳細

割り当てステートメントは、等号の右側の式を評価し、等号の左側に指定した変数に結果を格納します。

例: さまざまな式を使用した割り当てステートメント

さまざまな式を使用した割り当てステートメントを次に示します。

- `name='Amanda Jones';`
- `WholeName='Ms. '||name;`
- `a=a+b;`

関連項目

ステートメント:

- [“合計 ステートメント” \(374 ページ\)](#)

ATTRIB ステートメント

1 つまたは複数の変数に出力形式、入力形式、ラベル、長さを設定します。

該当要素: DATA ステップまたは PROC ステップ

カテゴリ: CAS

In-Database 処理

情報

種類: 宣言型

制限事項: ATTRIB ステートメントで VARCHAR 変数を宣言することはできません。
In-database DATA ステップの場合、ATTRIB ステートメントは INFORMAT=属性をサポートしていません。詳細については、“[DATA Step Acceleration](#)” ([SAS In-Database Products: User's Guide](#))を参照してください。

構文

ATTRIB *variable-list(s) attribute-list(s);*

引数

variable-list(s)

属性を設定する変数を指定します。

ヒント SAS で使用可能な任意の形式で変数をリストします。

attribute-list(s)

variable-list に割り当てる 1 つまたは複数の属性を指定します。ATTRIB ステートメントでは、次の属性を 1 つ以上指定します。

FORMAT=*format*

variable-list の変数に出力形式を割り当てます。

ヒント この出力形式は、標準の SAS 出力形式でも FORMAT プロシジャで定義した出力形式でもかまいません。

INFORMAT=*informat*

variable-list の変数に入力形式を割り当てます。

ヒント この入力形式は、標準の SAS 入力形式でも FORMAT プロシジャで定義した入力形式でもかまいません。

LABEL='*label*'

variable-list の変数にラベルを割り当てます。

LENGTH=<\$>*length*

variable-list の変数に長さを指定します。

範囲 文字変数の場合は、すべての動作環境で 1 バイトから 32767 バイトまで指定できます。

数値変数の場合、指定できる最小の長さは、オペレーティングシステムで使用される浮動小数点形式によって異なります。ほとんどのシステムでは IEEE 浮動小数点形式が使用され、最小値は 3 バイトです。

制限事項 DATASETS プロシジャから LENGTH=を使用して変数の長さを変更することはできません。

ATTRIB ステートメントを使用して、VARCHAR 変数を宣言することはできません。

要件 文字変数の長さの前には、ドル記号(\$)を挿入します。

ヒント SET ステートメントで既存のデータセットから変数を持ってくる前に、ATTRIB ステートメントで出力データセットの変数の長さを指定します。

TRANSCODE=YES | NO

文字変数をトランスコードするかどうかを指定します。トランスコードを実行しないようにするには、TRANSCODE=NO と指定します。

TRANSCODE=NO の変数は返されません。

デフォルト YES

制限事項 TRANSCODE=NO 属性は、SAS の以前のリリースの一部ではサポートされていません。以前のリリース(SAS Viya プラットフォームや SAS 9.4 より前)では、TRANSCODE=NO の変数がトランスコードされます。また、SAS の旧リリースは、TRANSCODE=NO 属性を持つ変数を含む SAS 9.4 データセットや SAS Viya プラットフォーム上のデータセットにアクセスできません。

操作 “[VTRANSCODE 関数](#)”および“[VTRANSCODEX 関数](#)”(SAS 各国語サポート(NLS): [リファレンスガイド](#))を使用すると、文字変数のトランスコードがオンまたはオフかどうかを示す値を返すことができます。

TRANSCODE=属性が NO に設定された文字変数がデータセット内にあると、PROC CONTENTS は、データセット内の変数ごとに、TRANSCODE=値が含まれるトランスコーディング列を出力します。データセットのすべての文字変数が TRANSCODE=YES (デフォルト)として指定されている場合、PROC CONTENTS 出力にトランスコーディング列は出力されません。

参照 [“各国語サポート関連のトランスコーディング” \(SAS 各国語サポート項目: リファレンスガイド\)](#)

詳細

基本情報

DATA ステップで ATTRIB ステートメントを使用すると、変数を含む SAS データセットのディスクリプター情報が変更されるため、変数と属性は恒久的に関連付けられます。

PROC ステップで ATTRIB ステートメントを使用することはできますが、規則が異なります。

ATTRIB ステートメントで INFORMAT=オプションを使用して入力形式を指定した場合の変数の処理

ATTRIB ステートメントで INFORMAT=オプションを使用して変数に割り当てた入力形式は、修飾リスト入力で使用される入力形式と同じように機能します。

注: 修飾リスト入力では、データを読み込むためにフォーマット修飾子を使用します。例えば、:(コロン)修飾子と入力形式を併用すると、8 バイトを超えるか、長さが異なる文字値を読み込むことができます。数値の場合は、:修飾子を使って非標準値を読み込むことができます。

ここでは、SAS が修飾リスト入力の変数をどのように扱うかを説明します。

- 入力形式の *w* の値を使用して、事前に定義されていない文字変数の長さを設定する
- 入力形式の *w* の値を使用して、外部ファイルの入力フィールド幅や列位置を指定しない
- 数値の入力形式の *w* の値は無視する
- 入力形式の *d* の値を、数値入力形式の場合と同じように使用する
- 入力データに埋め込まれているブランクは、INFILE ステートメントの DLM=オプションまたは DLMSTR=オプションを変更しない限り、区切り文字として処理する

フォーマット入力や列入力などの別の入力スタイルを使用するように INPUT ステートメントをコード化した場合、ATTRIB ステートメントの INFORMAT オプションの使用時にはその入力スタイルは使用されません。

SET ステートメントおよび MERGE ステートメント使用時のトランスコード変数の処理

SET ステートメントや MERGE ステートメントを使って複数のデータセットからデータセットを作成する場合、DATA ステップで読み込んだ最初のデータセットの TRANSCODE=値によって、出力データセットの TRANSCODE=値が決まります。例えば、“[例 3: MERGE ステートメントをトランスコード変数と組み合わせて使用する](#)”では、2つの入力データセット A と B には、共有変数 Var1 に対して異なる TRANSCODE=属性値が入っています。入力データセット A の値は NO、入力データセット B の値は YES です。これらの2つのデータセットが単一の出力データセット C に結合されたとき、変数 Var1 の TRANSCODE=属性値は、データセット A (MERGE ステートメントで読み込んだ最初の入力データセット)の値と等しいので NO となります。

“[例 2: SET ステートメントをトランスコード変数と組み合わせて使用する](#)” (56 ページ)と“[例 3: MERGE ステートメントをトランスコード変数と組み合わせて使用する](#)” (56 ページ)を参照してください。

注: TRANSCODE=属性は、変数が入力データセットまたは ATTRIB TRANSCODE=ステートメントで最初に検出されたときに設定されます。SET ステートメントまたは MERGE ステートメントが ATTRIB TRANSCODE=ステートメントの前に使用されている場合に、この TRANSCODE=属性が SET ステートメントまたは MERGE ステートメントと矛盾すると、警告メッセージが表示されます。

比較

変数に関連付けられた属性を変更する場合、ATTRIB ステートメントを使用することも、個々の属性のステートメントを使用することもできます。たとえば、これらのステートメントのいずれかを使用して、FORMAT、INFORMAT、LABEL、または LENGTH 属性を変更できます。

例

例 1: 指定する変数と属性の数が異なる ATTRIB ステートメントの例

さまざまな数の変数と属性を含む ATTRIB ステートメントの例を次に示します。

- 1 変数と 1 属性:
attrib cost length=4;
- 1 つの変数と複数の属性:
attrib saleday informat=mmddyy.
format=worddate.;
- 複数の変数と同じ複数の属性:
attrib x y length=\$4 label='TEST VARIABLE';

- 複数の変数と異なる複数の属性:

```
attrib x length=$4 label='TEST VARIABLE'
       y length=$2 label='RESPONSE';
```

- 変数リストと 1 つの属性:

```
attrib month1-month12
       label='MONTHLY SALES';
```

例 2: SET ステートメントをトランスコード変数と組み合わせて使用する

この例では、SET ステートメントを使用します。データセット A が読み込まれた最初の入力データセットであり、データセット A の Var1 の TRANSCODE=属性が NO であるため、出力データセット C の Var1 の TRANSCODE=属性は NO となります。

```
data A;
  length Var1 $4;
  Var1 = "rain";
  attrib Var1 transcode = no;
run;

data B;
  length Var1 $4;
  Var1 = "snow";
  attrib Var1 transcode = yes;
run;

data C;
  set A;
  set B;
  /* Check transcode setting for variable Var1 */
  rc1 = vtranscode(Var1);
  put rc1=;
run;
```

例 3: MERGE ステートメントをトランスコード変数と組み合わせて使用する

この例では、MERGE ステートメントを使用します。データセット A が MERGE ステートメントで読み込まれた最初の入力データセットであり、データセット A の Var1 の TRANSCODE=属性が NO であるため、出力データセット C の Var1 の TRANSCODE=属性は NO となります。

```
data A;
  length Var1 $4;
  Var1 = "rain";
  attrib Var1 transcode = no;
run;

data B;
  length Var1 $4;
  Var1 = "snow";
  attrib Var1 transcode = yes;
run;
```

```
data C;
  merge A B;
  /* Check transcode setting for variable Var1 */
  rc1 = vtranscode(Var1);
  put rc1=;
run;
```

関連項目

- [“名前の長さの規則” \(SAS プログラマガイド: エッセンシャル\)](#)

関数:

- [“VTRANSCODE 関数” \(SAS 各国語サポート\(NLS\): リファレンスガイド\)](#)
- [“VTRANSCODEX 関数” \(SAS 各国語サポート\(NLS\): リファレンスガイド\)](#)

ステートメント:

- [“FORMAT ステートメント” \(131 ページ\)](#)
- [“INFORMAT ステートメント” \(182 ページ\)](#)
- [“LABEL ステートメント” \(230 ページ\)](#)
- [“LENGTH ステートメント” \(237 ページ\)](#)

BY ステートメント

DATA ステップ内の SET、MERGE、MODIFY、UPDATE の各ステートメントの実行を制御し、特別なグループ変数を設定します。

該当要素: DATA ステップまたは PROC ステップ

カテゴリ: CAS
ファイル操作

種類: 宣言

注: このステートメントは、[2025.04](#) 以降の Databricks および [2025.06](#) 以降の SingleStore でサポートされています。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#) を参照してください。

参照項目: [“BY グループ処理の定義” \(SAS プログラマガイド: エッセンシャル\)](#)

例:

```
data ShoesByRegion;
  set sashelp.shoes;
  by descending Region
     descending Subsidiary;
run;
```

例: [“変数のグループ化と降順の並べ替え” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)

構文

BY <DESCENDING> *variable-1* <...> <DESCENDING> *variable-n*
<NOTSORTED> <GROUPFORMAT>;

必須引数

variable

データセットの並べ替えやインデックス付けに使用される変数を指定します。
指定した変数は、現在の DATA ステップまたは PROC ステップの BY 変数として参照されます。

要件 BY グループ処理で BY 変数として名前リテラルを指定し、対応する FIRST、一時変数または LAST、一時変数を参照する場合、FIRST、または LAST が含まれる 2 レベル変数名を単一引用符で囲んで指定する必要があります。次に例を示します。

```
data sedanTypes;
  set cars;
  by 'Sedan Types'n;
  if 'first.Sedan Types'n then type=1;
run;
```

ヒント データセットでは、複数の変数を基準に並べ替えやインデックスの作成を実行できます。

例 [“例 1: BY 変数の指定” \(61 ページ\)](#)

[“例 2: 並べ替え順序の指定” \(62 ページ\)](#)

[“例 3: 並べ替えを実行していないデータの BY グループ処理” \(62 ページ\)](#)

[“例 4: フォーマット値を用いてオブザベーションをグループ化する” \(62 ページ\)](#)

オプション引数

DESCENDING

指定した変数の値を基にデータセットを降順で並べ替えるように指定します。降順では、数値変数であれば最大値から最小値に、文字変数であれば逆アルファベット順に並べ替えが実行されます。

制限事項 CAS で DATA ステップを実行する場合、BY ステートメントの最初の変数に DESCENDING オプションを使用することはできません。

インデックス付きデータセットには DESCENDING オプションを使用できません。これは、インデックスが常に昇順で保存されるためです。

例

[“例 2: 並べ替え順序の指定” \(62 ページ\)](#)

GROUPFORMAT

BY 変数の内部値ではなく、フォーマット値を使用して、BY グループが開始し、終了する場所を判断します。GROUPFORMAT は、*FIRST.variable* と *LAST.variable* の割り当て方も決定します。GROUPFORMAT オプションは BY ステートメントのどの位置に指定してもかまいません。このオプションは、BY ステートメントにある *すべての変数*に適用されます。

制限事項 BY ステートメントで GROUPFORMAT オプションを使用する前に、BY 変数の値に基づいてデータセット内のオブザベーションを並べ替える必要があります。

BY ステートメントの GROUPFORMAT オプションは、DATA ステップのみで使用できます。

操作 また、NOTSORTED オプションを使用する場合、データセットを並べ替えたり、インデックスを作成することなく、データセット内のオブザベーションを BY 変数のフォーマット値に基づいてグループ化することができます。

注 GROUPFORMAT を使用した DATA ステップでの BY グループ処理は、SAS プロシジャでのフォーマット値の BY グループ処理と同じです。

ヒント GROUPFORMAT オプションは、グループ化するデータの表示に独自の出力形式を定義する場合に便利です。

DATA ステップで GROUPFORMAT オプションを使用すると、データセットの作成に使用する BY グループと、グループ化され、フォーマットされたデータをレポートする PROC ステップの BY グループを一致させることができます。

参照項目: [“DATA ステップでの BY グループの処理” \(SAS プログラマガイド: エッセンシャル\)](#)

例 [“例 4: フォーマット値を用いてオブザベーションをグループ化する” \(62 ページ\)](#)

NOTSORTED

同じ BY 変数の値を含むオブザベーションをグループ化します。ただし、これらのオブザベーションはアルファベット順または数値順では並べ替えられません。

制限事項 MERGE および UPDATE ステートメントでは、NOTSORTED オプションを使用できません。

ヒント NOTSORTED オプションは BY ステートメントのどの位置に指定してもかまいません。

NOTSORTED オプションは、日付順やカテゴリなど他の論理グループに分類されるデータが存在する場合に便利です。

例 [“例 3: 並べ替えを実行していないデータの BY グループ処理” \(62 ページ\)](#)

詳細

BY グループの開始と終了の識別について

BY グループの開始と終了は、BY 変数に 2 つの一時変数を作成して識別します。FIRST.variable and LAST.variable. この 2 つの変数の値は 0 または 1 のどちらかになります。FIRST.変数の値は、BY グループの最初のオブザベーションを読み込むときに 1 に設定されます。LAST.変数の値は、BY グループの最後のオブザベーションを読み込むときに 1 に設定されます。これら 2 つの一時変数は、DATA ステップでのプログラミングには利用できますが、出力データセットには変数として書き出されません。

SAS がグループ化したデータを処理する方法およびデータを準備する方法の詳細については、“[DATA ステップでの BY グループの処理](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

DATA ステップで BY ステートメントを使用する

BY ステートメントは、DATA ステップで先行する SET、MERGE、MODIFY、UPDATE のいずれかのステートメントにのみ適用されます。DATA ステップでは、1 つの BY ステートメントのみ、これらの各ステートメントに指定できます。

SET、MERGE、UPDATE のステートメントにリストされたデータセットは、BY ステートメントにリストされている変数の値で並べ替えられているか、適切なインデックスを持つ必要があります。デフォルトでは、データセットは昇順の数値順またはアルファベット順で並べられていると判断されます。オブザベーションは次のいずれかの方法で配置できます。

- データセットを並べ替える
- 変数のインデックスを作成する
- オブザベーションを順番に読み込む

注: MODIFY ステートメントではデータを並べ替える必要はありませんが、並べ替えを実行するとパフォーマンスを向上させることができます。

注: BY ステートメントでは、SORTSEQ=LINGUISTIC オプションが指定された SORT プロシジャを使用して並べ替えたデータの言語照合が優先されます。

PROC ステップで BY ステートメントを使用する

BY ステートメントを一部の SAS プロシジャで使用すると、その SAS プロシジャのアクションを変更することができます。BY ステートメントが SAS プロシジャの処理に与える影響については、[Base SAS プロシジャガイド](#)にある各プロシジャの説明を参照してください。

SAS ビューで BY ステートメントを使用する

DBMS からの読み込みに基づき DATA ステップビューを作成する際、SET、MERGE、UPDATE、MODIFY の各ステートメントの後ろに BY ステートメントが指定されてい

る場合、BY ステートメントでは DBMS 側でデータを並べ替えを行い、並べ替えたデータを返すことがあります。データの並べ替えが発生すると、実行時間が長くなります。

SAS による BY グループの処理方法

FIRST.変数と LAST.変数には、次の値が割り当てられます。

- FIRST.変数は、次の場合に値が 1 に設定されます。
 - 現在のオブザベーションがデータセットから読み込む最初のオブザベーションである場合。
 - GROUPFORMAT オプションを使用していないときに、現在のオブザベーションにある変数の内部値が前のオブザベーションにある内部値と異なる場合。
GROUPFORMAT オプションを使用しているとき、FIRST.変数が 1 に設定されるのは、現在のオブザベーションにある変数のフォーマット値が前のオブザベーションにあるフォーマット値と異なる場合です。
 - FIRST.変数が、BY ステートメント内でその前にあるどの変数に対しても 1 に設定される場合。
該当しない場合、FIRST.変数の値は 0 になります。
- LAST.変数は、次の場合に値が 1 に設定されます。
 - 現在のオブザベーションがデータセットから読み込む最後のオブザベーションである場合。
 - GROUPFORMAT オプションが使用されているときに、現在のオブザベーションにある変数の内部値が次のオブザベーションにある内部値と異なる場合。
GROUPFORMAT オプションが使用されている場合、LAST.変数が 1 に設定されるのは、現在のオブザベーションにある変数のフォーマット値が次のオブザベーションにあるフォーマット値と異なる場合です。
 - LAST.変数が、BY ステートメント内でその前にあるどの変数に対しても 1 に設定される場合。
該当しない場合、LAST.変数の値は 0 になります。

例

例 1: BY 変数の指定

- オブザベーションは DEPT 変数の昇順で並べられます。
by dept;
- オブザベーションは CITY のアルファベット順(昇順)で並べられます。CITY の値内では、ZIPCODE の昇順で並べられます。
by city zipcode;

例 2: 並べ替え順序の指定

注: DESCENDING は CAS で実行する DATA ステップではサポートされていません。

- オブザベーションは SALESREP の昇順で並べられます。SALESREP の値内では、JANSALES の値の降順で並べられます。

```
by salesrep descending jansales;
```

- オブザベーションは BEDROOMS の降順で並べられます。BEDROOMS の値内では、PRICE の値の降順で並べられます。

```
by descending bedrooms descending price;
```

例 3: 並べ替えを実行していないデータの BY グループ処理

オブザベーションは費用が発生した月の名前で並べられます。

```
by month notsorted;
```

例 4: フォーマット値を用いてオブザベーションをグループ化する

次に GROUPFORMAT オプションを使用例を示します。

```
proc format;
  value range
    low -55 = 'Under 55'
    55-60 = '55 to 60'
    60-65 = '60 to 65'
    65-70 = '65 to 70'
    other = 'Over 70';
run;
proc sort data=sashelp.class out=sorted_class;
  by height;
run;
data _null_;
  format height range.;
  set sorted_class;
  by height groupformat;
  if first.height then
    put 'Shortest in ' height 'measures ' height:best12.;
run;
```

SAS は次の出力を SAS ログに書き込みます。

```
Shortest in Under 55 measures 51.3
Shortest in 55 to 60 measures 56.3
Shortest in 60 to 65 measures 62.5
Shortest in 65 to 70 measures 65.3
Shortest in Over 70 measures 72
```


例 5: 複数のオブザベーションを結合した後で 1 つの BY の値を基にグループ化する

次の例では、BY グループの処理で FIRST.変数と LAST.変数を使用する方法を示します。

```
data Inventory;
  length RecordID 8 Invoice $ 30 ItemLine $ 50;
  infile datalines;
  input RecordID Invoice ItemLine &;
  drop RecordID;
  datalines;
A74 A5296 Highlighters
A75 A5296 Lot # 7603
A76 A5296 Yellow Blue Green
A77 A5296 24 per box
A78 A5297 Paper Clips
A79 A5297 Lot # 7423
A80 A5297 Small Medium Large
A81 A5298 Gluestick
A82 A5298 Lot # 4422
A83 A5298 New item
A84 A5299 Rubber bands
A85 A5299 Lot # 7892
A86 A5299 Wide width, Narrow width
A87 A5299 1000 per box
;
data combined;
  array Line{4} $ 60 ;
  retain Line1-Line4;
  keep Invoice Line1-Line4;
  set Inventory;
  by Invoice;

  if first.Invoice then do;
    call missing(of Line1-Line4);
    records = 0;
  end;

  records + 1;
  Line[records]=ItemLine;

  if last.Invoice then output;
run;
proc print data=combined;
  title 'Office Supply Inventory';
run;
```

アウトプット 2.7 結合した複数のオブザベーションからの出力

Office Supply Inventory					
Obs	Line1	Line2	Line3	Line4	Invoice
1	Highlighters	Lot # 7603	Yellow Blue Green	24 per box	A5296
2	Paper Clips	Lot # 7423	Small Medium Large		A5297
3	Gluestick	Lot # 4422	New item		A5298
4	Rubber bands	Lot # 7892	Wide width, Narrow width	1000 per box	A5299

関連項目

ステートメント:

- [“MERGE ステートメント” \(252 ページ\)](#)
- [“MODIFY ステートメント” \(261 ページ\)](#)
- [“SET ステートメント” \(356 ページ\)](#)
- [“UPDATE ステートメント” \(380 ページ\)](#)

CALL ステートメント

SAS CALL ルーチン呼び出します。

該当要素:	DATA ステップ
カテゴリ:	アクション
	CAS
	In-Database 処理
種類:	実行

構文

CALL *routine*(*parameter-1* <, ...*parameter-n*>);

引数

routine
呼び出したい SAS CALL ルーチンの名前を指定します。

参照項目: 使用可能なルーチンの詳細については、[SAS 関数と CALL ルーチン: リファレンス](#)を参照してください。

(parameter)

ルーチンに渡す情報、またはルーチンから返される情報を指定します。

要件 ルーチンによっては、この情報を丸かっこで囲む必要があります。

ヒント カンマで区切るにより、他のパラメーターを追加指定することができます。

詳細

SAS CALL ルーチンを使用すると、値を割り当てたり、その他のシステム機能を実行できます。

関連項目

[SAS 関数と CALL ルーチン: リファレンス](#)

CARDS ステートメント

ステートメントの後にデータ行が続くことを指定します。

該当要素: DATA ステップ

カテゴリ: ファイル操作

種類: 宣言

別名: DATALINES
LINES

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。
このステートメントは、In-database DATA ステップではサポートされていません。
詳細については、“[DATA Step Acceleration](#)” ([SAS In-Database Products: User's Guide](#))を参照してください。

参照項目: “[DATALINES ステートメント](#)” (81 ページ)

CARDS4 ステートメント

ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	宣言
別名:	DATALINES4 LINES4
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。 詳細については、“ DATA Step Acceleration ” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。
参照項目:	“DATALINES4 ステートメント” (84 ページ)

CATNAME ステートメント

複数のカタログをカタログ参照名(ショートカット名)に関連付け、1 つのカタログに論理的に結合します。また、1 つまたはすべてのカタログ参照名をクリアします。1 つの連結またはすべての連結に含まれる連結カタログをリストします。

注: [CATNAME ステートメント](#)は SAS グローバルステートメントに移動しました。

CHECKPOINT EXECUTE_ALWAYS ステートメント

チェックポイント-再開データを無視して、このステートメントの直後にある DATA ステップまたは PROC ステップを実行するように指示します。

注: [CHECKPOINT EXECUTE_ALWAYS ステートメント](#)は SAS グローバルステートメントに移動しました。

コメント ステートメント

ステートメントまたはプログラムの目的を説明します。

注: [COMMENT ステートメント](#)は SAS グローバルステートメントに移動しました。

CONTINUE ステートメント

DO ループの現在の繰り返しを中止し、次の繰り返しから処理を再開します。

該当要素: DATA ステップ

カテゴリ:	CAS 制御 In-Database 処理
種類:	実行
制限事項:	DO ループでのみ使用できます。

構文

CONTINUE;

引数なし

CONTINUE ステートメントに引数はありません。このステートメントは、DO ループでのステートメントの現在の繰り返し処理を任意の条件に基づいて中止します。DO ループの次の繰り返しから処理を再開します。

比較

- CONTINUE ステートメントは、ループの現在の繰り返し処理を中止し、次の繰り返し処理から再開します。LEAVE ステートメントは、現在のループの処理を終了します。
- CONTINUE ステートメントは DO ループのみで使用できます。DO ループまたは SELECT グループ内で LEAVE ステートメントを使用します。

例: 他のステートメントの実行を回避する

この DATA ステップでは、正社員の福利厚生レポートを作成します。従業員のステータスが PT(パートタイム)の場合、この CONTINUE ステートメントでは 2 番めの INPUT ステートメントと OUTPUT ステートメントを実行しないようにします。

```
data new_emp;  
  drop i;  
  do i=1 to 5;  
    input name $ idno status $;  
    /* return to top of loop */  
    /* when condition is true */  
    if status='PT' then continue;  
    input benefits $10.;  
    output;  
  end;  
  datalines;  
Becker 9011 PT  
Tanaka 876 PT  
Green 1002 FT  
Eye/Dental
```

```
Kim 85111 PT
Patel 433 FT
HMO
;
```

関連項目

ステートメント:

- [“DO ステートメント: 反復” \(90 ページ\)](#)
- [“LEAVE ステートメント” \(236 ページ\)](#)

DATA ステートメント

DATA ステップを開始し、SAS データセット、ビュー、プログラムなどの出力の名前を提供します。

該当要素: DATA ステップ

カテゴリ: CAS
ファイル操作
In-Database 処理

種類: 宣言

構文

形式 1: **DATA statement for creating output data sets**

```
DATA <data-set-name-1 <(data-set-options-1)>>
<...data-set-name-n <(data-set-options-n)>>
</ <NESTING> <STACK=stack-size> <HEXLISTALL>> <NOLIST>;
```

形式 2: **DATA statement for creating a DATA step view**

```
DATA view-name <data-set-name-1 <(data-set-options-1)>>
<...data-set-name-n <(data-set-options-n)>> / VIEW=view-name
<(<password-option> <SOURCE=source-option> )>
<NESTING> <NOLIST>;
```

形式 3: **DATA statement for creating a stored compiled DATA step program**

```
DATA data-set-name / PGM=program-name
<(<password-option> <SOURCE=source-option> )>
<NESTING> <NOLIST>;
```

形式 4: **DATA statement for describing a DATA step view**

```
DATA VIEW=view-name <(password-option)> <NOLIST>;
DESCRIBE;
```

形式 5: **DATA statement for use with a stored compiled DATA step program**

```
DATA PGM=program-name <(password-option)> <NOLIST>;
  <DESCRIBE;>
  <REDIRECT INPUT | OUTPUT old-name-1 = new-name-1<... old-name-n = new-
    name-n> ;>
  <EXECUTE;>
```

形式 6: **DATA statement for use on the CAS server**

```
DATA<data-set-name-1 <(data-set-options-1)>>
<...data-set-name-n <(data-set-options-n)>>
/ <<SESSREF=session-reference-name> | <SESSUID=session-uuid> > >
<THREADS=number-of-threads> <SINGLE= NO | YES | NOINPUT>
<NESTING> <STACK=stack-size><HEXLISTALL> <NOLIST>;
```

引数なし

DATA ステートメントで出力データセットの名前を指定しなかったり、予約名の NULL を指定した場合、SAS は作成した連続したデータセットそれぞれに DATA1、DATA2... という名前を自動的に割り当てます。この機能は DATA n 命名規則と呼ばれています。現在許可されている n 値の最大値は 9999 です。

引数

data-set-name

DATA ステップで作成する SAS データファイルまたは DATA ステップビューの名前を指定します。DATA ステップビューを作成するには、*data-set-name* を少なくとも 1 つ指定する必要があります。また、*data-set-name* は *view-name* に一致させる必要があります。

制限事項 *data-set-name* は SAS の命名規則に準拠する必要があります。また、動作環境によっては、その他の制限事項が適用される場合があります。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

NULL がデータセット名として指定されている場合、DATA ステップの実行時にデータセットは作成されません。例については、[“例 5: カスタムレポートの作成” \(79 ページ\)](#)を参照してください。

参照項目: [“SAS 名” \(SAS プログラマガイド: エッセンシャル\)](#)

(data-set-options)

DATA ステップでオブザベーションを出力データセットに書き込むときに適用する引数を指定します。この引数はオプションです。

参照項目: [SAS データセットオプション: リファレンス](#)には、データセットオプションの定義や一覧が記載されています。

例 “例 1: 複数データファイルの作成とデータセットオプションの使用”
(78 ページ)

NOLIST

ERROR の値が 1 の場合、すべての変数を SAS ログに出力しないようにします。

制限事項 NOLIST は DATA ステートメントの最後のオプションとして指定する必要があります。

password-option

コンパイル済み DATA ステッププログラムまたは DATA ステップビューにパスワードを割り当てます。

注: パスワードで保護された DATA ステッププログラムを記述するには、パスワードを指定する必要があります。プログラムが複数のパスワードを持っている場合、一番厳格なパスワードを指定します。ALTER は一番厳格で、READ は最も制限が緩いものです。詳細については、“[DESCRIBE ステートメント](#)” (86 ページ) を参照してください。

次のパスワードオプションを指定できます。

ALTER=alter-password

SAS データファイルに ALTER パスワードを割り当てます。このパスワードを使用すると、コンパイル済み DATA ステッププログラムまたは DATA ステップビューの保護や置き換えが行えます。

別 名 PROTECT=

要件 コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に ALTER パスワードを使用する場合、プログラムやビューの置き換えには ALTER パスワードが必要です。

コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に ALTER パスワードを使用する場合、DESCRIBE ステートメントを実行するには ALTER パスワードが必要です。

PW=password

READ=パスワードと ALTER=パスワードを割り当てます。両方のパスワードに同じ値が設定されています。

READ=read-password

SAS データファイルに READ パスワードを割り当てます。このパスワードを使用すると、コンパイル済み DATA ステッププログラムまたは DATA ステップビューの読み取りや実行を行えます。

別 名 EXECUTE=

要件 コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用した場合、そのプログラムやビューを実行するには READ パスワードが必要となります。

コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用する場合、DESCRIBE ステートメントおよび EXECUTE ステートメントを実行するには READ パスワードが必要です。無効なパスワードを指定すると、DESCRIBE ステートメントだけが実行されます。

- ヒント コンパイル済み DATA ステッププログラムや DATA ステップビューの作成時に READ パスワードを使用する場合、プログラムやビューの置き換えにはパスワードは不要です。

PGM=*program-name*

DATA ステップで作成または実行するコンパイル済みプログラムの名前を指定します。コンパイル済みプログラムを作成する場合は、PGM=オプションの前にスラッシュ(/)を指定します。コンパイル済みプログラムを実行する場合は、スラッシュ(/)をつけずに PGM=オプションを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

ヒント ストアドプログラムの作成時、SAS マクロ変数の解決が行われます。マクロ変数の解決をビューの処理時まで延期する場合は、SYMGET 関数を使用します。

例 “例 4: コンパイル済みプログラムの保存と実行” (79 ページ)

SOURCE=*source-option*

次のソースオプションから 1 つ指定します。

ENCRYPTALL

保存されたコンパイル済み DATA ステッププログラムまたは DATA ステップビューを作成したソースコード(すべての文字列定数を含む)を暗号化して保存します。

注 2023.08 以降、ENCRYPTALL がオプション ENCRYPT に置き換わります。ENCRYPTALL は SAS004 暗号化を使用します。ENCRYPT は SAS 独自の暗号化を使用していました。SAS の暗号化レベルの詳細については“[概念: PWENCODE プロシジャ](#)” (*Base SAS プロシジャガイド*)を参照してください。より強力な暗号化を利用するには、ENCRYPT を使用して作成されたストアドプログラムとビューを再作成することを検討してください。

ENCRYPTALL で作成されたストアドプログラムとビューは、2023.08 より前のリリースでは使用できません。

ヒント ソースコードと文字定数を暗号化する場合は、ALTER パスワードオプションも使用します。ALTER パスワードを使用しないと、警告メッセージが表示されます。

NOSAVE

ソースコードは保存されません。

注意

DATA ステップビューに NOSAVE オプションを使用する場合、このビューを SAS のあるバージョンから別のバージョンにコピーすることはできません。

SAVE

コンパイル済み DATA ステッププログラムや DATA ステップビューを作成したソースコードを保存します。

デフォルト SAVE

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

VIEW=*view-name*

DATA ステップで入力 DATA ステップビューの格納に使用するビューの名前を指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

view-name は、出力データセット名のうちの 1 つと一致している必要があります。

1 つの DATA ステップで作成できるビューは 1 つだけです。

ヒント DATA ステートメントに追加データセットを指定する場合、後続の DATA ステップまたは PROC ステップでビューが処理されるときに追加データセットが作成されます。ビューには、ビューの実行時に他のデータセットを生成する機能があります。

ビューの作成時、SAS マクロ変数が解決されます。マクロ変数の解決をビューの処理時まで延期する場合は、SYMGET 関数を使用します。

例 “例 2: 入力 DATA ステップビューの作成” (78 ページ)

“例 3: ビューおよびデータファイルの作成” (78 ページ)

オプション引数

HEXLISTALL

LIST ステートメントで入力データの全行のログデータを 16 進数で書き込むことができます。

参照項目 LIST ステートメントで HEXLISTALL オプションを使用する方法については、“例 3: 入力データの 16 進数値のリスト” (247 ページ) を参照してください。

NESTING

DO-END および SELECT-END の各ネストレベルの開始と終了に関するメッセージを SAS ログに出力できます。このオプションを指定すると、一致していない DO-END ステートメントおよび SELECT-END ステートメントをデバッグすることができます。また、ネストレベルがわかりにくい大規模なプログラムに使用すると便利です。

STACK=*stack-size*

ネストする LINK ステートメントの最大数を指定します。

CAS で実行中の DATA ステップの引数

SESSREF=*session-reference-name*

DATA ステップを CAS セッションに関連付けます。SESSREF=オプションの値は、現在のアクティブなセッションを表します。

操作 DSCAS システムオプションが NODSCAS に設定されている場合、CAS で DATA ステップを実行するためには、SESSREF=または SESSUUID=オプションのどちらかが必要となります。

CAS で実行している DATA ステップに入力テーブルがない場合、SESSREF または SESSUUID オプションが指定されない限り、シングルスレッドで実行されます。

SESSUUID=*session-uuid*

DATA ステップが実行されている CAS セッションを指定します。*session-uuid* は、CAS セッションに関連付けられている UUID(Universally Unique Identifier) になります。セッションを作成し、そのセッション用の UUID を取得するには、CAS ステートメントまたは LIBNAME ステートメントを使用します。

操作 DSCAS システムオプションが NODSCAS に設定されている場合、CAS で DATA ステップを実行するためには、SESSREF=または SESSUUID=オプションのどちらかが必要となります。

CAS で実行している DATA ステップに入力テーブルがない場合、SESSREF または SESSUUID オプションが指定されない限り、シングルスレッドで実行されます。

ヒント セッションを作成し、そのセッション用の UUID を取得するには、CAS ステートメントまたは LIBNAME ステートメントを使用します。

SINGLE=NO | NOINPUT | YES

CAS の単一スレッドで DATA ステップを実行するかどうかを指定します CAS。デフォルトでは、CAS の入力テーブルが指定されていない場合、CAS で実行している DATA ステップはシングルスレッドで実行されます。CAS で実行している DATA ステップは、入力 CAS テーブルが指定されている場合、デフォルトでマルチスレッドで実行されます。例えば、以下の DATA ステップは、デフォルトで CAS のシングルスレッドで実行されます。

```
libname mycas cas;
data mycas.test;
  x='No input table specified';
run;
```

次のメッセージがログに返されます。 NOTE: The DATA step has no input data set and will run in a single thread.

この DATA ステップは、デフォルトでは CAS のマルチスレッドで実行されます。

```
libname mycas cas;
data mycas.class;
  set mycas.class;
```

```
run;
```

注意

CAS の単一スレッドで DATA ステップを実行した場合、各自の環境で使用可能なワーカーノードの数にかかわらず、処理は単一マシンに集約されます。 シングルスレッド DATA ステップで大きなテーブルを処理すると、プログラムの動作が遅くなるだけでなく、単一スレッドを実行するマシンがディスクやメモリのリソースを使い果たしてしまうことがあります。非常に大きなテーブルの場合、単一のスレッドで DATA ステップを実行することはお勧めしません。

CAS テーブルを再配布するには、Tables アクションセットの Partition アクションを使用します。[“SINGLE=オプション指定後のテーブル再配布” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)を参照してください

サーバーで実行するための要件が満たされていないために CAS での DATA ステップの実行に失敗した場合、DATA ステップは自動的に SAS で実行されます。SAS で DATA ステップを実行する場合、常にシングルスレッドで実行されます。

参照 項 目: [“スレッド処理の制御ルール” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)。

CAS で DATA ステップを実行するための要件の一覧については、[“CAS で DATA ステップを実行するための要件” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)を参照してください。

DATA ステップと CAS での処理の詳細については、[“シングルおよびマルチスレッド処理” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)を参照してください。

例

[“SINGLE=オプション指定後のテーブル再配布” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)

NOINPUT

CAS 入力テーブルが存在しない場合にのみ、データステッププログラムを単一スレッドで実行します。このオプションは、テーブルに対する出力データのみを生成する DATA ステップを記述する場合に便利です。NOINPUT は、入力テーブルが指定されていない場合、CAS における DATA ステップ処理のデフォルトです。

注意

CAS の単一スレッドで DATA ステップを実行した場合、各自の環境で使用可能なワーカーノードの数にかかわらず、処理は単一マシンに集約されます。 シングルスレッド DATA ステップで大きなテーブルを処理すると、プログラムの動作が遅くなるだけでなく、単一スレッドを実行するマシンがディスクやメモリのリソースを使い果たしてしまうことがあります。非常に大きなテーブルの場合、単一のスレッドで DATA ステップを実行することはお勧めしません。

NO

全スレッドで DATA ステップを実行します。4 つのワーカーが存在し、各ワーカーが 32 個のスレッドをサポートする場合、DATA ステップは 128 個のス

レッドで実行されます。NO は、入力テーブルが指定されている限り、CAS における DATA ステップ処理のデフォルトです。そうしないと、入力テーブルがない場合、CAS の DATA ステップはシングルレッドで実行されます。

YES

単一レッドで DATA ステップを実行します。入力テーブルからの行が処理されると、すべての行のコピーが 1 つのノードに転送され、シングルレッドで処理されます。この動作は、データの転送、データの複製、シングルレッドで可能な限られた処理のため、パフォーマンス上のペナルティが発生します。SAS で実行中の DATA ステップは、常にシングルレッドで実行されます。

注意

CAS の単一レッドで DATA ステップを実行した場合、各自の環境で使用可能なワーカーノードの数にかかわらず、処理は単一マシンに集約されます。シングルレッド DATA ステップで大きなテーブルを処理すると、プログラムの動作が遅くなるだけでなく、単一レッドを実行するマシンがディスクやメモリのリソースを使い果たしてしまうことがあります。非常に大きなテーブルの場合、単一のレッドで DATA ステップを実行することはお勧めしません。

デフォルト NOINPUT

参照項目: [“シングルおよびマルチレッド処理” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#).

[Partition アクション](#)

THREADS=*number-of-threads*

CAS で DATA ステップを実行するために使用するレッドの数を指定します。

要件 この数字は 0 以上である必要があります。値ゼロは、許可されている最大のレッド数を使用して DATA ステッププログラムを実行することを意味します。

参照項目: [“レッド処理の制御ルール” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)。

詳細

DATA ステートメントの使用

DATA ステートメントを使用すると、DATA ステップが開始されます。次のタイプの出力を作成するには、DATA ステートメントを使用します。

- [データセット](#)
- [データビュー\(VIEW=オプションを使用 \(72 ページ\)\)](#)
- [保存されたコンパイル済みプログラム\(PGM=オプション \(71 ページ\)を指定。\)](#)

DATA ステートメントには、複数の出力を指定できます。ただし、データビューは 1 つしか指定できません。

READ パスワードと ALTER パスワードの両方を使用する

コンパイル済み DATA ステッププログラムまたは DATA ステップビューの作成時に READ パスワードと ALTER パスワードを両方使用すると、次のようになります。

- コンパイル済み DATA ステッププログラムまたは DATA ステップビューを実行するには、READ パスワードまたは ALTER パスワードが必要になります。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューに DESCRIBE ステートメントおよび EXECUTE ステートメントが含まれている場合は、READ パスワードまたは ALTER パスワードが必要になります。
 - ALTER パスワードをこの DESCRIBE ステートメントおよび EXECUTE ステートメントに使用する場合、次のようになります。
 - DESCRIBE ステートメントと EXECUTE ステートメントの両方が実行されます。
 - 無効な ALTER パスワードを指定してコンパイル済み DATA ステッププログラムや DATA ステップビューを実行した場合、次のようになります。

DESCRIBE ステートメントは実行されません。

バッチモードでは、この EXECUTE ステートメントは何も影響しません。

対話型モードでは、READ パスワードの入力を求めるプロンプトが表示されます。READ パスワードが有効な場合、EXECUTE ステートメントが処理されます。READ パスワードが無効な場合、EXECUTE ステートメントは処理されません。

- READ パスワードを DESCRIBE ステートメントおよび EXECUTE ステートメントに使用すると、次のようになります。
 - 対話型モードでは、ALTER パスワードの入力を求めるプロンプトが表示されます。
 - 有効な ALTER パスワードを入力すると、DESCRIBE ステートメントと EXECUTE ステートメントの両方が実行されます。
 - 無効な ALTER パスワードを入力すると、EXECUTE ステートメントは実行されますが、DESCRIBE ステートメントは実行されません。
 - バッチモードでは、EXECUTE ステートメントは処理されますが、DESCRIBE ステートメントは処理されません。
 - 対話型モードでもバッチモードでも、無効な READ パスワードを指定すると、EXECUTE ステートメントは実行されません。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューに DESCRIBE ステートメントが含まれる場合には、ALTER パスワードが必要です。
- コンパイル済み DATA ステッププログラムまたは DATA ステップビューを置き換えるには、ALTER パスワードが必要です。

出力データセットを作成する(形式 1)

1 つまたは複数のデータセットを作成するには、DATA ステートメントを使用します。データセットオプションを使用すると、出力データセットをカスタマイズすることができます。

この DATA ステップでは、2 つの出力データセット EXAMPLE1 および EXAMPLE2 を作成します。この DATA ステップの例では、データセットオプション DROP を使用し、変数 IDNUMBER は EXAMPLE2 に書き込まれないようにします。

```
data example1 example2 (drop=IDnumber);
  set sample;
  ...more SAS statements. . .
run;
```

DATA ステップビューの作成(形式 2)

DATA ステップビューを先に作成して、後から実行することができます。DATA ステップの例では、DATA ステップビューが作成されます。また、SOURCE=ENCRYPT オプションを使用し、ソースコードの保存と暗号化を実行します。

```
data phone_list / view=phone_list (source=encrypt);
  set customer_list;
  ...more SAS statements. . .
run;
```

詳細については、“[SAS ビューの定義](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

コンパイル済み DATA ステッププログラムの作成(形式 3)

DATA ステッププログラムをコンパイルして保存できるため、このストアプログラムを後から実行できます。コンパイル済み DATA ステッププログラムにより、DATA ステッププログラムを繰り返しコンパイルする必要がなくなるため、処理コストを削減できます。この DATA ステップの例では、DATA ステッププログラムのコンパイルと保存を行います。ここでは、ALTER パスワードオプションを使用します。そのため、ユーザーは既存のストアプログラムを置き換えたり、コンパイル済みプログラムが置き換えられないように保護することができます。

```
data testfile / pgm=stored.test_program (alter=sales);
  set sales_data;
  ...more SAS statements. . .
run;
```

詳細については、“[Stored, Compiled DATA Step Programs](#)” ([SAS V9 LIBNAME Engine: Reference](#))を参照してください。

DATA ステップビューの記述(形式 4)

次の例では、DATA ステップビューで DESCRIBE ステートメントを使用し、ソースコードのコピーを SAS ログに書き込みます。

```
data view=inventory;
  describe;
run;
```

詳細については、“[DESCRIBE ステートメント](#)” ([86 ページ](#))を参照してください。

コンパイル済み DATA ステッププログラムの実行(形式 5)

この例では、コンパイル済みストアード DATA ステッププログラムを実行しています。ここでは、DESCRIBE ステートメントを使用し、ソースコードのコピーを SAS ログに書き込みます。

```
libname stored 'SAS library';
data pgm=stored.employee_list;
  describe;
  execute;
run;

libname stored 'SAS library';
data pgm=stored.test_program;
  describe;
  execute;
  ...more SAS statements...
run;
```

詳細については、“[DESCRIBE ステートメント](#)” (86 ページ)および“[EXECUTE ステートメント](#)” (103 ページ)を参照してください。

例

例 1: 複数データファイルの作成とデータセットオプションの使用

次の DATA ステートメントでは、複数のデータセットを作成してから、出力データセットの内容を変更します。

```
data error (keep=subject date weight)
  fitness(label='Exercise Study'
    rename=(weight=pounds));
```

ERROR データセットには 3 つの変数が含まれています。FITNESS データセットにラベルを割り当てた後、変数の名前を *weight* から *pounds* に変更します。

例 2: 入力 DATA ステップビューの作成

次のデータステップでは、SAS データファイルのかわりに、入力 DATA ステップビューを作成します。

```
libname ourlib 'SAS-library';
data ourlib.test / view=ourlib.test;
  set ourlib.fittest;
  tot=sum(of score1-score10);
run;
```

例 3: ビューおよびデータファイルの作成

次の DATA ステップでは、入力 DATA ステップビュー THEIRLIB.TEST を作成し、一時 SAS データセット SCORETOT を追加で作成します。

```
libname ourlib 'SAS-library-1';
```



```

libname theirlib 'SAS-library-2';
data theirlib.test scoretot
  / view=theirlib.test;
  set ourlib.fittest;
  tot=sum(of score1-score10);
run;

```

後続の DATA ステップまたは PROC ステップでビュー THEIRLIB.TEST が処理されるまで、データファイル SCORETOT は作成されません。

例 4: コンパイル済みプログラムの保存と実行

最初の DATA ステップでは、STORED.SALESFIG というコンパイル済みストアプログラムが作成されます。

```

libname in 'SAS-library-1';
libname stored 'SAS-library-2';
data salesdata / pgm=stored.salesfig;
  set in.sales;
  qtr1tot=jan+feb+mar;
run;

```

コンパイル済みプログラム STORED.SALESFIG を実行すると、データセット SALESDATA が作成されます。

```

data pgm=stored.salesfig;
run;

```

例 5: カスタムレポートの作成

次のプログラムの 2 番目の DATA ステップではカスタムレポートを生成します。また、_NULL_ キーワードが指定されているので、SAS データセットを作成せずに DATA ステップを実行します。

```

data sales;
  input dept : $10. jan feb mar;
  datalines;
shoes 4344 3555 2666
housewares 3777 4888 7999
appliances 53111 7122 41333
;
data _null_;
  set sales;
  qtr1tot=jan+feb+mar;
  put 'Total Quarterly Sales: '
    qtr1tot dollar12.;
run;

```

例 6: コンパイル済み DATA ステッププログラムにパスワードを使用する

次に示す最初の DATA ステップでは、STORED.ITEMS というコンパイル済み DATA ステッププログラムを作成します。また、このプログラムでは ALTER パスワードを使用しているので、プログラムへのアクセスが制限されます。

```

data sample;
  input Name $ TotalItems $;

```

```

        datalines;
Lin 328
Susan 433
Ken 156
Pat 340
;
proc print data=sample;
run;

libname stored 'SAS-library';

data employees / pgm=stored.items (alter=klondike);
    set sample;
    if TotalItems > 200 then output;
run;

```

この DATA ステップでは、コンパイル済み DATA ステッププログラム STORED.ITEMS を実行します。DESCRIBE ステートメントを使用し、ソースコードを SAS ログに出力します。このプログラムは ALTER パスワードを使用して作成されているので、DESCRIBE ステートメントを使用する場合はこのパスワードを使用する必要があります。パスワードを入力しないと、入力を求めるプロンプトが表示されます。

```

data pgm=stored.items (alter=klondike);
    describe;
    execute;
run;

```

例 7: ネストレベルの表示

このプログラムには、2 つのネストレベルが含まれています。4 つのログメッセージが作成されます。このメッセージは、各ネストレベルの開始時と終了時に 1 つずつ作成されます。

```

data _null_ /nesting;
do i = 1 to 10;
    do j = 1 to 5;
        put i=j;
    end;
end;
run;

```

例のコード 2.1 ネスティングレベルのデバッグ(部分的な SAS ログ)

```

6 data _null_ /nesting;
7   do i = 1 to 10;
8     -
      719
NOTE 719-185: *** DO begin level 1 ***.
8     do j = 1 to 5;
9       -
      719
NOTE 719-185: *** DO begin level 2 ***.
9       put i=j;
10      end;
    ---
    720
NOTE 720-185: *** DO end level 2 ***.
11    end;
    ---
    720
NOTE 720-185: *** DO end level 1 ***.
12 run;
```

関連項目

- [“DATA Step Programming in SAS LASR Analytic Server” - SAS LASR Analytic Server: Reference Guide](#)
- [“データセットオプションの定義” \(SAS データセットオプション: リファレンス\)](#)

ステートメント:

- [“DESCRIBE ステートメント” \(86 ページ\)](#)
- [“EXECUTE ステートメント” \(103 ページ\)](#)
- [“LINK ステートメント” \(242 ページ\)](#)

DATALINES ステートメント

ステートメントの後にデータ行が続くことを指定します。

該当要素: DATA ステップ

カテゴリ: ファイル操作

種類: 宣言

別名: CARDS
LINES

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。

このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

参照項目: データ行にセミコロンを含めることはできません。データにセミコロンが含まれている場合は、“[DATALINES4 ステートメント](#)” (84 ページ)を使用してください。

構文

DATALINES;

引数なし

外部ファイルに格納されているデータを読み込むのではなく、プログラムに直接入力するデータを読み込む場合は、DATALINES ステートメントを INPUT ステートメントとともに使用します。

詳細

DATALINES ステートメントの使用

DATALINES ステートメントは DATA ステップの最後にあるステートメントです。このステートメントのすぐ後に最初のデータ行が続きます。入力データの最後を示すには、null ステートメント(1 つのセミコロン)を使用します。

1 つの DATA ステップでは、DATALINES ステートメントを 1 つのみ使用できます。複数のデータのセットを入力する場合は、個々に DATA ステップを使用します。

長いデータ行の読み込み

データ行の長さを扱うには、[CARDIMAGE](#) システムオプションを使用します。CARDIMAGE オプションを使用すると、ブランクが埋め込まれた 80 バイトのパンチカードイメージのようにデータ行を処理します。NOCARDIMAGE を使用すると、全体で 80 列よりも長いデータ行を処理します。

インストリームデータでの入力オプションの使用

DATALINES ステートメントには、データの読み込みに使用できる入力オプションはありません。ただし、DATALINES ステートメントと INFILE ステートメントを組み合わせると、いくつかのオプションを使用できます。INFILE ステートメントに DATALINES ステートメントを指定してデータのソースを定義した後、必要なオプションを指定します。詳細については、“[例 2: オプションを使用したインストリームデータの読み込み](#)” (83 ページ)を参照してください。

比較

- データにセミコロンが含まれていない場合は、常に DATALINES ステートメントを使用します。データにセミコロンが含まれている場合は、DATALINES4 ステートメントを使用します。

- 次の SAS ステートメントを使用しても、データの読み込みやデータ格納先の指定を行えます。
 - INFILE ステートメントは、他のファイルに格納されている生データ行を指定します。INPUT ステートメントは、指定されたデータ行を読み込みます。
 - %INCLUDE ステートメントは、SAS プログラムステートメントや SAS ファイルまたは外部ファイルに格納されているデータ行を現在のプログラムに読み込みます。
 - SET、MERGE、MODIFY、UPDATE の各ステートメントは、既存の SAS データセットからオブザベーションを読み込みます。

例

例 1: DATALINES ステートメントの使用

この例では、DATA ステップのオブザベーションごとに、データ行を読み込んだ後、2つの文字変数 NAME と DEPT に値を割り当てます。

```
data person;  
  input name $ dept $;  
  datalines;  
John Sales  
Mary Acctng  
;
```

例 2: オプションを使用したインストリームデータの読み込み

この例では、インストリームデータを読み込む INFILE ステートメントで使用可能なオプションを利用します。DELIMITER=オプションを使用すると、リスト入力を使用して、ブランクではなくカンマで区切られたデータの値を読み込みます。

```
data person;  
  infile datalines delimiter=';';  
  input name $ dept $;  
  datalines;  
John,Sales  
Mary,Acctng  
;
```

関連項目

ステートメント:

- [“DATALINES4 ステートメント” \(84 ページ\)](#)
- [“INFILE ステートメント” \(143 ページ\)](#)

システムオプション:

- [“CARDIMAGE システムオプション” \(SAS システムオプション: リファレンス\)](#)

DATALINES4 ステートメント

ステートメントに続くデータ行に内部セミコロンが含まれるように指定します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	宣言
別名:	CARDS4 LINES4
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。 詳細については、 “DATA Step Acceleration” (SAS In-Database Products: User's Guide) を参照してください。

構文

DATALINES4;

引数なし

プログラムに直接入力するセミコロンを含むデータを読み込むには、DATALINES4 ステートメントを INPUT ステートメントと使用します。

詳細

DATALINES4 ステートメントは DATA ステップの最後にあるステートメントです。このステートメントのすぐ後に最初のデータ行が続きます。データ行の終わりには、第 1 列から第 4 列に配置される連続する 4 つのセミコロンを使用します。

比較

データにセミコロンが含まれている場合は、DATALINES4 ステートメントを使用します。データにセミコロンが含まれていない場合は、DATALINES ステートメントを使用します。

例: セミコロンが含まれるデータ行の読み込み

この例では、4つのセミコロンが検出されるまで、セミコロンが含まれるデータ行を読み込みます。プログラムは最後まで実行されます。

```
data biblio;  
  input number citation $50.;  
  datalines4;  
  KIRK, 1988  
  2 LIN ET AL., 1995; BRADY, 1993  
  3 BERG, 1990; ROA, 1994; WILLIAMS, 1992  
  ;;;
```

関連項目

ステートメント:

- ["DATALINES ステートメント" \(81 ページ\)](#)

DELETE ステートメント

現在のオブザベーションの処理を中止します。

該当要素:	DATA ステップ
カテゴリ:	アクション CAS In-Database 処理
種類:	実行

構文

DELETE;

引数なし

DELETE を実行すると現在のオブザベーションはデータセットに書き込まれません。次の繰り返しを実行するために、すぐに DATA ステップの先頭に戻ります。

詳細

DELETE ステートメントは、IF-THEN ステートメントの THEN 句または条件を指定して実行される DO グループの一部としてよく使用されます。

比較

- DELETE ステートメントは、データセットからオブザベーションを除外する条件を設定するほうが簡単な場合、または現在のオブザベーションに対して DATA ステップの処理を継続する必要がない場合に使用します。
- オブザベーションを含める条件を設定するほうが簡単な場合は、サブセット化 IF ステートメントを使用します。
- DROP ステートメントと DELETE ステートメントを混同しないでください。DROP ステートメントは出力データセットから変数を除外します。DELETE ステートメントはオブザベーションを除外します。

例

例 1: IF-THEN ステートメントの一部として DELETE ステートメントを使用する

LEAFWT の値が欠損している場合、現在のオブザベーションを削除します。

```
if leafwt=. then delete;
```

例 2: DELETE ステートメントを使用して生データをサブセット化する

```
data topsales;  
  infile file-specification;  
  input region office product yrsales;  
  if yrsales<100000 then delete;  
run;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“DROP ステートメント” \(98 ページ\)](#)
- [“IF ステートメント: サブセット化” \(137 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(140 ページ\)](#)

DESCRIBE ステートメント

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからソースコードを読み込みます。

該当要素:	DATA ステップ
カテゴリ:	アクション
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (SAS In-Database Products: User's Guide)を参照してください。 DESCRIBE はコンパイル済み DATA ステッププログラムおよび DATA ステップビューにのみ使用します。
要件	DATA ステートメントでは、PGM=オプションまたは VIEW=オプションを指定する必要があります。

構文

DESCRIBE;

引数なし

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからプログラムソースコードを読み込む場合は、DESCRIBE ステートメントを使用します。SAS はソースステートメントを SAS ログに書き込みます。

詳細

コンパイル済み DATA ステッププログラムまたは DATA ステップビューからプログラムソースコードを読み込む場合は、EXECUTE ステートメントを使用せずに DESCRIBE ステートメントを使用します。ソースコードを読み込んでからコンパイル済み DATA ステッププログラムを実行する場合は、DESCRIBE ステートメントと EXECUTE ステートメントを使用します。DATA ステートメントでこれらのステートメントを使用する方法の詳細については、“[DATA ステートメント](#)” (68 ページ)を参照してください。

注: パスワード保護されたビューまたは DATA ステッププログラムを DESCRIBE するには、パスワードを指定しなければなりません。ビューまたはプログラムが複数のパスワード付きで作成されている場合、一番厳格なパスワードを指定します。他の SAS ファイルと同様で、ALTER パスワードが一番厳格で、READ パスワードは最も制限が緩いです。詳細については、“[Stored, Compiled DATA Step Programs](#)” ([SAS V9 LIBNAME Engine: Reference](#))および“[ビュー付きのパスワード](#)” ([SAS プログラムガイド: エッセンシャル](#))を参照してください。

関連項目

ステートメント:

- [“DATA ステートメント” \(68 ページ\)](#)
- [“EXECUTE ステートメント” \(103 ページ\)](#)

DO ステートメント

1 単位として実行するステートメントのグループを指定します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行
参照項目:	DOLIST syntax (definition) “反復 DO ループの形式” (SAS プログラマガイド: エッセンシャル)

構文

DO;
...more SAS statements...

END;

引数なし

単純 DO グループの処理には、DO ステートメントを使用します。

詳細

DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。DO ステートメントと END ステートメントではさまれたステートメントは、*DO グループ*と呼ばれます。DO ステートメントは、DO グループ内でネストさせることができます。

DOLIST 構文を使って、値のリストを反復処理することもできます。参照: [“例 2: DOLIST 構文の使用” \(89 ページ\)](#)。

注: システムのメモリサイズによっては、使用可能な DO ステートメントのネスト数が制限される場合があります。詳細については、SAS ドキュメントのシステムメモ

リでサポートされる DO ステートメントのネストレベルについての説明を参照してください。

通常、単純 DO ステートメントは、IF 条件が真か偽かに基づいてステートメントのグループの実行を指定する IF-THEN/ELSE ステートメントの中で使用されます。

比較

DO ステートメントは、他に 3 種類あります。

- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。反復 DO ステートメントには WHILE 句または UNTIL 句を含めることができます。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。

例

例 1: 単一の DO ステートメントの使用

この単純 DO グループでは、YEARS の値が 5 を超えている場合にのみ DO と END には含まれたステートメントが実行されます。YEARS の値が 5 以下の場合、DO グループのステートメントは実行されません。プログラムは、ELSE ステートメントの後に続く割り当てステートメントを実行します。

```
data years;
  years=3;
  if years>5 then
  do;
    months=years*12;
    put years= months=;
  end;
  else yrsleft=5-years;
run;
```

例 2: DOLIST 構文の使用

この例では、DOLIST 構文を使用して値のリストを反復処理します。

```
data do_to;
  x=0;
  do i=0 to 10;
    x=x+1;
    output;
  end;
run;
```

```
proc print data=do_to;  
run;
```

アウトプット 2.8 DOLIST 構文出力を示す PROC PRINT 出力

Obs	x	i
1	-5	5
2	-5	5
3	-5	4
4	-5	5
5	-5	80
6	-5	85
7	-5	90
8	-5	60
9	-5	55
10	-5	50
11	-5	45
12	-5	40

関連項目

ステートメント:

- [“ループと条件の定義” \(SAS プログラマガイド: エッセンシャル\)](#)
- [“DO ステートメント: 反復” \(90 ページ\)](#)
- [“DO UNTIL ステートメント” \(95 ページ\)](#)
- [“DO WHILE ステートメント” \(97 ページ\)](#)

DO ステートメント: 反復

インデックス変数の値に基づいて、DO ステートメントと END ステートメントにはさまれている各ステートメントを実行します。

該当要素: DATA ステップ

カテゴリ: CAS
制御
In-Database 処理

種類: 実行

例:

```
data example;  
  do i=1 to 10;  
    put i=;  
  end;  
run;
```

構文

```
DO index-variable=specification-1<, ...specification-n>;
...more SAS statements...
```

```
END;
```

引数

index-variable

DO グループの実行を制御する値を持つ変数を指定します。

ヒント インデックス変数の削除を指定しない場合、作成されるデータセットにインデックス変数が含まれます。

注意 反復 DO グループ内でインデックス変数を変更すると、無限ループが発生する場合があります。

specification

1 つの式または複数の式を次の形式で指定します。

```
start <TO stop> <BY increment> <WHILE(expression) | UNTIL(expression)>
```

初回実行時、DO グループは、*index-variable* が *start* に等しいとして実行されます。*start* の値は、ループの初回実行前に評価されます。

start

インデックス変数の初期値を指定します。

TO *stop* または BY *increment* を指定せずに使用する場合、*start* の値は次の形式で指定されます。

```
item-1 <, ...item-n>;
```

item には、数値定数または文字定数を指定します。変数を指定する場合があります。文字定数は単一引用符で囲みます。リストに指定したそれぞれの値に対して DO グループが 1 度実行されます。WHILE 条件を追加する場合、この条件はそのすぐ前にある *item* にのみ適用されます。

要件 TO *stop* または BY *increment* を使用する場合、*start* には数値または数値を生成する式を指定する必要があります。

例 “例 1: さまざまな形式の反復 DO ステートメントを使用する” (93 ページ)

TO *stop*

インデックス変数の終了値を指定します。

start と *stop* の両方を指定する場合、*index-variable* の値が *stop* になるまで、*increment* の値に基づいてステートメントが実行されます。*start* と *increment* のみを指定する場合、ステートメントによってループの中止が指示されるか、または DO ステートメントで指定された WHILE 式または UNTIL 式を満たすまで、*increment* の値に基づいてステートメントが実行されます。*stop* も *increment* も指定されない場合、このグループは *start* の値に基づいて実行されます。*stop* の値は、ループの初回実行前に評価されます。

要件 *stop* の値は、数値または数値を生成する式を指定する必要があります。

ヒント DO グループでの *stop* の変更は、繰り返し回数に影響を与えることはありません。処理が終了する前にループの繰り返しを中止するには、*index-variable* の値を変更して *stop* の値を上回るようにするか、LEAVE ステートメントを使用してステートメントのループを中止します。

例 “例 1: さまざまな形式の反復 DO ステートメントを使用する” (93 ページ)

BY *increment*

index-variable の増分値を制御する正または負の数値(または数値を生成する式)を指定します。

increment の値はループを実行する前に評価されます。DO グループでの *increment* の変更は、繰り返し回数に影響を与えることはありません。*increment* を指定しない場合、インデックス変数は 1 つずつ増加します。*increment* が正の数値の場合、*start* の値を下限值にする必要があります。また、*stop* が存在する場合、この値はループの上限値にする必要があります。*increment* が負の数値の場合、*start* の値は上限値にする必要があります。また、*stop* が存在する場合、この値はループの下限值にする必要があります。

例 “例 1: さまざまな形式の反復 DO ステートメントを使用する” (93 ページ)

WHILE(*expression*)

UNTIL(*expression*)

DO グループの実行前または実行後に指定した SAS 式を評価します。式は丸かっこで囲みます。

WHILE 式はループの実行前に評価されるので、この式が真の間、グループ内のステートメントが繰り返し実行されます。UNTIL 式はループ実行後に評価されるので、この式が真になるまで、グループ内のステートメントが繰り返し実行されます。

制限事項 WHILE または UNTIL の指定は、WHILE または UNTIL が指定されている句の最後のアイテムにのみ影響を与えます。

参照項目: “DO WHILE ステートメント” (97 ページ)および“DO UNTIL ステートメント” (95 ページ)

例 “例 1: さまざまな形式の反復 DO ステートメントを使用する” (93 ページ)

要件 反復 DO ステートメントには、少なくとも 1 つの *specification* 引数が必要です。

ヒント オプションである TO 句と BY 句の位置は逆にすることができます。

複数の *specification* を使用する場合、それぞれの *specification* が実行前に評価されます。

比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。

例

例 1: さまざまな形式の反復 DO ステートメントを使用する

- 次の反復 DO ステートメントでは、*start* の値にアイテムのリストを使用します。

```
do month='JAN','FEB','MAR';
do count=2,3,5,7,11,13,17;
do i=5;
do i=var1, var2, var3;
do i='01JAN2001'd,'25FEB2001'd,'18APR2001'd;
```

- 次の反復 DO ステートメントでは、*start TO stop* 構文を使用します。

```
do i=1 to 10;
do i=1 to exit;
do i=1 to x-5;
do i=1 to k-1, k+1 to n;
do i=k+1 to n-1;
```

- 次の反復 DO ステートメントでは、BY *increment* 構文を使用します。

```
do i=n to 1 by -1;
do i=.1 to .9 by .1, 1 to 10 by 1,
  20 to 100 by 10;
do count=2 to 8 by 2;
```

- 次の反復 DO ステートメントでは、WHILE 句と UNTIL 句を使用します。

```
do i=1 to 10 while(x<y);
do i=2 to 20 by 2 until((x/3)>y);
do i=10 to 0 by -1 while(month='JAN');
```

- この例では、I=1 と I=2 のときに DO ループが実行されます。I=3 の場合に WHILE 条件が評価され、この条件が真の場合に DO ループが実行されます。

```
do i=1,2,3 while (condition);
```

ログ出力の完全な例を次に示します。

```
data do_list;
  x=2;
  do i=1, 2, 3 while (x=2);
    put i=; put x=;
  end;
run;
```

ログ出力:

```
i=1
x=2
i=2
x=2
i=3
x=2
```

```
data do_list;
  x=2;
  do i=1, 2, 3 while (x=3);
    put i=; put x=;
  end;
run;
```

ログ出力:

```
i=1
x=2
i=2
x=2
```

例 2: 無限ループを発生させずに反復 DO ステートメントを使用する

次のどちらの例でも DO グループを 10 回実行します。最初の例では推奨方法を示します。

```
/* correct coding */
do i=1 to 10;
  ...more SAS statements...
end;
```

次の例では、TO 引数および BY 引数を使用します。

```
do i=1 to n by m;
  ...more SAS statements...
  if i=10 then leave;
end;
if i=10 then put 'EXITED LOOP';
```

例 3: DO ループの実行を中止する

この例では、ループを終了させるために、インデックス変数の値を EXIT の現在値に設定します。

```
data iterate1;
  input x;
```



```
exit=10;
do i=1 to exit;
  y=x*normal(0);
  /* if y>25,      */
  /* changing i's value */
  /* stops execution */
  if y>25 then i=exit;
  output;
end;
datalines;
5
000
2500
;
```

関連項目

ステートメント:

- [“ARRAY ステートメント” \(40 ページ\)](#)
- [“配列参照 ステートメント” \(46 ページ\)](#)
- [“DO ステートメント” \(88 ページ\)](#)
- [“DO UNTIL ステートメント” \(95 ページ\)](#)
- [“DO WHILE ステートメント” \(97 ページ\)](#)
- [“GOTO ステートメント” \(135 ページ\)](#)

DO UNTIL ステートメント

条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

```
DO UNTIL (expression);
...more SAS statements...

END;
```

引数

(*expression*)

SAS 式を丸かっこで囲んで指定します。少なくとも 1 つの *expression* を指定する必要があります。

詳細

式は、DO ループのステートメントが実行されたループの最後に評価されます。この式が真になると、DO ループは実行されません。

注: 少なくとも 1 度は DO ループが実行されることになります。

比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントにはさまれている各ステートメントを実行します。
- DO WHILE ステートメントは、DO ループを繰り返す前に条件を毎回確認して、条件が真の間だけ DO ループ内のステートメントを実行します。DO UNTIL ステートメントではループの最後に条件を評価するのに対し、DO WHILE ステートメントではループの最初に条件を評価します。

注: DO UNTIL ループのステートメントは少なくとも 1 度は実行されますが、DO WHILE ループのステートメントは条件が偽の場合には 1 度も実行されません。

例: DO UNTIL ステートメントを使用したループの繰り返し

次のステートメントでは、N の値が 5 以上になるまでループが繰り返されます。式 $N \geq 5$ はループの最後に評価されます。(N の値が 0、1、2、3、4 の)全部で 5 回繰り返しが実行されます。

```
n=0;
do until(n>=5);
  put n=;
  n+1;
end;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“DO ステートメント: 反復” \(90 ページ\)](#)
- [“DO WHILE ステートメント” \(97 ページ\)](#)

DO WHILE ステートメント

条件が真の間、DO ループ内のステートメントを繰り返し実行します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

```
DO WHILE (expression);  
...more SAS statements...
```

```
END;
```

引数

(*expression*)

SAS 式を丸かっこで囲んで指定します。少なくとも 1 つの *expression* を指定する必要があります。

詳細

式は、ループの最初に DO ループのステートメントの実行前に評価されます。この式が真の場合、DO ループは繰り返し実行されます。この式が最初の評価時に偽となる場合、DO ループは 1 度も実行されません。

比較

DO ステートメントは、他に 3 種類あります。

- DO ステートメントは、DO グループの処理を指定するもっとも簡単な形式です。通常、IF-THEN/ELSE ステートメントの中で使用され、1 単位として実行するステートメントのグループを指定します。
- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントには含まれている各ステートメントを実行します。
- DO UNTIL ステートメントは、DO ループを繰り返した後に条件を毎回確認して、条件が真になるまで DO ループ内のステートメントを実行します。DO WHILE ステートメントではループの最初に条件を評価するのに対し、DO UNTIL ステートメントではループの最後に条件を評価します。

注: 式の評価結果が偽の場合、DO WHILE ループのステートメントは実行されません。ただし、DO UNTIL の式はループの最後に評価されるため、DO UNTIL ループのステートメントは少なくとも 1 度実行されます。

例: DO WHILE ステートメントの使用

次のステートメントでは、N の値が 5 未満の間ループが繰り返されます。式 $N < 5$ はループの最初に評価されます。(N の値が 0、1、2、3、4 の)全部で 5 回繰り返しが実行されます。

```
n=0;
do while(n<5);
  put n=;
  n+1;
end;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“DO ステートメント: 反復” \(90 ページ\)](#)
- [“DO UNTIL ステートメント” \(95 ページ\)](#)

DROP ステートメント

出力 SAS データセットから変数を除外します。

該当要素: DATA ステップ
 カテゴリ: CAS
 情報
 In-Database 処理

種類: 宣言

構文

DROP *variable-list*;

引数

variable-list

出力データセットから除外する変数の名前を指定します。

ヒント SAS で使用可能な任意の形式で変数をリストできます。

詳細

DROP ステートメントは、同一の DATA ステップ内で作成されたすべての SAS データセットに適用されます。また、ステップのどの位置に指定してもかまいません。DROP ステートメントの変数は DATA ステップでの処理が可能です。DROP または KEEP ステートメントが存在しない場合、DATA ステップで作成されたすべてのデータセットにすべての変数が含まれます。同一の DATA ステップで DROP ステートメントと KEEP ステートメントの両方を使用しないでください。

同じ変数が DROP ステートメントと KEEP ステートメントの両方にリストされている場合、ステートメントの順序に関係なく、DROP が KEEP よりも優先され、変数は削除されます。

比較

- DROP ステートメントと DROP=データセットオプションの違いは次の点になります。
 - SAS プロシジャステップでは、DROP ステートメントを使用できません。
 - DROP ステートメントは DATA ステートメントで指定したすべての出力データセットに適用されます。一部のデータセットから変数を除外し、その他のデータセットから除外しない場合は、DATA ステートメントで DROP=データセットオプションを使用します。
- KEEP ステートメントは、出力データセットへ書き込む変数のリストを指定する同種のステートメントです。含める変数が除外する変数よりも著しく少ない場合、DROP ステートメントのかわりに KEEP ステートメントを使用します。
- DROP ステートメントと DELETE ステートメントを混同しないでください。DROP ステートメントは出力データセットから変数を除外します。DELETE ステートメントはオブザベーションを除外します。

例

例 1: DROP ステートメントの基本的な使用法

これらの例では、DROP ステートメントで変数をリストする場合の正しい構文を示します。

```
drop time shift batchnum;  
drop grade1-grade20;
```

例 2: 出力データセットから変数を除外する

この例では、変数 PURCHASE と REPAIR が処理で使用されていますが、出力データセットである INVENTORY には書き込まれません。

```
data inventory;  
  drop purchase repair;  
  infile file-specification;  
  input unit part purchase repair;  
  totcost=sum(purchase,repair);  
run;
```

関連項目

データセットオプション:

- [“DROP= データセットオプション” \(SAS データセットオプション: リファレンス\)](#)

ステートメント:

- [“DELETE ステートメント” \(85 ページ\)](#)
- [“KEEP ステートメント” \(228 ページ\)](#)

END ステートメント

DO グループまたは SELECT グループの処理を終了します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	宣言

構文

END;

引数なし

DO グループまたは SELECT グループの処理を終了するには END を使用します。

詳細

END ステートメントは、DO グループまたは SELECT グループ内の最後のステートメントとして指定する必要があります。

例: END ステートメントの使用

この例では、END ステートメントを使って単純な DO グループを終了させる方法を紹介します。

```
do;  
    ...more SAS statements...  
end;
```

この例では、END ステートメントを使って単純な SELECT グループを終了させる方法を紹介します。

```
select(expression);  
    when(expression) SAS statement;  
    otherwise SAS statement;  
end;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“SELECT ステートメント” \(352 ページ\)](#)

ENDSAS ステートメント

SAS プログラムの中でこのステートメントが出てきた時点で、SAS プログラムの実行を停止し、SAS セッションを終了します。

注: [ENDSAS ステートメント](#)は SAS グローバルステートメントに移動しました。

ERROR ステートメント

`_ERROR_` を 1 に設定します。SAS ログに書き込まれるメッセージはオプションです。

該当要素:	DATA ステップ
カテゴリ:	アクション CAS In-Database 処理
種類:	実行

構文

ERROR <*message*>;

引数なし

引数を指定せずに ERROR を使用する場合は、自動変数 `_ERROR_` が 1 に設定され、ブランクのメッセージが SAS ログに書き込まれます。

引数

message

SAS ログにメッセージを書き込みます。

ヒント *message* には、文字リテラル(引用符で囲む)、変数名、出力形式、ポインターコントロールを指定できます。

詳細

ERROR ステートメントは自動変数 `_ERROR_` を 1 に設定します。SAS ログへの指定したメッセージの書き込みはオプションです。`_ERROR_ =1` の場合、現在のオブザベーションに対応するデータ行を SAS ログに書き込みます。

ERROR ステートメントは、次のステートメントの組み合わせに相当します。

- `_ERROR_` を 1 に設定する割り当てステートメント
- FILE LOG ステートメント
- PUT ステートメント(メッセージを指定する場合)
- PUT; ステートメント(メッセージを指定しない場合)
- FILE をあらかじめ指定した設定にリセットする別の FILE ステートメント

例: エラーメッセージの書き込み

IF-THEN ステートメントの条件を満たすオブザベーションごとに、エラーメッセージ、変数名、値を SAS ログに書き込みます。

- この例では、ERROR ステートメントは FILE ステートメントの指定をあらかじめ指定した設定に自動的にリセットします。

```
file file-specification;
  if type='teen' & age > 19 then
    error 'type and age don't match ' age=;
```

- この例では、ERROR ステートメントのかわりに、複数のステートメントを使用して同じ結果を出します。

```
file file-specification;
  if type='teen' & age > 19 then
    do;
      file log;
      put 'type and age don't match ' age=;
      _error_=1;
      file file-specification;
    end;
```

関連項目

ステートメント:

- [“PUT ステートメント” \(289 ページ\)](#)

EXECUTE ステートメント

コンパイル済み DATA ステッププログラムを実行します。

該当要素:	DATA ステップ
カテゴリ:	アクション
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (SAS In-Database Products: User's Guide)を参照してください。 EXECUTE はコンパイル済み DATA ステッププログラムのみで使用します。
要件	DATA ステップ内で PGM=オプションを指定します。

構文

EXECUTE;

引数なし

EXECUTE ステートメントは、コンパイル済み DATA ステッププログラムを実行します。

詳細

ソースコードを読み込んでからコンパイル済み DATA ステッププログラムを実行する場合は、同じ DATA ステップ内で EXECUTE ステートメントと DESCRIBE ステートメントを使用します。どちらのステートメントであるかを指定しない場合、EXECUTE ステートメントであるとみなされます。ステートメントを使用する順番は入れ替えることができます。DATA ステッププログラムは、ステップ境界に到達すると実行されます。DATA ステートメントでこれらのステートメントを使用する方法の詳細については、“[DATA ステートメント](#)” (68 ページ)を参照してください。

関連項目

ステートメント:

- “[DATA ステートメント](#)” (68 ページ)
- “[DESCRIBE ステートメント](#)” (86 ページ)

FILE ステートメント

PUT ステートメントの現在の出力ファイルを指定します。

該当要素: DATA ステップ

カテゴリ: CAS

ファイル操作

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“[DATA Step Acceleration](#)” (*SAS In-Database Products: User's Guide*)を参照してください。

SAS がロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、FILENAME ステートメントを使用できません。詳細については、“[LOCKDOWN](#)” (*SAS プログラマガイド: エッセンシャル*)を参照してください。

構文

FILE *file-specification* <*device-type*> <*options*>;

引数

file-specification

DATA ステップの PUT ステートメントの出力先に使用する外部ファイルを指定します。*file-specification* には、次の形式を指定できます。

'*external-file*'

外部ファイルの物理名を引用符で囲んで指定します。物理名には動作環境でファイルを判別できる名前を指定します。

fileref

外部ファイルのファイル参照名を指定します。

範囲 1 バイトから 8 バイトまで

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

要件 *fileref* は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。実行時に *fileref* を割り当てる唯一の方法は、FILE ステートメントで FILEVAR=オプションを使用することです。

注 FILEREF が EMAIL アクセスメソッドに関連付けられている場合、FILE ステートメントで追加オプションが利用できます。使用可能なオプションの完全なリストについては、“[FILENAME ステートメント: EMAIL\(SMTP\)アクセスメソッド](#)” (129 ページ)を参照してください。

参照 “[FILENAME ステートメント](#)” (128 ページ)
項目:

fileref(file)

ファイルの集約記憶域を表す外部ファイルに事前に割り当てられているファイル参照名を指定します。ファイル参照名の後ろに続けて、対応する集約記憶域に含まれているファイル名またはメンバー名をカッコで囲んで指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

要件 *fileref* は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。

動作環境 ディレクトリまたは区分データセットなど複数のファイルをまとめて保存する集約記憶域の名前は動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

注 集約記憶域に格納されているファイルの名前が有効な SAS 名ではない場合は、その名前を引用符で囲む必要があります。

参照項 [“FILENAME ステートメント”\(128 ページ\)](#)

目:

LOG

予約済みのファイル参照名です。PUT ステートメントで生成される出力を SAS ログに表示するように指示します。

DATA ステップの実行を開始するたびに、PUT ステートメントの出力先を指定するファイル参照名を自動的に LOG に設定します。したがって、DATA ステップの最初の PUT ステートメントは、PUT ステートメントの前に別の方法で指定する FILE ステートメントがない限り、常に SAS ログに書き込みます。

ヒント デフォルトでは、出力行は SAS ログに書き込まれます。そのため、FILE LOG ステートメントを使用するのは、変更した出力先をデフォルトの設定へ戻す場合、他の FILE ステートメントオプションを指定する場合です。

PRINT

予約済みのファイル参照名です。PUT ステートメントで生成される出力を、SAS プロシジャで生成される出力と同じファイルに出力するように指示します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

操作 ファイルへ出力する場合、N=オプションの値は 1 または PAGESIZE とする必要があります。

動作環境 ファイルに出力されるキャリッジコントロール文字は、動作環境によって異なる場合があります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

ヒント ファイル参照名に PRINT を指定した場合、キャリッジコントロール文字を使用して、出力ファイルの特性にあった出力が行われます。

参照項 ファイル出力の詳細に関する説明 - [SAS プログラマガイド: エッセンシャル](#)

目:

ヒント *file-specification* に指定したディレクトリにファイルが存在しない場合、ファイルが作成されます。*file-specification* に指定したディレクトリが存在しない場合、SYSERR マクロ変数が設定されます。この変数は、ERRORCHECK オプションが STRICT に設定されているかどうかを調べます。

device-type

デバイスの種類またはアクセスメソッドを指定します。これは、ファイル参照名が入出力デバイスや物理ファイルではない場所を参照している場合に使用されます。

CATALOG

CATALOG アクセスメソッドを指定します。

制限事項	この引数は、CAS で実行する DATA ステップではサポートされていません。
操作	DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。
参照項目:	CATALOG アクセスメソッドで指定可能なオプションの一覧については、“ FILENAME ステートメント: CATALOG アクセスメソッド ” (128 ページ)を参照してください。

CLIPBOARD

CLIPBOARD アクセスメソッドを指定します。

制限事項	この引数は、CAS で実行する DATA ステップではサポートされていません。
操作	DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。
参照項目:	CLIPBOARD アクセスメソッドで指定可能なオプションの一覧については、“ FILENAME ステートメント: CLIPBOARD アクセスメソッド ” (128 ページ)を参照してください。

DISK

デバイスがディスクドライブであることを指定します。

制限事項	この引数は、CAS で実行する DATA ステップではサポートされていません。
ヒント	ディスク上のファイルにファイル参照名を割り当てる場合、DISK を指定する必要はありません。

DUMMY

ファイルへの出力を破棄することを指定します。

制限事項	この引数は、CAS で実行する DATA ステップではサポートされていません。
操作	FILEVAR=オプションと DUMMY を併用した場合、ファイルは FILEVAR=オプションに記載されている名前には書き込まれません。
ヒント	テストを実行する場合は DUMMY を指定すると便利です。

FTP

FTP アクセスメソッドを指定します。

制限事項	この引数は、CAS で実行する DATA ステップではサポートされていません。
操作	DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照 FTP アクセスメソッドで指定可能なオプションの一覧については、
 項 “FILENAME ステートメント: FTP アクセスメソッド” (129 ページ)を参
 照してください。

例 infile dummy ftp user='myuid' pass='xxxx' filevar=file_to_read;

HADOOP

Hadoop アクセスメソッドを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場
 合、DATA ステップはこのオプションをアクセスメソッドに渡して処
 理します。

参照項 Hadoop アクセスメソッドで指定可能なオプションの一覧について
 目: は、“FILENAME ステートメント: Hadoop アクセスメソッド” (130 ペ
 ージ)を参照してください。

GTERM

出力デバイスの種類がグラフィックデータを受信するグラフィックデバイス
 であると指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

PIPE

名前の付いていないパイプを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

動作環境 一部の動作環境では、パイプはサポートされていません。

PLOTTER

バッファーされていないグラフィック出力デバイスを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

PRINTER

プリンターまたはプリンタースプールファイルを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

SOCKET

SOCKET アクセスメソッドを指定します。

制限事項 この引数は、CAS で実行する DATA ステップではサポートされてい
 ません。

- 操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。
- 参照項目: SOCKET アクセスメソッドで指定可能なオプションの一覧については、“[FILENAME ステートメント: SOCKET アクセスメソッド](#)”(130 ページ)を参照してください。

TAPE

テープドライブを指定します。

- 制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

TERMINAL

ユーザーの端末を指定します。

- 制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。

UPRINTER

ユニバーサルプリンターの定義名を指定します。

- 制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。
- ヒント FILENAME ステートメントにプリンター名を指定しない場合、PRINTERPATH オプションによって、使用するユニバーサルプリンターと出力先が制御されます。

WEBDAV

WEBDAV アクセスメソッドを指定します。

- 制限事項 この引数は、CAS で実行する DATA ステップではサポートされていません。
- 操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。
- 参照項目: WEBDAV アクセスメソッドで指定可能なオプションの一覧については、“[FILENAME ステートメント: WebDAV アクセスメソッド](#)”(130 ページ)を参照してください。

別名 DEVICE=*device-type*

デフォルト DISK

要件 *device-type* または DEVICE=*device-type* は、ステートメントの *file-specification* の直後に記述する必要があります。

動作環境 指定するデバイスによっては、さらに情報を指定することが必要になる場合があります。DISK 以外の値を指定する前に、各動作環境向けの

SAS ドキュメントを参照してください。動作環境によっては、ここで説明した値の他に使用できる値が存在する場合があります。

オプション

BLKSIZE=*block-size*

出力ファイルのブロックサイズを指定します。ブロックサイズは、1 回の出力操作で物理的に書き込まれるバイト数です。

別名 BLK=

デフォルト 8K

範囲 最大値は 1G-1 です。

COLUMN=*variable*

変数名を指定します。指定した変数の値は、ポインターの現在の列位置を示す値に設定されます。この変数は、自動変数と同じように、データセットに書き込まれません。

別名 COL=

参照項目: [LINE= \(115 ページ\)](#)

DELIMITER= *delimiter(s)*

リスト出力の *delimiter* に使用する、ブランクのかわりの区切り文字を指定します。使用できる区切り文字は次の通りです。

'list-of-delimiting-characters'

区切り文字として出力する文字を 1 つまたは複数指定します。

要件 指定する区切り文字のリストは引用符で囲む必要があります。

character-variable

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

別名 DLM=

デフォルト ブランク
ト

制限事項 文字列や文字変数を指定することはできますが、区切り文字として使用されるのは、指定した文字列や変数の最初の 1 文字だけです。FILE DLM=の処理は、INFILE DELIMITER=の処理とは異なります。

操作 出力に区切り文字を含める場合は、区切り文字を区別する DSD オプションを指定する必要があります。

ヒント DELIMITER=オプションはコロン(:)修飾子と併用できます(修飾リスト出力)。

区切り文字では大文字と小文字が区別されます。

参照項目: [“DLMSTR= delimiter” \(111 ページ\)](#)および[“DSD \(delimiter sensitive data\)” \(112 ページ\)](#)

DLMSTOPT='T' | t'

DLMSTR=T オプションの解析オプションを指定します。指定すると、区切り文字の末尾にある空白を削除します。

要件 DLMSTOPT=T オプションは、DLMSTR=オプションを使用している場合にのみ有効です。

ヒント DLMSTOPT=T オプションは、区切り文字として変数を使用する場合に便利です。

参照項目: [DLMSTR= \(111 ページ\)](#)

DLMSTR= delimiter

LIST 出力で空白のかわりに使用する区切り文字を指定します。使用できる *delimiter* は次のとおりです。

'delimiting-string'

区切り文字として出力する文字列を指定します。

要件 文字列は引用符で囲みます。

character-variable

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

デフォルト
空白

操作 FILE ステートメントに DLMSTR=オプションを複数指定すると、最後に指定した DLMSTR=オプションが使用されます。DELIMITER=オプションと DLMSTR=オプションの両方を指定すると、最後に指定したオプションが使用されます。

RECFM=N を指定する場合は、サイズが大きい入力アイテムでも十分に保持できる値が LRECL=オプションに指定されていることを確認してください。指定した値が十分ではない場合、区切り文字によってレコード境界間で分割される場合があります。

参照項目: [DELIMITER= \(110 ページ\)](#)、[DLMSTOPT= \(111 ページ\)](#)、および [DSD \(112 ページ\)](#)

DROPOVER

FILE ステートメントの LINESIZE=オプションまたは LRECL=オプションに指定した出力行の長さを超えるデータを切り捨てます。

デフォルトでは、現在の行の長さを超えるデータは、新しい行に書き込まれます。一方、DROPOVER を指定すると、現在の行に新しいデータアイテムを書き出すスペースがなければ、そのデータアイテム全体が切り捨てられます(または無視されます)。アイテム全体が切り捨てられる場合、列ポインターは、現在の行の最後の値を書き出した位置に留まります。十分なスペースが残っているか、列ポインターの位置を元に戻すと、PUT ステートメントで別のアイテムを現在の行に

書き出せます。データアイテムが切り捨てられても、DATA ステップの実行は正常に続行されます(自動変数_ERROR_=0)。DATA ステップの最後に、どのデータが切り捨てられたかを示すメッセージが各ファイルに対して出力されます。

デフォルト FLOWOVER

ヒント DROPOVER オプションは、PUT ステートメントで現在指定されている行の長さを超えて書き込みを行おうとした場合に、現在の長さを超えたデータアイテムを新しい行に出力しない場合に使用します。

参照項目: [“FLOWOVER” \(114 ページ\)](#)および[“STOPOVER” \(119 ページ\)](#)

DSD (delimiter sensitive data)

タブやカンマなどの区切り文字を含むデータ値を引用符で囲むように指定します。DSD オプションでは、区切り文字を含むデータ値をリスト出力に書き込むことができます。このオプションは他の種類の出力(例:フォーマット出力、列出力、名前付き出力)では無視されます。データ値に含まれる二重引用符は 2 回繰り返されます。変数の値に区切り文字が含まれ、FILE ステートメントに DSD を指定する場合、変数の値は出力が生成されるときに二重引用符で囲まれます。次に例を示します。

```
data _null_;
  file log dsd;
  x="lions, tigers, and bears";
  put x ' "Oh, my!";
run;
```

次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。

```
""lions, tigers, and bears"" , "Oh, my!"
```

引用符で囲まれている(テキスト)文字列に区切り文字が含まれ、FILE ステートメントに DSD が指定されている場合、引用符を含む文字列は PUT ステートメントを実行するときに二重引用符で囲まれません。次に例を示します。

```
data _null_;
  file log dsd;
  put 'lions, tigers, and bears';
run;
```

次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。

```
lions, tigers, and bears
```

操作 DSD を指定する場合、デフォルトの区切り文字はカンマ(,)です。別の区切り文字を使用する場合は、DELIMITER=オプションまたは DLMSTR=オプションを指定します。

ヒント デフォルトでは、指定した区切り文字を含まないデータ値は引用符で囲まれません。ただし、チルダ(~)修飾子を使用すると、欠損値を含むすべてのデータ値が引用符で囲まれます。その場合、区切り文字を含まないデータ値も引用符で囲まれます。

参照項目: [DELIMITER= \(110 ページ\)](#)および [DLMSTR= \(111 ページ\)](#)

ENCODING='encoding-value'

出力ファイルへの書き込み時に使用するエンコーディングを指定します。

ENCODING=の値は、出力ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

出力ファイルにデータを書き込む場合は、セッションエンコーディングから指定したエンコーディングにデータがトランスコードされます。

デフォルト SAS は、現在のセッションエンコーディングを使用します。

参照項目: [“SAS 言語要素中のエンコーディング値” \(SAS 各国語サポート\(NLS\): リファレンスガイド\)](#)

例 [“例 8: 出力ファイル書き込み時のエンコードの指定” \(126 ページ\)](#)

FILENAME=variable

文字変数の名前を指定します。指定した変数の値には、PUT ステートメントの出力で現在開いているファイルの物理名が設定されます。物理名とは、動作環境がどのようにファイルを認識するかということです。

ヒント この変数は、自動変数と同じように、データセットに書き込まれません。

文字変数のデフォルトの長さは 8 文字です。変数の長さが 8 バイトよりも長い場合、LENGTH ステートメントを使用して物理ファイル名の値を格納できるように十分な長さに設定してください。

参照項目 [FILEVAR= \(113 ページ\)](#)

目:

例 [“例 4: 現在の出力ファイルの識別” \(124 ページ\)](#)

FILEVAR=variable

変数の名前を指定します。この変数の値が変化すると、FILE ステートメントは現在の出力ファイルを閉じます。FILE ステートメントを次に実行するときに、新しいファイルを開きます。次の PUT ステートメントの出力先は、FILEVAR=オプションの変数の値に指定した新しいファイルになります。

制限事項 FILEVAR=変数の値は、物理ファイル名を含む文字列として表されます。

操作 FILEVAR=オプションを使用すると、*file-specification* は、実際のファイル名またはファイルに対して事前に割り当てられたファイル参照名ではなく、プレースホルダーになります。このプレースホルダーを使用して、処理情報を SAS ログに出力します。プレースホルダーには、ファイル参照名と同じルールが適用されます。

ヒント この変数は、自動変数と同じように、データセットに書き込まれません。

文字変数のデフォルトの長さは 8 文字です。8 文字を超える物理ファイル名がある場合、LENGTH ステートメントや INPUT ステートメントなどの他のステートメントを使用して、FILEVAR=オプションの変数に十分な長さを割り当ててください。

参照 [FILENAME= \(113 ページ\)](#)

項目:

例 [“例 5: 現在の出力ファイルを動的に変更する” \(125 ページ\)](#)

FLOWOVER

現在の行の長さを超えるデータは、新しい行に書き込まれます。指定した行の最大長(FILE ステートメントの LINESIZE=オプションに指定)を超える書き込みを PUT ステートメントが行う場合、現在の出力行をファイルに出力した後、現在の行の長さを超えるデータアイテムは新しい行に書き込まれます。

デフォルト

ルト

操作 PUT ステートメントに後置@が含まれる場合、ポインターは新しい行のデータアイテムを出力した後の位置で停止します。そのため、次の PUT ステートメントは、同じ行に書き込むことができます。このプロセスは、入力データの最後に到達するか、または後置@を指定しない PUT ステートメントによって現在の行がファイルに書き込まれるまで続きます。

参照項 [“DROPOVER” \(111 ページ\)](#)および[“STOPOVER” \(119 ページ\)](#)

目:

FOOTNOTES | NOFOOTNOTES

現在定義されているフットノートを出力するかどうかを制御します。

デフォルト

ト

要件 DATA ステップで出力するレポートにフットノートを出力するには、FILE ステートメントに FOOTNOTE オプションを指定する必要があります。

HEADER=label

ステートメントラベルを定義します。このラベルは、新しいページへの出力を開始するたびに実行する SAS ステートメントのグループを特定します。

制限事項 ラベルの後ろに記述する最初のステートメントは、実行ステートメントにする必要があります。それ以降は、任意のステートメントを使用できます。

HEADER=オプションは、出力ファイルに書き込む場合や PRINT=オプションを含める場合にのみ使用します。

ヒント このグループのステートメントが DATA ステップの反復のたびに実行されるのを防ぐには、RETURN ステートメントを 2 つ使用します。1 つはラベルの前に記述し、もう 1 つはグループの最後のステートメントとして記述します。

例 [“例 1: 新規ページの開始時にステートメントを実行する” \(123 ページ\)](#)

LINE=variable

変数の名前を指定します。ここに指定した変数の値には、出力ポインターを移動できる行範囲での現在の相対行番号が設定されます。変数名を指定すると、値が自動的に割り当てられます。

範囲 1 から N=オプションまたは#n 行ポインターコントロールに指定した値。どちらの値も指定されていない場合、LINE=変数の値は 1 になります。

ヒント この変数は、自動変数と同じように、データセットに書き込まれません。

LINE=に指定した変数の値は、PUT ステートメントの実行の最後に、次に出力できる行の番号に対して設定されます。

LINESIZE=line-size

レポートの 1 行あたりの最大列数、およびデータファイルの最大レコード長を設定します。

別名 LS=

デフォルト LINESIZE=のデフォルト値は、次の 2 つのオプションのいずれかで決まります。1) キャリッジコントロール文字を含むファイルまたは SAS ログに書き込む場合は、LINESIZE=システムオプションによってデフォルト値が決まります。または 2) ファイルに書き込む場合は、FILE ステートメントの LRECL=オプションによってデフォルト値が決まります。

範囲 最小値は 64 です。最大値は動作環境で許容されている最大論理レコード長です。

操作 LINESIZE=オプションで指定した行の長さを超えるデータの出力を PUT ステートメントが要求した場合の措置は、FLOWOVER オプション、DROPOVER オプション、STOPOVER オプションのうち、どのオプションが指定されているかによって異なります。デフォルト(FLOWOVER)では、データを複数の行に分けて書き出します。

動作環境 LINESIZE=オプションで許容される最大値は、動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

注 LINESIZE=オプションには、行のうちデータの出力に使用する長さを指定します。LRECL=オプションには、ファイルの物理的な長さを指定します。

参照項目: [LRECL= \(116 ページ\)](#)、["DROPOVER" \(111 ページ\)](#)、["FLOWOVER" \(114 ページ\)](#)、および["STOPOVER" \(119 ページ\)](#)

例 ["例 6: 出力行が出力ファイルの行の長さを超える場合" \(125 ページ\)](#)

LINESLEFT=variable

現在のページの残りの行数を格納する変数の名前を指定します。指定した変数には、現在のページの残りの行数の値が自動的に割り当てられます。

LINESLEFT=変数の値は、PUT ステートメントの実行終了時に設定されます。

別名 LL=

ヒント この変数は、自動変数と同じように、データセットに書き込まれません。

例 “例 2: 現在のページに残っている行数に基づき新規ページを決定する”
(123 ページ)

LRECL=*logical-record-length*

出力ファイルの論理レコード長を指定します。

デフォルト LRECL のデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。最大レコード長は 1G です。

操作 かわりに、[LRECL システムオプション](#)を使用すると、グローバルな論理レコード長を指定できます。

RECFM=F の場合、LRECL=の値により各出力レコードの長さが決定されます。出力レコードは、切り捨てられるか、指定されたサイズに合うように空白が追加されます。RECFM=N の場合、LRECL=の値は 256 以上にする必要があります。RECFM=V の場合、LRECL=の値により最大レコード長が決定されます。指定された長さを超えるレコードは、複数のレコードに分割されます。

注 LINESIZE=オプションには使用するレコードの長さを指定します。
LRECL=オプションには出力ファイルの物理的な長さを指定します。

参照 [LINESIZE= \(115 ページ\)](#)、[PAD \(117 ページ\)](#)、[PAGESIZE= \(118 ページ\)](#)、
項目: および [RECFM= \(118 ページ\)](#)

MEMVAR=*variable*

開くファイルを指定します。ディレクトリ fileref や物理名が FILE ステートメントで指定されると、MEMVAR=変数で参照されているファイルが開かれます。MEMVAR=変数の値が変わると、現在のファイルが閉じられ、新しいファイルが開かれます。

制限事項 MEMVAR=変数の値は、物理ファイル名を含む文字列として表されます。

ヒント この変数は、自動変数と同じように、データセットに書き込まれません。

文字変数のデフォルトの長さは 8 文字です。8 文字を超える物理ファイル名がある場合、LENGTH ステートメントや INPUT ステートメントなどの他のステートメントを使用して、MEMVAR=オプションの変数に十分な長さを割り当ててください。

参照項目: [FILENAME= \(113 ページ\)](#)

MOD

ファイル内の既存の行の後に出力行を書き込みます。

デフォルト OLD

制限事項 リスト出力以外の ODS 出力先に MOD オプションを使用しないでください。使用すると、予期しない結果が出力される場合があります。

参照項目: [“NEW|OLD” \(117 ページ\)](#)

N=available-lines

DATA ステップの現在の反復で出力ポインターの移動に使用できる行数を指定します。*available-lines* には、数値(*n*)を指定するか、またはキーワード PAGESIZE や PS を指定できます。

n

出力ポインターの移動に使用できる行数を指定します。指定した行数の範囲内で出力中にポインターを前後に動かしてから、ポインターを次の範囲に移動します。

PAGESIZE

このキーワードを指定すると、出力ポインターをページ全体で移動できます。

別名 PS

制限 事項 N=PAGESIZE の指定は、出力を表示する場合にのみ有効です。

現在のファイルが出力先のファイルの場合、*available-lines* には、1 または PAGESIZE のどちらかの値を指定する必要があります。

操作 出力ポインターの移動に使用できる行数を制御する場合、N=オプション使用するだけでなく、PUT ステートメントで#*n* 行ポインターコントロールを使用することもできます。

N=オプションの指定を省略し、行ポインターコントロール#を使用しない場合、移動できる範囲は 1 行です。つまり、デフォルトでは N=1 に設定されます。N=オプションを使用せず、行ポインターコントロール#を使用する場合、現在の DATA ステップの PUT ステートメントで使用した行ポインターコントロール#のうち、最大値が N=の値に割り当てられます。

ヒント N=PAGESIZE に設定すると、複数の列を含んだページの出力で、列ごとの出力ができます。

例 [“例 3: ページ全体のコンテンツを配置する” \(124 ページ\)](#)

NEW | OLD

新規ファイルまたは既存ファイルのどちらを出力に使用するかを指定します。NEW を指定すると、出力用に新規のファイルが開きます。ファイルがすでに存在する場合は、削除された後でファイルが再作成されます。OLD を指定すると、以前のファイルの内容が置き換えられます。NEW がデフォルト値です。

PAD | NOPAD

LRECL=オプションに指定した長さに達するまで、外部ファイルに書き込むレコードにブランクを追加するかどうかを指定します。

デフォルト 可変長ファイルに書き込む場合は、NOPAD がデフォルト値になります。固定長ファイルに書き込む場合は、PAD がデフォルト値になります。

ヒント PAD オプションを指定すると、可変長ファイルの中に固定長レコードを簡単に作成できます。

参照項目: [LRECL= \(116 ページ\)](#)

PAGESIZE=*value*

レポートの 1 ページあたりの行数を設定します。

別名 PS=

デフォルト PAGESIZE=システムオプションの値

範囲 15 から 32767 までの値を指定できます。

操作 TITLE ステートメントが現在定義されている場合、1 ページあたりの行数には、タイトルの行数が含まれます。

ヒント PAGESIZE=オプションに指定した値に達すると、出力ポインターは新しいページの 1 行目に移動します。

FILE LOG を指定する場合、最初のページに書き出される行数は、SAS 起動時の注意事項で使用する行数だけ少なくなります。たとえば、PAGESIZE=20 と指定し、SAS 起動時の注意事項に 9 行使用する場合、最初のページの出力に使用できるのは 11 行だけになります。

参照項目: [“PAGESIZE= システムオプション” \(SAS システムオプション: リファレンス\)](#)

PRINT | NOPRINT

キャリッジコントロール文字を出力行で使用するかどうかを制御します。

制限事項 ファイルへ出力する場合、N=オプションの値は 1 または PAGESIZE とする必要があります。

動作環境 ファイルに出力されるキャリッジコントロール文字は、動作環境によって異なる場合があります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

ヒント ファイル参照名に PRINT を使用する場合、PRINT オプションを指定する必要はありません。

対話型 SAS セッションで FILE PRINT を指定する場合、**出力**ウィンドウではフォームフィードコントロール文字をページ区切りと解釈します。また、フォームフィードの前に書き出される空行は出力から削除されます。**出力**ウィンドウから結果をフラットファイルに出力すると、改ページ文字を含まないファイルが生成されます。フォームフィード文字をファイルに含める必要がある場合、FILE ステートメントに物理ファイルの場所と PRINT オプションを指定する必要があります。

RECFM=*record-format*

出力ファイルのレコード形式を指定します。

D デフォルトのレコード形式(可変長形式)。

F 固定長形式。つまり、各レコードの長さは同じです。RECFM=F は、キャリッジコントロール文字を含む外部ファイルには使用しないでください。

- N バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。
- P プリント形式。SAS で、キャリッジコントロール文字が書き込まれます。
- V 可変長形式。各レコードは改行文字で終わります。

範囲 値は動作環境に依存しています。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

操作 グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

STOPOVER

PUT ステートメントが現在の行の長さを超えてデータアイテムを書き込もうとする場合、DATA ステップの処理を直ちに中止します。その場合、SAS は現在の行の長さを超えたデータアイテムを破棄し、エラー発生前に作成された行を書き込んでから、エラーメッセージを発行します。

デフォルト FLOWOVER

参照項目: [“FLOWOVER” \(114 ページ\)](#) および [“DROPOVER” \(111 ページ\)](#)

TERMSTR=

Linux でフォーマットされたファイル中の行末区切り文字を制御します。デフォルトでは、ラインフィードのみ、またはキャリッジリターンとラインフィードが行の終わりを示します。行末区切り文字を明示的に指定するには、次のいずれかの値を指定します。

CR キャリッジリターン。

CRLF キャリッジリターン、ラインフィード。

LF ラインフィード。このパラメーターは、Linux でフォーマットされたファイルの読み込みに使用されます。

TITLES | NOTITLES

現在のタイトル行をファイルのページ上に出力するかどうかを制御します。NOTITLES の指定を省略するか、TITLES を指定する場合、定義されているタイトルがあれば出力されます。

デフォルト TITLES

UNBUF

その後の FILE ステートメントでは、バッファされた書き込みをファイルに対して行わないよう SAS に指示します。このオプションは特に、データ収集デバイスに書き込んでいるときに適用されます。

FILE=variable

この FILE ステートメントの現在の出力バッファを参照する文字変数の名前を指定します。この変数は他の変数と同じように使用できます。また、値を割り当てることもできます。この変数の値は自動的に保持されます。初期値はブランクです。自動変数のように、_FILE_=変数はデータセットに書き込まれません。

制限事項 *variable* には、すでに定義済みの変数は指定できません。_FILE_=オプションには、その DATA ステップ内で初めて使用する変数を指定する必要があります。LENGTH ステートメントや ATTRIB ステートメントを使用して、_FILE_=に指定した変数の長さを設定したり変更することはできません。ただし、ATTRIB ステートメントや FORMAT ステートメントを使用して、この変数に出力形式を指定することができます。

操作 この文字変数の最大長は、指定した FILE ステートメントの論理レコード長 (LRECL) になります。ただし、プログラムの実行直前まで、SAS がこのファイルを開いて LRECL=オプションの値を確認することはありません。そのため、コンパイル実行中は、この変数のサイズは 32767 バイトになります。

ヒント この変数の内容に対する変更は、FILE ステートメントの現在の出力バッファの内容に直ちに反映されます。この FILE ステートメントに対する後続の PUT ステートメントにより、変更されたバッファの内容が書き出されます。N=オプションで複数の出力を指定した場合でも、_FILE_=に指定した変数は、指定した FILE ステートメントの現在の出力バッファにのみアクセスできます。

他のステートメントで _FILE_=オプションを使用せずに出力バッファの内容にアクセスするには、自動変数 _FILE_ を使用します。

参照項目: [“_FILE_変数の更新” \(121 ページ\)](#)

詳細

概要

デフォルトでは、PUT ステートメントの出力は、SAS ログに書き込まれます。FILE ステートメントを使用すると、この出力をプロシジャの出力先と同じ外部ファイル、または別の外部ファイルに書き出すことができます。キャリッジコントロール文字をファイルに追加するかどうかも指定できます。[PRINT | NOPRINT オプション \(118 ページ\)](#)を参照してください。

FILE ステートメントは実行ステートメントなので、条件(IF-THEN)処理で FILE ステートメントを使用できます。1 つの DATA ステップでは、複数の FILE ステートメントを使用して複数の外部ファイルに書き込むことができます。

外部ファイルの直接更新

FILE ステートメントを INFILE ステートメントや PUT ステートメントと併用すると、レコード全体の更新やレコード内の選択したフィールドのみの更新など、外部ファイルをバッファ内で更新できます。次のガイドラインに従ってください。

- INFILE ステートメントは必ず最初に記述します。
- INFILE ステートメントと FILE ステートメントで、同一の fileref または物理ファイル名を指定します。

- INFILE ステートメントには、FILE ステートメントと INFILE ステートメントの両方で使用できるオプションを使用します。(これらのオプションを FILE ステートメントに指定しても無視されます。)
- INFILE ステートメントに SHAREBUFFERS オプションを指定して、FILE ステートメントと INFILE ステートメントが同一のバッファを使用できるようにします。それによって CPU 時間が節約されるとともに、レコード全体ではなく個々のフィールドを更新することが可能になります。

出力バッファのコンテンツへのアクセス

`_FILE_`に指定する変数の他に、`_FILE_`自動変数を使用しても最後に実行した FILE ステートメントの出力バッファの内容を参照できます。この文字変数の値は自動的に保持されます。初期値は空白です。他の自動変数と同じように、`_FILE_`自動変数はデータセットに書き込まれません。

INFILE ステートメントに `_FILE_`=オプションを指定すると、指定した変数の値は自動変数 `_FILE_`でも間接的に参照されます。自動変数 `_FILE_`を指定し、特定の FILE ステートメントで `_FILE_`=オプションの指定を省略した場合、`_FILE_`=オプションの変数がある FILE ステートメントに対して内部的に作成されます。これ以外の場合に、特定の FILE ステートメントに対して `_FILE_`=オプションの変数が作成されることはありません。

実行時または参照時、自動変数 `_FILE_`の最大長は、そのときに使用している `_FILE_`=オプションの変数の最大長に一致します。ただし、自動変数 `_FILE_`が参照している他の変数の長さは実行時まで確定されないため、自動変数 `_FILE_`の長さはコンパイル時に 32,767 バイトに設定されます。たとえば、自動変数 `_FILE_`の値を長さが定義されていない新しい変数に割り当てると、新しい変数のデフォルトの長さは 32,767 バイトになります。LENGTH ステートメントや ATTRIB ステートメントを使用して、自動変数 `_FILE_`の変数の長さを設定したり、無効にすることはできません。ただし、FORMAT ステートメントや ATTRIB ステートメントを使用すると、自動変数 `_FILE_`に出力形式を割り当てることができます。

`_FILE_`変数の更新

他の SAS 変数と同じように、自動変数 `_FILE_`を更新することができます。次の 2 つの方法を使用できます。

- 割り当てステートメントで自動変数 `_FILE_`を使用する
- PUT ステートメントを使用する

次の形式で割り当てステートメントを使用すると、`_FILE_`変数を更新できます。

```
_FILE_ = <'string-in-quotation-marks' | character-expression>
```

この割り当てステートメントでは、現在の出力バッファの内容を更新し、バッファの長さを `'string-in-quotation-marks'`または `character-expression`の長さに設定します。ただし、割り当てステートメントを使用しても、PUT ステートメントの現在の出力ポインタの位置は変化しません。この FILE ステートメントの次の PUT ステートメントに対しては列 1 から、PUT ステートメントに後置@を使用する場合は既知の最終位置からバッファの更新を開始します。

次の例では、割り当てステートメントで現在の出力バッファの内容を更新します。PUT ステートメントの列ポインタの位置は変わりません。

```
file print;
```

```
_file_ = '_FILE_';
put 'This is PUT';
```

次の出力が作成されます。This is PUT

この例では、PUT ステートメントは列 14 から始まる現在の出力バッファの内容を更新しています。

```
file print;
_file_ = 'This is from FILE, sir.';
put @14 'both';
```

次の出力が作成されます。This is from both, sir.

PUT ステートメントを使用しても、自動変数_FILE_を更新することができます。PUT ステートメントでは出力バッファのデータに出力形式を設定し、自動変数_FILE_がそのバッファを参照するため、自動変数_FILE_は PUT ステートメントで更新されます。ただし、デフォルトでは、この出力バッファは、PUT ステートメントを実行し、現在のレコード(または、N=オプションに指定したレコードブロック)を書き出した後に消去されます。そのため、書き出し前に自動変数_FILE_の内容の確認や変更を行うには、PUT ステートメントに後置@または後置@@を指定します(N=1 の場合)。値が N=1 以外の場合、行ポインターコントロールの最終位置をレコードブロックの最後のレコードに指定する PUT ステートメントで、後置@または後置@@を使用します。

この例では、N=1 の場合、最初の PUT ステートメントで Something を書き出し、末尾の@で行を保持し、_FILE_を Something に設定します。そして、Y には Something is here が割り当てられます。

```
file ABC;
put 'Something' @;
Y = _file_||' is here';

file ABC;
put 'Nothing';
Y = _file_||' is here';
```

2 番目の FILE ステートメントでは、PUT ステートメントが Nothing を書き出し、現在の出力バッファがクリアされます。2 番目の PUT ステートメントには末尾に@がないため、Y には is here が割り当てられます。

自動変数_FILE_に変更を加えると、現在の FILE ステートメントの出力バッファに直ちに変更が反映されます。FILE ステートメントに続けて PUT ステートメントを実行すると、変更したバッファの内容が書き出されます。

N=オプションに複数のバッファを指定した場合でも、自動変数_FILE_は、その FILE ステートメントで現在使用中の出力バッファにのみアクセスします。N=オプションに指定したすべてのバッファにアクセスできますが、特定のバッファを現在の出力バッファに設定するには、PUT ステートメントに#行ポインターコントロールを指定する必要があります。

比較

- FILE ステートメントは、PUT ステートメントの出力ファイルを指定します。INFILE ステートメントは、INPUT ステートメントの入力ファイルを指定します。

- FILE ステートメントおよび INFILE ステートメントでは、使用する外部ファイルに関する追加情報を SAS システムに提供するオプションを使用できます。
- プログラムエディター、ログ、出力の各ウィンドウでは、FILE コマンドで外部ファイルを指定し、ウィンドウの内容をそのファイルに書き出します。

例

例 1: 新規ページの開始時にステートメントを実行する

次の DATA ステップは、HEADER=オプションの使用法を示しています。

- レポートを出力します。DATA_NULL_を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。


```
data _null_;
  set sprint;
  by dept;
```
- SAS 出力ウィンドウに出力を送信します。ヘッダー情報を参照します。ファイル参照名に PRINT を指定し、プロシジャの出力先と同じ場所に出力を送信します。HEADER=オプションは、各ページのヘッダーを作成するステートメントの前に配置されているラベルを参照します。


```
file print header=newpage;
```
- 部門ごとに新しいページを開始します。


```
if first.dept then put _page_;
put @22 salesrep @34 salesamt;
```
- 各ページにヘッダーを出力します。新しいページを開始するたびに、次のステートメントが実行されます。ラベルが指定されたグループの最後のステートメントとして、ラベルの前に RETURN ステートメントを指定する必要があります。


```
return;
newpage:
put @20 'Sales for 1989' /
  @20 dept=;
return;
run;
```

例 2: 現在のページに残っている行数に基づき新規ページを決定する

次の DATA ステップでは、LINESLEFT=オプションを使用し、現在のページに残っている行数に基づいて改ページを実行する位置を決定する例を示します。

- レポートを出力します。DATA_NULL_を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。


```
data _null_;
  set info;
```
- 標準の SAS 出力ウィンドウに出力を送信します。ファイル参照名に PRINT を指定し、プロシジャの出力先と同じ場所に出力を送信します。LINESLEFT オプショ

ンは、変数 REMAIN に現在のページに残っている行数が格納されていることを示しています。

```
file print linesleft=remain pagesize=20;
put @5 name @30 phone
    @35 bldg @37 room;
```

- 現在のページの残りの行数が指定した値よりも少なくなると、新しいページが開始されます。この条件に従って、PUT_PAGE_は新しいページを開始し、ポインターを 1 行目に配置します。

```
if remain<7 then put _page_;
run;
```

例 3: ページ全体のコンテンツを配置する

この例では DATA ステップで N=PAGESIZE オプションを使用し、2 列配置された電話番号リストを生成する方法を示しています。各列には、名前と電話番号が入っています。

- レポートを作成し、SAS 出力ウィンドウに書き込みます。DATA_NULL_を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。ファイル参照名は PRINT です。キャリッジコントロール文字を使用して、出力するファイルの特性にあった出力を行います。N=PAGESIZE と指定すると、出力ポインターをページ全体に移動できます。

```
data _null_;
file 'external-file' print n=pagesize;
```

- レポートに使用する列を指定します。DATA ステップの繰り返しごとに、DO ループを 2 回繰り返します。COL の値は最初の繰り返しで 1 に設定され、2 回目の繰り返しで 40 に設定されます。

```
do col=1, 40;
```

- データを 20 行書き込みます。次の DO ループを 20 回繰り返して列 1 に 20 行データを書き込みます。書き込みが完了すると、外側のループで COL が 40 に設定されます。この DO ループを 20 回繰り返し、列 2 にデータを 20 行書き込みます。LINE および COL の値は DO ステートメントによって設定され、増分が追加されます。この LINE と COL の値によって、PUT ステートメントでページ上に NAME と PHONE の値を書き込む位置が制御されます。

```
do line=1 to 20;
set info;
put #line @col name $20. +1 phone 4.;
end;
```

- 2 列分のデータがすべて揃ったら、ページ全体を出力します。次の END ステートメントで外側のループを終了します。PUT_PAGE_は現在のページを出力し、ポインターを次のページの最上部に移動させます。

```
end;
put _page_;
run;
```

例 4: 現在の出力ファイルの識別

次の DATA ステップでは、ファイル識別メッセージを SAS ログに出力し、現在の出力ファイルの値を変数 MYOUT に割り当てます。次の PUT ステートメントは、

MYOUT に適切な値を割り当ててから、この変数の値を出力ファイルに書き込む例を示しています。

```
data _null_;
  length myout $ 200;
  file file-specification filename=myout;
  put myout=;
  stop;
run;
```

次の PUT ステートメントは、現在の出力ファイルに、ファイルの物理名が記述された行を書き込みます。

```
MYOUT=your-output-file
```

例 5: 現在の出力ファイルを動的に変更する

次の DATA ステップでは、FILEVAR=オプションを使用して、現在開かれている出力ファイルを新しい物理ファイルに動的に変更します。

- レポートを出力します。その次に、長い文字変数を作成します。DATA_NULL_を使用して、データセットの作成ではなく、レポートの出力を行うように指示します。LENGTH ステートメントを使用して、外部ファイル名を格納するのに十分な長さを割り当てた変数を作成します。

```
data _null_;
  length name $ 200;
```

- インストリームデータ行を読み込み、NAME 変数に値を割り当てます。

```
input name $;
```

- NAME 変数の値が変わったら、現在の出力ファイルを閉じ、新しいファイルを開きます。file-specification は、プレースホルダーです。これには、有効な SAS 名を指定できます。

```
file file-specification filevar=name mod;
date = date();
```

- 現在開かれている出力ファイルにログレコードを追加します。

```
put 'records updated ' date date.;
```

- 外部ファイルの名前を指定します。

```
datalines;
external-file-1
external-file-2
external-file-3
;
```

例 6: 出力行が出力ファイルの行の長さを超える場合

変数の行の長さを合計すると出力行(80 文字)よりも長くなるので、次の PUT ステートメントでは、自動的に 3 つのレコードに分割して書き込みます。

```
file file-specification linesize=80;
put name $ 1-50 city $ 51-90 state $ 91-104;
```

変数 NAME の値は 1 番目のレコードに出力されます。変数 CITY の値の出力は 2 番目のレコードの列 1 から、変数 STATE の値の出力は 3 番目のレコードの列 1 から始まります。

例 7: TCP/IP ソケットを使用したデータ読み込みとテキスト書き込み

この例では、TCP/IP ソケットを使用してファイルから生データを読み込んでいます。INFILE ステートメントでは、NBYTE=オプションを使用しています。

```
/* Start this first as the server */
filename serve socket ':5205' server
  recfm=s
  lrecl=25 blocksize=2500;
data _null_;
  nb=25;
  infile serve nbyte=nb;
  input text $char25.;
  put _all_;
run;
```

この例では、TCP/IP ソケットを使用してファイルにテキストを出力します。

```
/* While the server test is running, */
/*continue with this as the client. */
filename client socket "&hstname:5205"
  recfm=s
  lrecl=25 blocksize=2500;
data _null_;
  file client;
  put 'Some text to length 25...';
run;
```

例 8: 出力ファイル書き込み時のエンコードの指定

この例では、SAS データセットから外部ファイルを作成します。現在のセッションエンコーディングは Wlatin1 ですが、外部ファイルのエンコーディングは UTF-8 にする必要があります。デフォルトでは、外部ファイルは現在のセッションエンコーディングを使用して書き込まれます。

外部ファイルへの書き込み時に使用するエンコーディングを指定するには、ENCODING=オプションを指定します。外部ファイルを UTF-8 エンコーディングにするように指示すると、外部ファイルへの書き込み時に、Wlatin1 から指定した UTF-8 エンコーディングにデータがトランスコードされます。

```
libname myfiles 'SAS-library';
filename outfile 'external-file';
data _null_;
  set myfiles.cars;
  file outfile encoding="utf-8";
  put Make Model Year;
run;
```


例 9: FTP アクセスメソッドを使用して Excel スプレッドシートにデータを出力する

この例では、FTP アクセスメソッドと FILEVAR オプションを使用して、複数の Microsoft Excel ワークシートにデータを書き込みます。

```
data _null_;
  do i = 1 to 3;
    sheet = cats('excel|[test-sheet.xlsx]Sheet', i, '!r1c1:r10c2');
    file area ftp filevar=sheet;
    do x = 1 to 10;
      y = 2*x;
      put x y;
    end;
  end;
run;
```

関連項目

- [“名前の長さの規則” \(SAS プログラマガイド: エッセンシャル\)](#)

ステートメント:

- [“FILENAME ステートメント” \(128 ページ\)](#)
- [“INFILE ステートメント” \(143 ページ\)](#)
- [“LABEL ステートメント” \(230 ページ\)](#)
- [SAS Intelligence Platform: Application Server Administration Guide の “LOCKDOWN ステートメント”](#)
- [“PUT ステートメント” \(289 ページ\)](#)
- [“RETURN ステートメント” \(350 ページ\)](#)
- [“FILENAME Statement: ACTIVEMQ Access Method” \(Application Messaging with SAS\)](#)
- [“FILENAME ステートメント: Azure Data Lake Storage アクセス方式” \(SAS グローバルステートメント: リファレンス\)](#)
- [“FILENAME Statement: JMS Access Method” \(Application Messaging with SAS\)](#)
- [“FILENAME ステートメント: S3 アクセスメソッド” \(SAS グローバルステートメント: リファレンス\)](#)

システムオプション:

- [SAS Intelligence Platform: Application Server Administration Guide の “LOCKDOWN システムオプション”](#)

FILENAME ステートメント

SAS ファイル参照名を外部ファイルまたは出力デバイスと関連付けたり、ファイル参照名と外部ファイルの関連付けを取り消します。また、外部ファイルの属性を書き込みます。

注: [FILENAME ステートメント](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: Azure アクセスメソッド

Microsoft Azure Data Lake Storage のデータにアクセスできるようにします。

注: [FILENAME ステートメント: Azure アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: CATALOG アクセスメソッド

SAS カタログを外部ファイルとして参照できます。

注: [FILENAME ステートメント: CATALOG アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: CLIPBOARD アクセスメソッド

ホストコンピューター上のクリップボードに対して、テキストデータを読み込み、書き込みできます。

注: [FILENAME ステートメント: CLIPBOARD アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: DATAURL アクセスメソッド

ユーザー指定のテキストからデータを読むことができます。

注: [FILENAME ステートメント: DATAURL アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: EMAIL(SMTP)アクセスメソッド

SMTP(Simple Mail Transfer Protocol)電子メールインターフェイスを介して、SAS から電子メールをプログラムによって送信できます。

注: [FILENAME ステートメント: EMAIL\(SMTP\)アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: FILESRVC アクセスメソッド

SAS Viya プラットフォームのファイルサービス ON でファイルを保存し、取得できるようにします。

注: [FILENAME ステートメント: FILESRVC アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: FTP アクセスメソッド

FTP プロトコルを使用してリモートファイルにアクセスできます。

注: [FILENAME ステートメント: FTP アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: Hadoop アクセスメソッド

構成ファイルに場所が指定されている Hadoop 分散ファイルシステム(HDFS)上のファイルにアクセスできます。

注: [FILENAME ステートメント: Hadoop アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: S3 アクセスメソッド

Amazon S3 ファイルにアクセスできるようにします。

注: [FILENAME ステートメント: S3 アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: SOCKET アクセスメソッド

TCP/IP ソケットからの読み込みや書き込みができるようにします。

注: [FILENAME ステートメント: SOCKET アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: URL アクセスメソッド

URL アクセスメソッドを使用してリモートファイルにアクセスできます。

注: [FILENAME ステートメント: URL アクセスメソッド](#)は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: WebDAV アクセスメソッド

WebDAV プロトコルを使用してリモートファイルにアクセスできます。

注: [FILENAME ステートメント: WebDAV アクセスメソッド](#) は SAS グローバルステートメントに移動しました。

FILENAME ステートメント: ZIP アクセスメソッド

ZIP ファイルにアクセスできるようにします。

注: [FILENAME ステートメント: ZIP アクセスメソッド](#) は SAS グローバルステートメントに移動しました。

FOOTNOTE ステートメント

プロシジャまたは DATA ステップの出力の下部にテキストを 10 行まで書き込みます。

注: [FOOTNOTE ステートメント](#) は SAS グローバルステートメントに移動しました。

FORMAT ステートメント

変数に出力形式を関連付けます。

該当要素:	DATA ステップまたは PROC ステップ
カテゴリ:	CAS In-Database 処理 情報
種類:	宣言

構文

```
FORMAT variable-1 <...variable-n> <format> <DEFAULT=default-format>;
FORMAT variable-1 <...variable-n> format <DEFAULT=default-format>;
FORMAT variable-1 <...variable-n> format variable-1 <...variable-n> format;
```

引数

variable

1 つまたは複数の変数に出力形式を関連付けます。少なくとも 1 つの *variable* を指定する必要があります。

変数から出力形式の関連付けを取り消すには、DATA ステップまたは PROC DATASETS で出力形式を指定せず、FORMAT ステートメントでこの変数を使用します。DATA ステップでは、この FORMAT ステートメントを SET ステートメントの後に配置します。“例 3: 出力形式の取り消し” (135 ページ) を参照してください。PROC DATASETS を使うこともできます。

format

変数の値を出力するときに適用する出力形式を指定します。

ヒント FORMAT ステートメントを使用して変数に関連付けられた出力形式は、コロン修飾子で使われる出力形式と同じように、後続の PUT ステートメントで動作します。コロン修飾子の使用方法の詳細については、“PUT ステートメント: リスト” (321 ページ) を参照してください。

参照項 [SAS 出力形式と入力形式: リファレンス](#)
目:

DEFAULT=default-format

FORMAT ステートメントにリストされていない変数の値を表示するときに使用する一時的なデフォルトの出力形式を指定します。このデフォルトの出力形式は、現在の DATA ステップにのみ適用されます。出力データセット内の変数に常に関連付けられるわけではありません。

DEFAULT= に指定した出力形式の適用対象

- FORMAT ステートメントまたは ATTRIB ステートメントに指定されていない変数
- SAS データセット内に常に関連付けられてる出力形式がない変数
- 出力形式を明示的に指定して出力されていない変数

デフォルト DEFAULT=の指定を省略すると、デフォルトの数値の出力形式として BESTw. を使用し、デフォルトの文字の出力形式として \$w. をそれぞれ使用します。

制限事項 このオプションは DATA ステップ内でのみ使用できます。

ヒント DEFAULT=は、FORMAT ステートメントのどの位置に指定してもかまいません。数値に対するデフォルト値、文字に対するデフォルト値、またはその両方を指定できます。

例 “例 1: 出力形式とデフォルト値を割り当てる” (133 ページ)

詳細

FORMAT ステートメントでは、標準の SAS 出力形式または PROC FORMAT に定義したユーザー定義の出力形式を使用できます。1 つの FORMAT ステートメントでは、同じ出力形式を複数の変数に関連付けることができます。また、変数ごとに異なる出力形式を関連付けることもできます。同じ変数が複数の FORMAT ステートメント内に指定されている場合、最後に関連付けられた出力形式が使用されます。

変数に出力形式を常に割り当てるようにするには、DATA ステップ内で FORMAT ステートメントを使用します。ステートメントを使用すると、指定した変数が含まれる SAS データセットのディスクリプター情報が変更されます。一部の PROC ステップでも FORMAT ステートメントを使用できますが、ルールは異なります。詳細については、[Base SAS プロシージャガイド](#)を参照してください。

比較

ATTRIB ステートメントと FORMAT ステートメントは、どちらも変数に出力形式に関連付けることができます。また、どちらのステートメントを使用しても変数に関連付けられた出力形式を変更することができます。変数に関連付けられた出力形式の変更や取り消しを行うには、PROC DATASETS 内で FORMAT ステートメントを使用します。

例

例 1: 出力形式とデフォルト値を割り当てる

この例では、FORMAT ステートメントを使用して数値変数と文字変数に出力形式とデフォルトの出力形式を割り当てます。デフォルトの出力形式はデータセット内の変数に関連付けられません。しかし、現在の DATA ステップで PUT ステートメントを使用して変数を入力するとき、指定した内容が反映されます。

```
data tstfmt;
  format W $char3.
  Y 10.3
  default=8.2 $char8.;
  W='Good morning.';
  X=12.1;
  Y=13.2;
  Z='Howdy-doody';
  put W/X/Y/Z;
run;
proc contents data=tstfmt;
run;
proc print data=tstfmt;
run;
```

出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

アウトプット 2.9 PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポート

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
1	W	Char	3	\$CHAR3.
3	X	Num	8	
2	Y	Num	8	10.3
4	Z	Char	11	

アウトプット 2.10 PROC PRINT で生成したレポート

Obs	W	Y	X	Z
1	Goo	13.200	12.1	Howdy-doody

デフォルトの出力形式が変数 X と Z に適用され、指定した出力形式が変数 W と Y に適用されています。

PUT ステートメントの実行結果は次のようになります。

```
----+----1----+----2
Goo
12.10
13.200
Howdy-do
```

例 2: 1 つの出力形式を複数の変数に関連付ける

この例では、FORMAT ステートメントを使用して 1 つの出力形式を複数の変数に関連付けます。

```
data report;
  input Item $ 1-6 Material $ 8-14 Investment 16-22 Profit 24-31;
  format Item Material $upcase9. Investment Profit dollar15.2;
  datalines;
shirts cotton 2256354 83952175
ties silk 498678 2349615
suits silk 9482146 69839563
belts leather 7693 14893
shoes leather 7936712 22964
;
run;
options pageno=1 nodate ls=80 ps=64;
proc print data=report;
  title 'Profit Summary: Kellam Manufacturing Company';
run;
```


アウトプット 2.11 1つの出力形式を複数の変数に関連付けた後の出力結果

Profit Summary: Kellam Manufacturing Company				
Obs	Item	Material	Investment	Profit
1	SHIRTS	COTTON	\$2,256,354.00	\$83,952,175.00
2	TIES	SILK	\$498,678.00	\$2,349,615.00
3	SUITS	SILK	\$9,482,146.00	\$69,839,563.00
4	BELTS	LEATHER	\$7,693.00	\$14,893.00
5	SHOES	LEATHER	\$7,936,712.00	\$22,964.00

例 3: 出力形式の取り消し

この例では、SAS データセットにある変数から指定した出力形式の関連付けを取り消します。FORMAT ステートメントと SET ステートメントを指定する順序が重要です。

```
data rtest;
  set rtest;
  format x;
run;
```

関連項目

- [“DATASETS プロシジャ” \(Base SAS プロシジャガイド\)](#)

ステートメント:

- [“ATTRIB ステートメント” \(52 ページ\)](#)

GOTO ステートメント

指定したステートメントラベルにプログラム実行を移動します。RETURN ステートメントが後に続く場合、DATA ステップの先頭に実行を戻します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行
別名:	GOTO

構文

GOTO *label*;

引数

label

GOTO の出力先となるステートメントラベルを指定します。出力先は同じ DATA ステップ内になければなりません。*label* 引数は必ず指定する必要があります。

比較

GOTO ステートメントと LINK ステートメントは似ています。GOTO ステートメントは RETURN ステートメントを指定せずに使用できますが、通常、LINK ステートメントでは RETURN ステートメントを明示的に指定して使用します。GOTO ステートメントと LINK ステートメントでは、後続く RETURN ステートメントのアクションが異なります。LINK ステートメントの後に RETURN ステートメントを指定する場合は、LINK ステートメントの直後のステートメントに実行を戻します。GOTO ステートメントの後に RETURN ステートメントを指定し、ステートメントの前に LINK ステートメントがない場合は、DATA ステップの先頭に実行を戻します。このとき、GOTO ステートメントの前に LINK ステートメントがある場合は、LINK ステートメントの直後の最初のステートメントから実行を続けます。

GOTO ステートメントは、DO-END および IF-THEN/ELSE のプログラムロジックに置き換えることができます。

例: GOTO ステートメントで RETURN ステートメントを使用する

次に示す GOTO ステートメントを使用します。

- この例では、条件が真になる場合、GOTO ステートメントは ADD という名前のラベルに移動し、そこから実行を続けます。条件が偽になる場合、PUT ステートメントを実行し、GOTO ラベルに指定されているステートメントを実行します。

```
data info;
  input x;
  if 1<=x<=5 then goto add;
  put x=;
  add: sumx+x;
  datalines;
7
6
323
;
```

各 DATA ステップの最後に暗示的な RETURN ステートメントが含まれているため、合計ステートメントを実行すると、プログラムの実行は DATA ステップの先

頭に戻されます。そのため、DATA ステップの最後に明示的な RETURN ステートメントを指定する必要はありません。

- 条件が偽になるオブザベーションに対して合計ステートメントを実行しない場合は、コードを変更し、次のように明示的な RETURN ステートメントを指定します。

```
data info;
  input x;
  if 1<=x<=5 then goto add;
  put x=;
  return;
  /* SUM statement not executed */
  /* if x<1 or x>5          */
add: sumx+x;
datalines;
7
6
323
;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“Label: ステートメント” \(234 ページ\)](#)
- [“LINK ステートメント” \(242 ページ\)](#)
- [“RETURN ステートメント” \(350 ページ\)](#)

IF ステートメント: サブセット化

指定した式の条件に一致するオブザベーションの処理を継続します。

該当要素: DATA ステップ
 カテゴリ: アクション
 CAS
 In-Database 処理
 種類: 実行

注: WHERE ステートメントで乱数関数を使用すると、サブセット IF ステートメントで乱数関数を使用する場合とは異なる結果セットが生成される場合があります。SAS により基準がどのように内部で最適化され、予想される動作であるかがこの違いが発生する可能性があります。

構文

IF *expression*;

引数

expression

SAS 式を指定します。

詳細

基本情報

サブセット化 IF ステートメントにより、DATA ステップにおいて、この IF ステートメントに指定した式の条件を満たす SAS データセットの生データレコードやオブザベーションのみ処理が継続されます。式がオブザベーションに対して真として評価される場合(値がゼロでも欠損値でもない場合)、DATA ステップが引き続き実行され、オブザベーションが出力データセットに挿入されます。実行後に生成される SAS データセットには、オリジナルの外部ファイルまたは SAS データセットのサブセットが含まれます。

式が偽として評価される場合(値が 0 または欠損値である場合)、それ以降のステートメントはオブザベーションやレコードに対して実行されません。現在のオブザベーションはデータセットに書き込まれず、DATA ステップの残りのプログラムステートメントは実行されません。サブセット化 IF ステートメントでは、オブザベーションの処理を中止するために他のステートメントを実行する必要がないため、すぐに DATA ステップの先頭に戻ります。

CONTAINS 演算子や LIKE 演算子に相当する演算子を IF ステートメントで使用する

WHERE 句内の LIKE 演算子は、複数のワードからなるパターンと一致します。IF ステートメントで同じ結果を得るには、'='演算子を使用します。この演算子は、文字列の先頭から始まるパターンと一致します。次に例を示します。

```
data test;
  input name $;
  datalines;
John
Diana
Diane
Sally
Doug
David
;
run;
```

```
data test;
  set test;
  if name = 'D';
run;
```

```
proc print;
run;
```

WHERE 句内の CONTAINS 演算子は、値の内部に指定の文字列が含まれているかどうかをチェックします。IF ステートメントで同じ結果を得るには、INDEX 関数を使用します。次に例を示します。

```
data test;
  set test;
  if index(name,'ian') ge 1;
run;

proc print;
run;
```

比較

- サブセット化 IF ステートメントは、次の IF-THEN ステートメントと同じです。

```
if not (expression)
then delete;
```

- SAS データセットの作成時、オブザベーションを含める条件を指定するほうが簡単な場合はサブセット化 IF ステートメントを使用します。オブザベーションを除外する条件を指定するほうが簡単な場合は DELETE ステートメントを使用します。
- サブセット化 IF ステートメントと WHERE ステートメントは同じではありません。この 2 つのステートメントでは、動作や生成される出力データセットが異なる場合があります。重要な違いは次の点になります。
 - サブセット化 IF ステートメントでは、プログラムデータベクトルに読み込まれたオブザベーションを選択します。WHERE ステートメントでは、プログラムデータベクトルに読み込まれる前にオブザベーションを選択します。サブセット化 IF ステートメントでは、入力データセットの各オブザベーションをプログラムデータベクトルに読み込む必要があるため、WHERE ステートメントと比較すると効率的ではない場合があります。
 - SAS データセットのインタリーブ、マージ、更新を実行する DATA ステップでは、サブセット化 IF ステートメントと WHERE ステートメントの結果が異なる場合があります。
 - サブセット化 IF ステートメントと MERGE ステートメントを併用する場合、現在のオブザベーションを結合した後にオブザベーションが選択されます。WHERE ステートメントと MERGE ステートメントを同時に使用する場合、現在のオブザベーションを結合する前に各データセットに選択基準を適用します。
 - サブセット化 IF ステートメントでは、既存の SAS データセットや、INPUT ステートメントを使用して読み込んだ生データからオブザベーションを選択できます。WHERE ステートメントでは、既存の SAS データセットからのみオブザベーションを選択できます。
 - サブセット化 IF ステートメントは実行ステートメントですが、WHERE ステートメントは実行ステートメントではありません。

例: オブザベーションの限定

- 次の例では、変数 SEX の値が F であるオブザベーションのみがデータセットに含まれます。

```
if sex='F';
```

- 次の例では、変数 AGE の値が欠損値でも 0 でもないすべてのオブザベーションがデータセットに含まれます。

```
if age;
```

関連項目

- [“WHERE 式” \(SAS プログラマガイド: エssenシャル\)](#)

データセットオプション:

- [“WHERE= データセットオプション” \(SAS データセットオプション: リファレンス\)](#)

ステートメント:

- [“DELETE ステートメント” \(85 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(140 ページ\)](#)
- [“WHERE ステートメント” \(389 ページ\)](#)

IF-THEN/ELSE ステートメント

特定の条件を満たすオブザベーションに対して、SAS ステートメントを実行します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

```
IF expression THEN statement;  
<ELSE statement;>
```

引数

expression

任意の SAS 式で、これは必須の引数です。

statement

実行 SAS ステートメントまたは DO グループです。

詳細

SAS では、IF-THEN ステートメントの式を評価し、非ゼロ、ゼロ、または欠損のいずれかの数値の結果を生成します。評価結果が非ゼロおよび非欠損の場合、この式は真になります。評価結果がゼロまたは欠損の場合、この式は偽になります。

IF 句に指定した条件に一致すると、IF-THEN ステートメントは SAS データセットから読み込んだオブザベーション、外部ファイルのレコード、または計算値に対して、SAS ステートメントを実行します。THEN 句が実行されない場合、オプションの ELSE ステートメントが代替アクションを指示します。ELSE ステートメントを使用する場合、IF-THEN ステートメントの後に指定します。

ELSE ステートメントを指定せずに IF-THEN ステートメントを使用する場合、すべての IF-THEN ステートメントが評価されます。ELSE ステートメントを指定して IF-THEN ステートメントを使用する場合、真のステートメントが検出されるまで IF-THEN ステートメントが実行されます。その後の IF-THEN ステートメントは評価されません。

注: 効率を高めるには、IF-THEN/ELSE ステートメントに条件をその確率の降順になるように指定します。

比較

- 相互排他な多数の条件を指定する場合、IF-THEN ステートメントではなく SELECT グループを使用します。
- IF 句に指定した条件に一致するオブザベーションまたはレコードのみの処理を続けるには、THEN 句を指定せずにサブセット化 IF ステートメントを使用します。

例: IF-THEN/ELSE ステートメントの各種の指定方法

これらの例では、IF-THEN/ELSE ステートメントの各種の指定方法を示します。

- if x then delete;
- if status='OK' and type=3 then count+1;
- if age ne agecheck then delete;
- if x=0 then

```

        if y ne 0 then put 'X ZERO, Y NONZERO';
        else put 'X ZERO, Y ZERO';
        else put 'X NONZERO';
    ■ if answer=9 then
        do;
            answer=.;
            put 'INVALID ANSWER FOR ' id=;
        end;
    else
        do;
            answer=answer10;
            valid+1;
        end;
    ■ data region;
        input city $ 1-30;
        if city='New York City'
            or city='Miami' then
            region='ATLANTIC COAST';
        else if city='San Francisco'
            or city='Los Angeles' then
            region='PACIFIC COAST';
        datalines;
        ...more data lines...
    ;

```

関連項目

- [“IF ステートメント” \(SAS プログラマガイド: エッセンシャル\)](#)

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“IF ステートメント: サブセット化” \(137 ページ\)](#)
- [“SELECT ステートメント” \(352 ページ\)](#)

%INCLUDE ステートメント

SAS プログラミングステートメント、データ行、またはその両方を現在の SAS プログラムに読み込みます。

注: [%INCLUDE ステートメント](#) は SAS グローバルステートメントに移動しました。

INFILE ステートメント

INPUT ステートメントで読み込む外部ファイルを指定します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。 SAS がロックダウン状態にある場合、ロックダウンパスリストに含まれていないファイルに関しては、INFILE ステートメントを使用できません。詳細については、“LOCKDOWN” (<i>SAS プログラマガイド: エッセンシャル</i>)を参照してください。 INFILE ステートメントは、前の FILENAME ステートメントによって設定されたデバイスの種類をオーバーライドしたり、変更したりすることはできません。

構文

INFILE *file-specification* <*device-type*> <*options*>;

INFILE *DBMS-specifications*;

引数

file-specification

入力データレコードのソースを指定します。これは外部ファイルまたはインストリームデータになります。*File-specification* には、次の形式を指定できます。

'external-file'

外部ファイルの物理名を指定します。物理名とは、動作環境がそのファイルにアクセスするために使用する名前のことです。

fileref

外部ファイルのファイル参照名を指定します。

範囲 1 バイトから 8 バイトまで

要件 ファイル参照名は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。

参照項目: “[FILENAME ステートメント](#)” (128 ページ)

fileref(file)

集約記憶域に関連付けられているファイル参照名の後に、その集約記憶域に含まれているファイルまたはメンバーの名前を丸かっこで囲んで指定します。

要件 ファイルが集約された保存場所にあり、SAS 名が無効な場合、名前を引用符で囲む必要があります。

ファイル参照名は、FILENAME ステートメントまたは FILENAME 関数を使用するか、適切な動作環境のコマンドを使用して、あらかじめ前のステップで外部ファイルに関連付けておく必要があります。

動作環境 ディレクトリ、MACLIB、区分データセットなど複数のファイルをまとめて保存する集約記憶域の名前は動作環境によって異なります。外部ファイルの指定方法の詳細については、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。

参照 [“FILENAME ステートメント” \(128 ページ\)](#)

項目:

DATALINES**DATALINES4**

DATA ステップ内の DATALINES ステートメントまたは DATALINES4 ステートメントの直後に入力データが続くことを指定します。DATALINES を指定すると、INFILE ステートメントの各種オプションを使用して、INPUT ステートメントがインストリームデータ行を読み込む方法を制御できます。

別名 CARDS | CARDS4

例 [“例 1: 区切り文字の扱い方の変更” \(170 ページ\)](#)

ヒント ERRORCHECK オプションが STRICT に設定されている場合、SYSERR マクロ変数を使用すると、*file-specification* の指定の有無を確認できます。

device-type

デバイスの種類またはアクセスメソッドを指定します。これは、ファイル参照名が入出力デバイスや物理ファイルではない場所を参照している場合に使用されます。

ACTIVEMQ

このアクセスメソッドを指定すると、ActiveMQ メッセージブローカーへアクセスできます。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: [“FILENAME Statement: ACTIVEMQ Access Method” \(Application Messaging with SAS\)](#)

CATALOG

CATALOG アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: CATALOG アクセスメソッドで指定可能なオプションの一覧については、“[FILENAME ステートメント: CATALOG アクセスメソッド](#)” (128 ページ)を参照してください。

CLIPBOARD

CLIPBOARD アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: CLIPBOARD アクセスメソッドで指定可能なオプションの一覧については、“[FILENAME ステートメント: CLIPBOARD アクセスメソッド](#)” (128 ページ)を参照してください。

DISK

デバイスがディスクドライブであることを指定します。

ヒント ディスク上のファイルにファイル参照名を割り当てる場合、DISK を指定する必要はありません。

DUMMY

外部ファイルからの入力を破棄するように指定します。

操作 DUMMY オプションは、対応する INPUT ステートメントでファイルの即時終了に遭遇していることを示します。DUMMY と一緒に FILEVAR=オプションが指定された場合、FILEVAR=変数で指定されたファイルは読み込まれません。

ヒント テストを実行する場合は DUMMY を指定すると便利です。

FTP

FTP アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: FTP アクセスメソッドで指定可能なオプションの一覧については、“[FILENAME ステートメント: FTP アクセスメソッド](#)” (129 ページ)を参照してください。

例 `infile dummy ftp user='myuid' pass='xxxx' filevar=file_to_read;`

GTERM

出力デバイスの種類がグラフィックデータを受信するグラフィックデバイスであると指定します。

HADOOP

Hadoop アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: Hadoop アクセスメソッドで指定可能なオプションの一覧については、[“FILENAME ステートメント: Hadoop アクセスメソッド”\(130 ページ\)](#)を参照してください。

JMS

Java Message Service (JMS) の出力先を指定します。

PIPE

名前の付いていないパイプを指定します。

注 一部の動作環境では、パイプはサポートされていません。

PLOTTER

バッファされていないグラフィック出力デバイスを指定します。

PRINTER

プリンターまたはプリンタースプールファイルを指定します。

SOCKET

SOCKET アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照項目: SOCKET アクセスメソッドで指定可能なオプションの一覧については、[“FILENAME ステートメント: SOCKET アクセスメソッド”\(130 ページ\)](#)を参照してください。

TAPE

テープドライブを指定します。

TERMINAL

ユーザーの端末を指定します。

UPRINTER

ユニバーサルプリンターの定義名を指定します。

ヒント FILENAME ステートメントにプリンター名を指定しない場合、PRINTERPATH オプションによって、使用するユニバーサルプリンターと出力先が制御されます。

URL

URL アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照 [“STATUS=variable” \(157 ページ\)](#)。URL アクセスメソッドで指定可能なオプションの一覧については、[“FILENAME ステートメント: URL アクセスメソッド” \(130 ページ\)](#)を参照してください。

WEBDAV

WEBDAV アクセスメソッドを指定します。

操作 DATA ステップでアクセスメソッドのオプションを認識できない場合、DATA ステップはこのオプションをアクセスメソッドに渡して処理します。

参照 WEBDAV アクセスメソッドで指定可能なオプションの一覧については、[“FILENAME ステートメント: WebDAV アクセスメソッド” \(130 ページ\)](#)を参照してください。

別名 DEVICE=*device-type*

デフォルト DISK

要件 *device-type* または DEVICE=*device-type* は、ステートメントの *file-specification* の直後に記述する必要があります。

動作環境 指定するデバイスによっては、さらに情報を指定することが必要になる場合があります。DISK 以外の値を指定する前に、各動作環境向けの SAS ドキュメントを参照してください。動作環境によっては、ここで説明した値の他に使用できる値が存在する場合があります。

オプション

BLKSIZE=*block-size*

入力ファイルのブロックサイズを指定します。

別名 BLK=

デフォルト ご使用の動作環境によって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

COLUMN=*variable*

変数名を指定します。指定した変数の値は、入力ポインターの現在の列位置を示す値に設定されます。自動変数と同じように、COLUMN=に指定した変数はデータセットに書き込まれません。

別名 COL=

参照項目: [LINE= \(152 ページ\)](#)

例 [“例 8: ポインター位置の出力” \(177 ページ\)](#)

DELIMITER= *delimiter(s)*

LIST 入力でブランクのかわりに使用する区切り文字を指定します。使用できる *delimiter(s)*は次のとおりです。

'list-of-delimiting-characters'

区切り文字として読み込む文字を 1 つまたは複数指定します。

要件 指定する区切り文字のリストは引用符で囲む必要があります。

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

character-variable

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

別名 DLM=

デフォルト ブランク

ヒント 区切り文字では大文字と小文字が区別されます。

一般的な区切り文字としては、カンマ(,)、縦棒(|)、セミコロン(;)、タブが挙げられます。タブは 16 進数の文字で指定します。UNIX および Windows 環境では、タブの値は'09'x になります。z/OS 環境では、タブの値は'05'x になります。

参照項目 “区切り文字で区切られたデータ値の読み込み” (165 ページ)、
DLMSTR= (148 ページ)、および“DSD (delimiter-sensitive data)” (149 ページ)

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

DLMSTR= delimiter

LIST 入力でブランクのかわりに使用する区切り文字を指定します。使用できる *delimiter* は次のとおりです。

'delimiting-string'

区切り文字として読み込む文字列を指定します。

要件 文字列は引用符で囲みます。

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

character-variable

文字変数の名前を指定します。指定した変数の値が区切り文字として使用されます。

デフォルト ブランク

操作 INFILE ステートメントに DLMSTR=オプションを複数指定すると、最後に指定した DLMSTR=オプションが使用されます。DELIMITER=オプションと DLMSTR=オプションの両方を指定すると、最後に指定したオプションが使用されます。

RECFM=N を指定する場合は、サイズが大きい入力アイテムでも十分に保持できる値が LRECL に指定されていることを確認してください。指定し

た値が十分ではない場合、区切り文字によってレコード境界間で分割される場合があります。

ヒント 区切り文字では大文字と小文字が区別されます。区切り文字で大文字小文字を区別するには、DLMSOPT='I'オプションを使用します。

参照項目: “区切り文字で区切られたデータ値の読み込み” (165 ページ)、
DELIMITER= (147 ページ)、 DLMSOPT= (149 ページ)、 および DSD (149 ページ)

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

DLMSOPT= 'options'

DLMSTR=オプションの解析オプションを指定します。ここでは、*option(s)*は次の値のいずれかになります。

I

大文字小文字を区別せずに比較を行うことを指定します。

T

区切り文字の末尾にあるブランクを削除するよう指定します。

ヒント T オプションは、区切り文字として変数を使用する場合に便利です。

I、T、またはその両方を指定できます。

要件 DLMSOPT=オプションは、DLMSTR=オプションを使用している場合にのみ有効です。

参照項目: DLMSTR= (148 ページ)

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

DSD (delimiter-sensitive data)

データ値が引用符で囲まれている場合、そのデータ値内の区切り文字を文字データとして扱うことを指定します。DSD オプションを指定すると、LIST 入力を使用する場合の SAS による区切り文字の扱い方を変更し、デフォルトの区切り文字をカンマに変更できます。DSD を指定すると、SAS は 2 つの連続する区切り文字を欠損値として扱い、文字値から引用符を削除します。

操作 区切り文字を変更するには、DELIMITER=または DLMSTR=オプションを使用します。

ヒント 引用符に囲まれている文字列内に区切り文字を含んでいる文字値を読み込むには、DSD オプションと LIST 入力を使用します。INPUT ステートメントは、そのような区切り文字を有効な文字として扱い、その値を格納する前に、文字列から引用符を削除します。引用符を保持するには、チルダ (~)フォーマット修飾子を使用します。

参照項目: “区切り文字で区切られたデータ値の読み込み” (165 ページ)、
DELIMITER= (147 ページ)、 および DLMSTR= (148 ページ)

例 “例 1: 区切り文字の扱い方の変更” (170 ページ)

“例 2: 欠損値と短いレコードをリスト入力で処理する” (172 ページ)

ENCODING= 'encoding-value'

外部ファイルからの読み取り時に使用するエンコーディングを指定します。
ENCODING=の値は、外部ファイルのエンコーディングが現在のセッションエンコーディングとは異なることを示しています。

外部ファイルからデータを読み込む場合は、指定したエンコーディングからセッションエンコーディングにデータがトランスコードされます。

デフォルト SAS では、外部ファイルのエンコーディングがセッションエンコーディングと同じであるとみなします。

参照項目: エンコーディングの有効な値については、“[SAS 言語要素中のエンコーディング値](#)” ([SAS 各国語サポート\(NLS\): リファレンスガイド](#))を参照してください。

例 “[例 11: 外部ファイル読み込み時のエンコードの指定](#)” (180 ページ)

END=variable

変数名を指定します。現在の入力データレコードが最後である場合、この変数に値 1 が設定されます。SAS が最後のデータレコードを処理するまで、END=変数には値 0 が設定されます。自動変数と同じように、この変数はデータセットに書き込まれません。

制限事項 END=オプションは、UNBUFFERED オプションと一緒に使用できません。また、複数の入力データレコードを読み込む DATALINES ステートメント、DATALINES4 ステートメント、または INPUT ステートメントでは使用できません。

ヒント END=が無効である場合、オプション [EOF= \(150 ページ\)](#)を使用します。

例 “[例 5: 複数の外部ファイルからの読み込み](#)” (174 ページ)

EOF=label

ステートメントラベルを指定します。これは、INFILE ステートメントがファイルの終わりに達した場合に明示的な GOTO のオブジェクトとなります。INPUT ステートメントが、それ以上レコードを含んでいないファイルから読み込もうとする場合、このステートメントラベルに実行が移動されます。

操作 UNBUFFERED オプションと一緒に使用する場合や、複数の入力データレコードを読み込む DATALINES ステートメント、DATALINES4 ステートメントや INPUT ステートメントと組み合わせて使用する場合、END=オプションのかわりに EOF=を使用します。

ヒント EOF=オプションは、複数の入力ファイルを順番に読み込む場合に便利です。

参照項目: [END= \(150 ページ\)](#)、[EOV= \(150 ページ\)](#)、および [UNBUFFERED \(158 ページ\)](#)

EOV=variable

変数名を指定します。一連の連結ファイルに含まれている各ファイルの最初のレコードが読み込まれた場合、この変数に値 1 が設定されます。この変数は、SAS が次のファイルを検出した場合にのみ値が設定されます。自動変数と同じように、に EOV=に指定した変数はデータセットに書き込まれません。

ヒント SAS が各ファイル境界を検出した時点で、EOV=変数は 0 にリセットされます。

参照項目: [END= \(150 ページ\)](#) および [EOF= \(150 ページ\)](#)

EXPANDTABS | NOEXPANDTABS

タブ文字を標準タブ設定に拡張するかどうかを指定します。EXPANDTABS を指定すると、タブ文字が、列 9 から始まる 8 列の間隔に設定されます。

デフォルト NOEXPANDTABS

ヒント EXPANDTABS オプションは、お使いの動作環境にネイティブなタブ文字を含むデータを読み込む場合に便利です。

FILENAME=variable

変数名を指定します。この変数には、現在開かれている入力ファイルの物理名が設定されます。一連の連結ファイルでは、この変数は、SAS が次のファイルを検出した場合にのみ更新されます。自動変数と同じように、FILENAME=に指定した変数はデータセットに書き込まれません。

ヒント LENGTH ステートメントを使用すると、物理ファイル名を格納できる長さにまで変数の長さを拡張できます。

参照項目: [FILEVAR= \(151 ページ\)](#)

例 “例 5: 複数の外部ファイルからの読み込み” (174 ページ)

FILEVAR=variable

変数名を指定します。この変数の値が変化すると、INFILE ステートメントは現在の入力ファイルを閉じ、新しいファイルを開きます。次の INPUT ステートメントが実行されると、FILEVAR=変数に指定された新しいファイルから読み込みが実施されます。自動変数と同じように、この変数はデータセットに書き込まれません。

制限事項 FILEVAR=変数には、物理ファイル名を表す文字列を指定する必要があります。

操作 FILEVAR=オプションを使用すると、*file-specification* は、実際のファイル名またはファイルに対して事前に割り当てられたファイル参照名ではなく、プレースホルダーになります。このプレースホルダーを使用して、処理情報を SAS ログに出力します。プレースホルダーには、ファイル参照名と同じルールが適用されます。

ヒント FILEVAR=オプションを使用すると、現在開かれている入力ファイルを新しい物理ファイルに動的に変更できます。

FILEVAR=オプションを使用する場合、現在開かれている入力ファイルが最後のファイルかどうかを知ることはできません。DATA ステップがファイル終端マーカーまたはすべての開かれているデータセットの終わりに到達すると、シャットダウンが行われます。また、FILEVAR を FIRSTOBS と併用すると、複数のファイルの中でヘッダーレコードのみを含むファイルによって、DATA ステップの通常のシャットダウンがトリガーされます。このシャットダウンは、ファイル終端指示子を超えて読み込みを行うと

DATA ステップが終了するために発生します。EOF=オプションを使用すると、シャットダウンを回避することができます。

参照 項
目: “外部ファイルの直接更新” (164 ページ)

例 “例 5: 複数の外部ファイルからの読み込み” (174 ページ)

FIRSTOBS=*record-number*

レコード番号を指定します。SAS はこの番号を使用して、入力ファイル内の入力データレコードの読み込みを開始します。

デフ 1
ォル
ト

ヒント FIRSTOBS=と OBS=を組み合わせることで、ファイル内に含まれている特定の範囲のレコードを読み込むことができます。

ファイル内のヘッダーレコードをスキップするには、FIRSTOBS=2 を使用します。

例 次のステートメントでは、50 番目から 100 番目までのレコードを処理します。
`infile file-specification firstobs=50 obs=100;`

FLOWOVER

INPUT ステートメントが、ステートメント内にあるすべての変数に関して、現在の入力行内に値を検出しなかった場合、次の入力データレコードを継続して読み込むようにします。FLOWOVER は、INPUT ステートメントのデフォルトの動作です。

参照項目: “行の終わりを越えて読み込む” (169 ページ)、MISSOVER (154 ページ)、STOPOVER (157 ページ)、および TRUNCOVER (158 ページ)

LENGTH=*variable*

変数名を指定します。この変数には、現在の入力ファイルの長さが設定されます。SAS は、INPUT ステートメントが実行されるまで、この変数に値を割り当てません。自動変数と同じように、LENGTH=に指定した変数はデータセットに書き込まれません。

ヒント このオプションは、フィールド幅が異なる場合に、\$VARYING 入力形式と組み合わせると便利です。

例 “例 4: 可変長レコードを含むファイルの読み込み” (174 ページ)

“例 7: コピーされたレコードの切り捨て” (176 ページ)

LINE=*variable*

変数名を指定します。この変数には、入力バッファ内の入力ポインタの行位置が設定されます。自動変数と同じように、LINE=に指定した変数はデータセットに書き込まれません。

範囲 1 から N=オプションに指定された値

操作 LINE=変数の値は、INPUT ステートメントで N=オプションまたは#*n* 行ポインターコントロールによって指定された行グループ内の現在の相対行番号です。

参照項目: [COLUMN= \(147 ページ\)](#)および [N= \(154 ページ\)](#)

例 [“例 8: ポインター位置の出力” \(177 ページ\)](#)

LINESIZE=*line-size*

INPUT ステートメントで使用可能なレコード長を指定します。

別名 LS=

範囲 最大 32767

操作 INPUT ステートメントが、LINESIZE=オプションで指定された列を超えて読み込もうとした場合、FLOWOVER、MISSOVER、SCANOVER、STOPOVER、TRUNCOVER のいずれのオプションが有効であるかによって、実行されるアクションが異なります。FLOWOVER がデフォルトです。

動作環境 *line-size* の値は、動作環境のレコードサイズによって異なります。詳細については、使用している動作環境に対応する SAS ドキュメントを参照してください。

ヒント LINESIZE=を使用すると、レコード全体を読み込みたくない場合にレコード長を制限できます。

例 データ行の 73 列目から 80 列目までに順序番号が含まれている場合、次の INFILE ステートメントを使用すると、INPUT ステートメントによる読み込みを先頭から 72 列目までに制限できます。
infile *file-specification* linesize=72;

LRECL=*logical-record-length*

論理レコード長を指定します。

デフォルト LRECL=のデフォルト値は 32,767 です。固定長レコード(RECFM=F)を使用する場合、LRECL=のデフォルト値は 256 になります。最大値は 1G です。

制限事項 DATALINES ファイル指定を使用する場合、LRECL は無効となります。

操作 代わりに、“[LRECL= 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](#))を使用すると、グローバルな論理レコード長を指定できます。グローバル LRECL システムオプションのデフォルト値は 32767 です。固定長レコード(RECFM=F)を使用する場合、LRECL のデフォルト値は 256 になります。

RECFM=F の場合、LRECL=オプションの値により、1 レコードとして読み取られるバイト数が決定されます。RECFM=N の場合、LRECL=オプションの値は 256 以上にする必要があります。RECFM=V の場合、LRECL=オプションの値により最大レコード長が決定されます。指定された長さを超えるレコードは切り捨てられます。

ヒン LRECL=オプションには、ファイルの物理的な行の長さを指定します。
 ト LINESIZE=オプションには、INPUT ステートメントが読み込む行の長さを指定します。

MISSOVER

INPUT ステートメントが、ステートメント内にあるすべての変数に関して、現在の入力行内に値を検出しなかった場合、新しい入力データレコードを読み込まないようにします。INPUT ステートメントが現在の入力データレコードの終端に到達すると、何の値も割り当てられていない変数には欠損値が設定されます。

ヒント 最後のフィールドが欠損値であり、対応する変数に欠損値を割り当てたい場合には、MISSOVER を使用します。

参照項 “行の終わりを超えて読み込む” (169 ページ)、FLOWOVER (152 ページ)、
 目: SCANOVER (156 ページ)、STOPOVER (157 ページ)、および
 TRUNCOVER (158 ページ)

例 “例 2: 欠損値と短いレコードをリスト入力で処理する” (172 ページ)

MEMVAR=variable

ディレクトリに含まれる、または ZIP ファイルなどのディレクトリ型ファイルに含まれる個々のファイルを開くことができます。ディレクトリ fileref や物理名が INFILE ステートメントで指定されると、MEMVAR=変数で参照されているファイルが開かれます。MEMVAR=変数の値が変わると、現在のファイルが閉じられ、新しいファイルが開かれます。ディレクトリのすべてのメンバーを読み込むには、MEMVAR=変数を空白に設定します。

制限 MEMVAR=変数の値には、物理ファイル名を含む文字列が含まれている必要
 事項 がある、または値には空白のみが含まれている必要があります。
 MEMVAR=変数の値が空白に設定されている場合、INFILE ステートメントは、現在のファイルを閉じ、ディレクトリからメンバー名を取得し、この名前を MEMVAR=変数に入れて、新しいファイルを開きます。

ヒン この変数は、自動変数と同じように、データセットに書き込まれません。
 ト

文字変数のデフォルトの長さは 8 文字です。8 文字を超える物理ファイル名がある場合、LENGTH ステートメントや INPUT ステートメントなどの他のステートメントを使用して、MEMVAR=オプションの変数に十分な長さを割り当ててください。

参照 FILENAME= (113 ページ)
 項目:

例 “例 12: ディレクトリのすべてのファイルメンバーの読み込み” (180 ページ)

“例 13: ファイルメンバーのリストの読み込み” (181 ページ)

N=available-lines

入力ポインターで一度にアクセスできる行数を指定します。

デフォルト DATA ステップ内の任意の INPUT ステートメントでの#ポインターコントロールの後に指定されている最大数。#ポインターコントロールを省略した場合、デフォルト値は 1 になります。

操作 このオプションは、ポインターで一度にアクセスできる行数にのみ影響します。INPUT ステートメントが読み込む行数には影響しません。

ヒント N=の値よりも小さな値の#ポインターコントロールを INPUT ステートメントで使用すると、予期しない結果が発生することがあります。予期しない結果を防ぐには、N=オプションの値に等しい#ポインターコントロールを含めるようにします。次に例を示します。

```
infile 'external file' n=5;
input #2 name : $25. #3 job : $25. #5;
```

レコードからデータが読み込まれなくても、INPUT ステートメントには #5 ポインターコントロールが含まれます。

例 “例 8: ポインター位置の出力” (177 ページ)

NBYTE=variable

変数名を指定します。この変数には、ストリームレコード形式でデータを読み込む場合に(FILENAME ステートメントで RECFM=S を指定)、ファイルから読み込むバイト数が含まれます。

デフォルト ファイルの LRECL 値

操作 読み込むバイト数を-1 に設定すると、FTP および SOCKET アクセスメソッドは、入力バッファ内で現在使用可能なバイト数を返します。

参照項目: FILENAME ステートメントの SOCKET アクセスメソッドでの [RECFM=オプション](#)、および FILENAME ステートメントの FTP アクセスメソッドでの [RECFM=オプション](#)

OBS=record-number | MAX

record-number では、順次読み込みの入力ファイルから読み込む最終レコードのレコード番号を指定します。

MAX では、処理するオブザベーションの最大数を指定します。この値は、少なくとも符号付き 32 ビット整数の最大値と同じ大きさになります。絶対最大値は、お使いの動作環境により異なります。

デフォルト MAX

ヒント OBS=と FIRSTOBS=を組み合わせることで、ファイル内に含まれている特定の範囲のレコードを読み込むことができます。

例 次のステートメントは、ファイルに含まれている先頭から 100 レコードのみを読み込みます。

```
infile file-specification obs=100;
```

PAD | NOPAD

LRECL=オプションに指定された長さに達するまで、外部ファイルから読み込むレコードに空白を追加するかどうかを指定します。

デフォルト NOPAD

参照項目: [LRECL=オプション \(153 ページ\)](#)

PRINT | NOPRINT

入力ファイルにキャリッジコントロール文字が含まれているかどうかを指定します。

ヒント キャリッジコントロール文字を削除する必要なしに DATA ステップでファイルを読み込む場合は、PRINT を指定します。キャリッジコントロール文字をデータ値として読み込む場合は、NOPRINT を指定します。

RECFM=record-format

入力ファイルのレコード形式を指定します。取り得る値は次のとおりです。

- D デフォルトのレコード形式(可変長形式)。
- F 固定長形式。つまり、各レコードの長さは同じです。
- N バイナリ形式。ファイルはバイトストリームで構成され、レコード境界はありません。LRECL オプションを指定しない場合、デフォルトではファイルから 256 バイトが読み込まれます。
- P プリント形式。
- V 可変長形式。各レコードは改行文字で終わります。

RESET=variable

現在の入力ファイルを閉じて新しいファイルを開くか、現在の入力ファイルを閉じて再び開くかを指定します。variable 値が 0 の場合、FILEVAR=変数の値が変わると現在のファイルは閉じられ、新しいファイルが開かれます。variable 値が 1 のとき、現在のファイルは閉じられ、FILEVAR=変数で参照されているファイルが再び開かれます。

デフォルト FILEVAR=変数の値が変化すると、現在のファイルが閉じられ、新しいファイルが開かれます。

SCANOVER

INPUT ステートメントが、@'character-string'式に指定された文字列を検出するまで、入力データレコードをスキャンするようにします。

操作 MISCOVER、TRUNCOVER、および STOPOVER オプションを使用すると、INPUT ステートメントが@'character-string'式をスキャンしてレコードの終端に到達した場合に、その INPUT ステートメントがどのように動作するかを変更できます。デフォルト(FLOWOVER オプション)では、INPUT ステートメントは次のレコードをスキャンします。それ以外のオプションでは、INPUT ステートメントはスキャンを中止します。

ヒント SCANOVER と FLOWOVER の両方を指定することは冗長です。

- 参照 “行の終わりを超えて読み込む” (169 ページ)、FLOWOVER (152 ページ)、
 項目: MISCOVER (154 ページ)、STOPOVER (157 ページ)、および TRUNCOVER
 (158 ページ)
- 例 “例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする”
 (173 ページ)

SHAREBUFFERS

FILE ステートメントと INFILE ステートメントが同じバッファを共有することを指定します。

別 名 SHAREBUFS

ヒ INFILE、FILE、PUT ステートメントで SHAREBUFFERS オプションを使用
 シ すると、外部ファイルを直接更新できます。外部ファイルを直接更新する
 ト と、PUT ステートメント出力が出力バッファを経由せずに入力バッファ
 ーから外部ファイルへと直接書き込まれるため、CPU 時間を節約できます。

また、SHAREBUFFERS を使用すると、レコード全体ではなく、外部ファイル内の特定のフィールドを更新できます。

例 “例 6: 外部ファイルの更新” (176 ページ)

注 **SHAREBUFFERS、RECFM=V、_INFILE_を使用する際、同一の長さのレコ**
 意 **ードを読み込んだ後、異なる長さのレコードを含むファイルを更新する場**
合には注意が必要です。 レコードの長さを変更するには、_INFILE_を使用
 します。この潜在的な問題を回避する方法の 1 つとして、元々のレコード
 長が維持されるように _INFILE_ を追加または削除することが考えられま
 す。

START=variable

変数名を指定します。この変数の値は、PUT _INFILE_ステートメントが書き込む
 レコードの最初の列番号として使用されます。自動変数と同じように、START 変
 数はデータセットに書き込まれません。

参照項目: PUT ステートメントの _INFILE_ オプション (291 ページ)

STATUS=variable

値に URL リクエストからのリターンステータスコードを持つ変数を指定しま
 す。

参照項目: “FILENAME ステートメント: URL アクセスメソッド” (130 ページ)

例 “例 14: URL リクエストからのリターンコードの読み込み” (181 ペ
 ジ)

STOPOVER

INPUT ステートメントで、ステートメント内のすべての変数に対する値が見つ
 けられずに現在のレコードの終わりに達した場合、DATA ステップの処理が停止しま
 す。期待された数の値が入力行に含まれていない場合、STOP ステートメントが
 実行されないならば、SAS は _ERROR_ に値 1 を設定してデータセットの作成を中
 止し、不完全なデータ行を出力します。

ヒント デフォルトの動作に戻すには FLOWOVER を使用します。

参照項目: “行の終わりを越えて読み込む” (169 ページ)、FLOWOVER (152 ページ)、MISSOVER (154 ページ)、SCANOVER (156 ページ)、および TRUNCOVER (158 ページ)

例 “例 2: 欠損値と短いレコードをリスト入力で処理する” (172 ページ)

TERMSTR=

Linux でフォーマットされたファイル中の行末区切り文字を制御します。デフォルトでは、ラインフィードのみ、またはキャリッジリターンとラインフィードが行の終わりを示します。行末区切り文字を明示的に指定するには、次のいずれかの値を指定します。

CR キャリッジリターン。

CRLF キャリッジリターン、ラインフィード。

LF ラインフィード。このパラメーターは、Linux でフォーマットされたファイルの読み込みに使用されます。

TRUNCOVER

入力データレコードの長さが INPUT ステートメントの期待する長さよりも短い場合の INPUT ステートメントのデフォルトの動作を無効化します。デフォルトでは、INPUT ステートメントは次の入力データレコードを自動的に読み込みます。TRUNCOVER を指定すると、INPUT ステートメントの期待する長さよりも短いレコードが存在する場合、可変長のレコードを読み込むことができます。値を割り当てられていない変数には、欠損値が設定されます。

ヒント TRUNCOVER を使用すると、フィールドの長さが期待された長さより短い場合、入力バッファの内容を変数に割り当てることができます。

参照項目: “行の終わりを越えて読み込む” (169 ページ)、FLOWOVER (152 ページ)、MISSOVER (154 ページ)、SCANOVER (156 ページ)、および STOPOVER (157 ページ)

例 “例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする” (173 ページ)

UNBUFFERED

バッファされた(先行)読み込みを実施しないよう指示します。

別名 UNBUF

操作 UNBUFFERED を指定すると、SAS は END=変数に値 1 を設定しなくなります。

ヒント DATALINES ステートメントを使用してインストリームデータを読み込む場合、UNBUFFERED が有効となります。

INFILE=variable

この INFILE ステートメントの現在の入力バッファの内容を参照する文字変数を指定します。この変数は他の変数と同じように使用できます。また、値を割り当てることもできます。この変数の値は自動的に保持されます。初期値はブラ

ンクです。自動変数と同じように、_INFILE_に指定した変数はデータセットに書き込まれません。

制限事項 *variable* には、すでに定義済みの変数は指定できません。_INFILE_=オプションには、その DATA ステップ内で初めて使用する変数を指定する必要があります。LENGTH ステートメントや ATTRIB ステートメントを使用して、_INFILE_に指定した変数の長さを設定したり変更することはできません。ただし、ATTRIB ステートメントや FORMAT ステートメントを使用して、この変数に出力形式を指定することができます。

操作 この文字変数の最大長は、指定した INFILE ステートメントの論理レコード長(LRECL= (153 ページ))になります。ただし、プログラムの実行直前まで、SAS がこのファイルを開いて LRECL=オプションの値を確認することはありません。そのため、コンパイル実行中は、この変数のサイズは 32,767 バイトになります。

ヒント この変数の内容に対する変更は、INFILE ステートメントの現在の入力バッファの内容にただちに反映されます。バッファの変更後に、その INFILE ステートメントが有効なスコープで PUT _INFILE_ステートメントを実行すると、変更されたバッファの内容が反映されます。N=オプションで複数のバッファを指定した場合でも、_INFILE_に指定した変数は、指定した INFILE ステートメントの現在の入力バッファにのみアクセスできます。

別のステートメントで _INFILE_=オプションを使用せずに入力バッファの内容にアクセスするには、自動変数 _INFILE_ を使用します。

_INFILE_変数は固定長ではありません。_INFILE_変数に値を割り当てると、その変数の長さは、割り当てられた値の長さへと変更されます。

参照項目: “入力バッファのコンテンツへのアクセス” (164 ページ)

例 “例 9: 入力バッファ内のレコードを操作する” (177 ページ)

“例 10: 複数のファイルの入力バッファへのアクセス” (179 ページ)

DBMS 指定

DBMS-Specifications

いくつかの DBMS ファイルからレコードを読み込むことができます。使用している DBMS の SAS/ACCESS ドキュメントを参照してください。(SAS/ACCESS for Relational Databases: [リファレンス](#)または SAS/ACCESS for Nonrelational Databases: [リファレンス](#))

詳細

INFILE ステートメントの使用

INFILE ステートメントは読み込み対象となるファイルを指定するものであるため、入力データレコードを読み込む INPUT ステートメントよりも前に実行する必要があります。INFILE ステートメントは実行ステートメントであるため、IF-THEN ステートメントのような条件付き処理で INFILE ステートメントを使用できます。INFILE ステートメントを使用すると、入力データレコードのソースを制御できます。

通常、INFILE ステートメントは、外部ファイルからデータを読み込む場合に使用します。データをジョブストリームから読み込む場合、ATALINES ステートメントを使用する必要があります。ただし、INFILE ステートメントでのみ使用可能なデータ読み込みオプションを使用する場合、同じ DATA ステップ内で、ATALINES および ATALINES ステートメントのファイル指定を含む INFILE ステートメントを使用できます。詳細については、“[長いインストリームデータレコードの読み込み](#)” (168 ページ)を参照してください。

複数の INFILE ステートメントを使用して同じファイル指定を行い、各 INFILE ステートメントでオプションを使用した場合、それらのオプションが追加されます。混乱を避けるために、特定の外部ファイルに関するオプションはすべて最初の INFILE ステートメントで指定するようにしてください。

複数の外部ファイルの読み込み

DATA ステップの 1 回の反復で複数の入力ファイルの読み込みを行うには、次の 2 つの方法のうちどちらかを使用します。

- 複数の INFILE ステートメントを指定することで、複数のファイルを開いておき、どのファイルを読み込むかを変えることができます。DATA ステップは、DATA ステップの反復ごとに、複数の入力ファイルから読み込みます。SAS は 1 回の反復であるファイルを次のファイルに切り替えるため、各ファイルの最初の行が読み込まれるまで、各ファイルは開いたままになります。入力ポインターはファイルごとに現在の位置のまま移動せず、次回 DATA ステップの反復で INPUT ステートメントを実行すると、ファイルの読み込みがその位置から開始されます。
- INFILE ステートメントで [FILEVAR=オプション](#)を指定して、単一の DATA ステップの反復内で現在の入力ファイルを動的に変更します。FILEVAR=オプションは、DO ループ内の 2 番目の INPUT ステートメントを使用して、ファイルを開きその行をすべて読み込みます。その後、ファイルを閉じて、ATALINES ステートメントに記載されている次のファイルを開き、というように、すべてのファイルが読み込まれるまで繰り返します。

関連項目

“[例 5: 複数の外部ファイルからの読み込み](#)” (174 ページ)。

FILEVAR=オプションによる複数の外部ファイルの読み込み

DATALINES ステートメントによるファイル名の読み込み

INFILE ステートメントの **FILEVAR=オプション** は、複数の外部ファイルを 1 つの SAS データセットに読み込むためのシンプルで効率的な手段を提供します。この例では、SAS は複数の INFILE ステートメントと **DO ループ** を使用して、3 つの外部ファイル(file1、file2、file3)を出力データセット output に読み込みます。外部ファイルの内容を次に示します。

図 2.1 外部テキストファイルの内容

file1	file2	file3
A 1	D 4	G 7
B 2	E 5	H 8
C 3	F 6	I 9

この例では、最初の INFILE ステートメントで **DATALINES オプション** を指定して、SAS に入力データの場所(DATALINES ステートメントではファイル名のリストとして)を伝えています。2 つ目の INFILE ステートメントでは、SAS が DATALINES から読み込んだファイル名を一時変数 file2read に格納します。次の DATA ステップの繰り返しでは、DATALINES に記載された次のファイル名が file2read に格納され、INPUT ステートメントで開かれます。この繰り返しは、DATALINES に記載されている最後のファイルが読み込まれるまで続きます。

```
data output;
  length file2read $40;          /* 1 */
  input file2read $;             /* 2 */
  infile datalines;              /* 3 */
  infile dummy filevar = file2read end = done; /* 4 */

  do while(not done);            /* 5 */
    input Var1 $ Var2;
    output;                      /* 6 */
  end;
  datalines;                     /* 7 */
  C:\Users\sasuser\file1
  C:\Users\sasuser\file2
  C:\Users\sasuser\file3
  ;
proc print data=output; run;
```

- 1 **LENGTH ステートメント** を指定して、file2read という変数を新規作成します。変数 file2read は、DATALINES ステートメントで指定された外部ファイルの名前をまとめて表しています(file1、file2、file3)。各ファイルには、出力データセットに読み込まれる行が含まれています。
- 2 **INPUT ステートメント** を指定して、DATA ステップの各反復において、変数 file2read の値を読み込みます。
- 3 **DATALINES オプション** で最初の INFILE ステートメントを指定します。DATALINES オプションは、DATALINES ステートメントのファイルのリストから入力データを読み込むよう SAS に指示します。

- 4 2 番目の INFILE ステートメントを 3 つのオプション(**DUMMY**、**FILEVAR=**、および **END=**)で指定します。説明は次のとおりです。

DUMMY オプションは、INFILE ステートメントが慣習的なファイル参照を予想せず、代わりに **FILEVAR=** オプションを実行するようプレースホルダーの役割を果たします。filevar=file2read ステートメントを使用して、SAS はファイル名を変数 file2read に格納します。DATA ステップの各反復において、SAS は各外部ファイル名を読み込み、変数 file2read に格納します。

END= オプションでは、一時変数 done を指定します。**END=** オプションは、最初に変数 done を 0 に設定します。これは、DATA ステートメント読み込む行がまだあることを示します。DATA ステップは、現在開いているファイルの最終行(ファイルの終わり)に到達すると、**END=** 変数を 1 に設定し、これ以上読み込む行がないことを示します。この一時変数は、最後のファイルの最後のレコードを読み込んだときに、DO ループが DATA ステップ処理を停止する条件として使われます。

注: SAS では、最後の外部ファイルの最後のレコードを読み込んだときに、done を 1 に設定します。

注: **END=** 変数値(done)と **FILEVAR=** 変数(file2read)は出力データセットに書き込まれません。

- 5 **DO WHILE** ステートメントを、done 変数を条件として指定します。変数 done が 1 に等しくない限り、DATA ステップは現在開いているデータセットの行を処理し続けます。DATA ステップが **DATALINES** ステートメントに記載されている最後のファイルの終わりを読み込むと、SAS は done を 1 に設定し、DO ループは現在のファイルからの行の読み込みを停止し、DATA ステップに制御を戻します。その後、DATA ステップは再び繰り返し、file2read の次の値(**DATALINES** ステートメントの次のファイル名)を読み込みます。
- 6 **OUTPUT** ステートメントを指定します。
- 7 **DATALINES** ステートメントには、読み込む外部ファイルのリストが含まれています。

アウトプット 2.12 出力 SAS データセット

Obs	Var1	Var2
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6
7	G	7
8	H	8
9	I	9

INPUT ステートメントを含む **DO ループ** を使用していない場合、DATA ステップは各外部ファイルの最初のレコードのみを読み込みます。これは、DATA ステップが、INPUT ステートメントがファイルの最後のレコードを読み込んだ時点で、ファイルの処理を自動的に停止するためです。最後のレコードが読み込まれると、DO の処理は停止し、DO ループに続く次のステートメントに制御が移ります。DATA ステップ処理の詳細については、[DATA ステップでの入力データの処理方法](#)を参照してください。

DO ループは、通常、**DO WHILE** または **DO UNTIL** ステートメントの形式をとります。この 2 種類のループについて、以下の点に注意してください。

- DO UNTIL は、DO ループの終わりに条件を評価します。その結果、DO UNTIL ステートメントは、常に 1 回以上ループコードを実行します。読み込み中のファイルのいずれかが空の場合、DO UNTIL により DATA ステップは早期に終了します。
- DO WHILE は、DO ループの先頭に条件を評価します。その結果、DO WHILE ステートメントではループ内のコードが全く実行されない場合があります。読み込み中のファイルのいずれかが空の場合、DO WHILE により DATA ステップが早期に終了することはありません。

FILEVAR=オプションで IF や DELETE を使うときの注意点

サブセット IF 文と **DELETE ステートメント**は、FILEVAR=オプションと併用すると予想外の結果になることがあります。

DO ループ内の IF ステートメントの条件が満たされない場合、SAS は DATA ステップの現在の反復処理を停止し、DO ループの後の最初の実行可能なステートメントに処理制御を戻します。**DELETE ステートメント**にも同様の注意点があります。SAS が DATA ステップの現在の反復処理を停止すると、DO ループによるレコードの読み込みは終了し、現在のファイルは閉じられ、次のファイルが処理のために開かれます。

関連項目

- [“DATA ステップのデータの処理方法” \(SAS プログラマガイド: エッセンシャル\)](#)
- [“IF ステートメント” \(SAS プログラマガイド: エッセンシャル\)](#)

SAS データセットからの複数のファイル名の読み込み

DATALINES ステートメントではなく、SAS データセットに入力ファイル名を記載することができます。そのためには、SET ステートメントと INFILE FILEVAR ステートメントを併用して、データセットから入力ファイルを読み込みます。

```
data one;
  set two;
  infile dummy filevar= file2read end=done;
  do while(not done);
    input var1 $ var2;
    output;
  end;
run;
```

関連項目

[“例 5: 複数の外部ファイルからの読み込み” \(174 ページ\)](#)

外部ファイルの直接更新

INFILE ステートメントを FILE ステートメントと使用することで、外部ファイルに含まれているレコードを更新できます。

- 1 FILE ステートメントの前に INFILE ステートメントを指定します。
- 2 各ステートメントで、同一のファイル参照名または物理ファイル名を指定します。
- 3 FILE ステートメントのかわりに、INFILE ステートメントに、FILE ステートメントと INFILE ステートメントの両方で使用できるオプションを使用します。(これらのオプションを FILE ステートメントに指定しても無視されます。)

[“例 6: 外部ファイルの更新” \(176 ページ\)](#)を参照してください。

レコード全体ではなく、レコード内の個々のフィールドを更新する方法については、[SHAREBUFFERS オプション \(157 ページ\)](#)を参照してください。

入力バッファのコンテンツへのアクセス

`_INFILE_`変数を使用する以外に、`_INFILE_`自動変数を使用することでも、最後に実行した INFILE ステートメントの入力バッファの内容を参照できます。この文字変数の値は自動的に保持されます。初期値はブランクです。他の自動変数と同じように、`_INFILE_`はデータセットに書き込まれません。

INFILE ステートメントで `_INFILE_`オプションを指定すると、変数 `_INFILE_`の値を自動変数 `_INFILE_`でも間接的に参照できます。自動変数 `_INFILE_`が存在する場合に、特定の INFILE ステートメントで `_INFILE_`を省略すると、その INFILE ステートメント用の内部的な `_INFILE_`変数が作成されます。これ以外の場合に、特定の FILE に対して `_INFILE_`変数が作成されることはありません。

実行時または参照時に、この文字変数の最大長は、現在の `_INFILE_`変数の最大長と一致します。ただし、`_INFILE_`が参照している他の変数の長さは実行時まで確定されないため、`_INFILE_`の長さはコンパイル時に 32,767 バイトに設定されます。たとえば、`_INFILE_`を長さが定義されていない新しい変数に割り当てる場合、新しい変数のデフォルトの長さは 32,767 バイトになります。LENGTH ステートメントや ATTRIB ステートメントを使用して、`_INFILE_`の長さを設定または無効化することはできません。ただし、FORMAT ステートメントや ATTRIB ステートメントを使用すると、`_INFILE_`に出力形式を割り当てることができます。

他の SAS 変数と同じように、割り当てステートメントを使用して、`_INFILE_`変数を更新することができます。また、PUT ステートメントの `_INFILE_`でも出力形式を使用できます。たとえば、次の PUT ステートメントでは、16 進数の出力形式を使用して入力バッファの内容を出力します。

```
put _infile_ $hex100.;
```

`_INFILE_`に変更を加えると、現在の INFILE ステートメントの入力バッファにただちに変更が反映されます。バッファの内容を変更した後、PUT `_INFILE_`ステートメントを実行すると、変更したバッファの内容が出力されます。

N=オプションに複数のバッファを指定した場合でも、_INFILE_は、その INFILE ステートメントで現在使用中の入力バッファの内容にのみアクセスします。N=に指定したすべてのバッファにアクセスできますが、特定のバッファを現在の入力バッファに設定するには、INPUT ステートメントに#行ポインターコントロールを指定する必要があります。

区切り文字で区切られたデータ値の読み込み

デフォルトでは、リスト入力での入力データレコードの読み込みに使用される区切り文字は、1つの空白になります。DSD (delimiter-sensitive data)オプション、DELIMITER=オプション、DLMSTR=オプション、DLMSOPT=オプションは、リスト入力での区切り文字の取り扱い方法に影響を与えます。DELIMITER=オプションまたは DLMSTR=オプションは、INPUT ステートメントで、リスト入力により読み取られるデータ値の区切り文字として、空白以外の文字を使用することを指定します。DSD オプションを有効にすると、INPUT ステートメントではデフォルトの区切り文字としてカンマを使用します。

また、DSD オプションを指定すると、2つの連続する区切り文字が欠損値として読み込まれるようになります。デフォルトでは、INPUT ステートメントは、連続する区切り文字を1つの単位として扱います。DSD オプションを指定すると、INPUT ステートメントは、連続する区切り文字をそれぞれ独立したものとして扱うようになります。このため、2つの連続する区切り文字で囲まれた空の文字列は欠損値として読み込まれます。区切り文字をカンマから別の値へと変更する場合、DELIMITER=オプションまたは DLMSTR=オプションを使用します。

たとえば、次の DATA ステッププログラムでは、リスト入力を使用して、カンマで区切られたデータを読み込みます。2番目のデータ行には欠損値が含まれています。SAS では、リスト入力での連続する区切り文字の使用を許可しているため、この INPUT ステートメントでは欠損値を検出できません。

```
data scores;
  infile datalines delimiter=';';
  input test1 test2 test3;
  datalines;
91,87,95
97,,92
,1,1
;
```

FLOWOVER オプションが有効である場合、データセット SCORES には、3つではなく2つのオブザベーションが含まれます。2番目のオブザベーションは正しく作成されません。

OBS	TEST1	TEST2	TEST3
1	91	87	95
2	97	92	1

この問題を修正するには、INFILE ステートメントで DSD オプションを使用する必要があります。

```
data scores;
  input test1 test2 test3;
  datalines;
```

```

91,87,95
97,,92
,1,1
;
infile datalines dsd;

```

今回の INPUT ステートメントでは、2 つの連続する区切り文字を検出するため、2 番目のオブザベーションに含まれている変数 TEST2 には欠損値が割り当てられます。

OBS	TEST1	TEST2	TEST3
1	91	87	95
2	97	.	92
3	.	1	1

また、DSD オプションを指定すると、リスト入力で、引用符で囲まれた文字列内に区切り文字を含む文字値を読み込むことができますようになります。たとえば、データがカンマで区切られている場合、DSD オプションを指定すると、そのような文字列を引用符で囲むことで、同文字列に含まれているカンマを有効な文字として読むことができます。この場合の引用符は、文字値の一部としては保存されません。引用符を文字値の一部として保持するには、INPUT ステートメントでチルダ(~)フォーマット修飾子を使用します。“[例 1: 区切り文字の扱い方の変更](#)” (170 ページ)を参照してください。

注: テキストファイルがローカルのエンコーディング環境以外の場所で作成された場合、EBCDIC 環境または ASCII 環境のどちらかで ENCODING=オプションを指定する必要がある場合があります。たとえば、ASCII プラットフォーム上で EBCDIC ファイルを読み込む場合、INFILE ステートメントに ENCODING=オプションを指定することをお勧めします。ただし、INFILE ステートメントで DSD および DLM=オプションまたは DLMSTR=オプションを使用する場合、これらのオプションにはセッションエンコーディングで特定の文字(引用符、コンマ、ブランクなど)が含まれている必要があるため、ENCODING=オプションが必須です。エンコーディング固有の入力形式は、バイナリファイルに使用するためのものです(このようなバイナリファイルには、文字フィールドと文字以外のフィールドが含まれています)。

デフォルト長よりも大きい長さのデータを扱う場合、入力形式を指定する必要があります。たとえば、日付または時間値、名前、住所などは、8 文字よりも長い場合があります。そのような場合、INFORMAT ステートメントを DATA ステップに追加するか、または入力形式を INPUT ステートメントに直接追加します。入力形式を追加する場合は、入力形式の前にコロン(:)を付ける必要があります。コロンは、現在の区切り文字の後の最初の文字から、入力形式で指定された文字数までを読み込むよう SAS に指示します。

```

data a;
  infile 'c:\sas\example.csv' dlm='09'x dsd trunccover;
  input fname :$20. lname :$30. address1 :$50. address2 :$50. city :$40.
        state :$2. zip phone :$12.;
run;

```


区切り文字で区切られたデータを読み込む場合に考慮すべきその他の INFILE オプションを次に示します。

FIRSTOBS=

最初を読み込むオブザベーションの番号を指定します。たとえば、ヘッダーレコードをスキップする場合には、FIRSTOBS=2 と指定します。

TERMSTR=

行末文字を指定する場合に使用します。このオプションは、あるオペレーティングシステムで作成されたデータを、別のオペレーティングシステムと共有する場合に使用すると便利です。たとえば、UNIX 環境で作業している場合に、Windows 環境で作成されたファイルを読み込む必要があるならば、TERMSTR=CRLF と指定します。同様に、Windows 環境で作業している場合に、UNIX 環境で作成されたファイルを読み込む必要があるならば、TERMSTR=LF と指定します。

TRUNCOVER=

現在のレコード内にある使用可能な入力データのみを使用して、できるだけ多くの変数を割り当てるよう指定します。デフォルトでは、INPUT ステートメントが、リストされている変数に割り当てる十分な数のデータを見つけられずにレコードの終端を超えて読み込む場合、同ステートメントは次のレコードからデータを継続して読み込みます。次のレコードを読み込まないようにするには、TRUNCOVER=オプションを使用します。

トラブルシューティングの例をいくつか次に示します。

問題	解決策
SAS ログに次のようなメッセージが出力されており、データセット内のすべてのデータが読み込み用に表示されています。One or more lines were truncated.	多くの場合、これは問題にはなりません。 このメッセージは、以下のいずれかの条件が満たされたときに発生します。 ■ 最大レコード長がアクティブな論理レコード長よりも短い。 ■ すべてのデータがデータセットに入っている。 ■ FIRSTOBS=に 1 を超える値を使用している。 このメッセージは、行の長さが LRECL=の値よりも大きいために、FIRSTOBS=オプションによりその行がスキップされたことを意味します。SAS は、行の長さが LRECL=の値よりも大きいことを認識しますが、FIRSTOBS=オプションによりその行がスキップされたことは認識しません。
最後の変数が数値変数であり、すべてのオブザベーションで、この変数が欠損値となっています。 変数が欠損値となるよくある原因としては、改行文字としてラインフィード(LF)のみを使用する UNIX 環境で Windows ファイルを読み込んだことが挙げられます。UNIX オペレーティングシス	INFILE ステートメントで TERMSTR=CRLF オプションを指定します。 SAS®9 以前のリリースを使用している場合、ファイルを適切な UNIX 構造に変換するか、またはキャリッジリターン(<CR>)を区切り文字のリストに追加します。 CSV ファイルを読み込む場合、キャリッジリターンを DL DLM='2C0D'x として指定します。

問題	解決策
テムは、キャリッジリターン(CR)をデータとして読み込みます。データは数値データではないため、無効なデータであるとのメッセージが生成されます。	タブで区切られたファイルを読み込む場合、DLM='090D'x を使用します。 これら以外の区切り文字を使用している場合、SAS テクニカルサポートに連絡してください。
CSV ファイルを読み込むと、引用符で囲まれた文字列の内部でレコードが分割されます。たとえば、長いコメントフィールドが、読み込み中のファイル内の行の途中でいきなり中断され、残りの部分は次の行に続いている箇所があります。ファイルをテキストエディターで開くと、このことを確認できます。 この問題は、Microsoft Excel ワークシートからファイルを作成した場合によく発生します。Excel では、フォーマットを支援するために、列にソフトリターンが含まれています。ワークシートを CSV ファイルとして保存すると、このソフトリターンはレコード終端マークとして SAS に誤って表示され、レコードが誤って分割されます。	SAS®9 以降を使用する場合、Windows 環境で、INFILE ステートメントに TERMSTR=CRLF オプションを使用できます。この設定は、SAS が完全なリターンとラインフィードのみをレコード終端マークとして受け付けるよう指示します。 SAS®9 以前のリリースを使用している場合、次のプログラムを実行します。このプログラムは、二重引用符で囲まれているラインフィード文字を空白へと変換し、SAS がファイルを正しく読み込むことができるようにします。このプログラムでは、INFILE ステートメントと FILE ステートメントで同じファイルを参照しています。SHAREBUFFERS オプションを指定することにより、外部ファイルを直接更新して、問題のある文字を削除できるようにしています。 <pre> data _null_; infile 'c:_today\mike.csv' recfm=n sharebuffers; file 'c:_today\mike.csv' recfm=n; input a \$char1.; retain open 0; /* toggle the open flag */ if a="" then open=not open; if a='0A'x and open then put ' '; run; </pre> このプログラムは、ファイルを 1 バイトずつ読み込み、必要ならばラインフィード文字を置き換えます。

長いインストリームデータレコードの読み込み

INFILE ステートメントと DATALINES ファイル指定を組み合わせることで、インストリームデータを処理できます。INPUT ステートメントは、DATALINES ステートメントの後に記述されたデータレコードを読み込みます。CARDIMAGE システムオプションを使用すると、空白が埋め込まれた 80 バイトのパンチカードイメージのようにデータ行を処理します。INFILE ステートメントではデフォルトで FLOWOVER オプションが使用されるため、INPUT ステートメントは、同ステートメントに含まれているすべての変数用の値を現在のレコード内で検出できない場合、次のレコードを読み込みます。データが正しく処理されることを保証するには、レコード長が 80 バイトを超える場合の入力では外部ファイルを使用します。

注: NOCARDIMAGE システムオプション("CARDIMAGE 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)は、データ行を 80 バイトのカードイメージ

ジとして扱わないよう指定します。データ行の終端は、引用符で囲まれた文字列の場合を除いて、常に最後のトークンの終端として扱われます。

行の終わりを超えて読み込む

デフォルトでは、INPUT ステートメントが現在の入力データレコードの終端を超えて読み込もうとする場合、同ステートメントは入力ポインターを次のレコードの列 1 に移動してから、残りの値を読み込みます。このデフォルトの動作は、FLOWOVER オプションにより指定されます。メッセージが SAS ログに書き込まれます。

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

行の終端に到達した場合の INPUT ステートメントの動作を変更する方法として、複数のオプションを使用できます。STOPOVER オプションを指定すると、この状態をエラーと判断し、データセットの作成を終了します。MISSOVER および TRUNCOVER オプションを指定すると、現在の INPUT ステートメントが満たされない場合でも、入力ポインターが次の行へと移動しなくなります。SCANOVER オプションを '@character-string' とともに使用すると、指定した *character-string* が検出されるまで入力レコードがスキャンされます。デフォルトの動作に戻すには FLOWOVER オプションを使用します。

TRUNCOVER と MISSOVER は類似したオプションです。MISSOVER オプションを指定すると、INPUT ステートメントは、同ステートメント内で指定されたフィールド長よりも値の長さが小さいためにフィールド全体を読み込めない場合、その値を欠損値に設定します。TRUNCOVER オプションを指定すると、対応する変数に読み込まれた文字が書き込まれます。

たとえば、可変長レコードを含む外部ファイルに次のレコードが含まれているとします。

```
----+----1----+----2
1
22
333
4444
55555
```

次の DATA ステップは、このデータを読み込み、SAS データセットを作成します。これらの入力レコードのうち、変数 TESTNUM の長さ(入力形式でフォーマットされたもの)と同じ長さのレコードは 1 つだけです。

```
data numbers;
  infile 'external-file';
  input testnum 5.;
run;
```

デフォルトでは入力レコードの読み込み時に FLOWOVER オプションが使用されるため、この DATA ステップは、5 つの入力レコードから 3 つのオブザベーションを作成します。

INFILE ステートメントで MISSOVER オプションを指定すると、この DATA ステップは 5 つのオブザベーションを作成します。短すぎるレコードから読み込まれた値はすべて欠損値に設定されます。短すぎるために現在の INPUT ステートメントを満

たすことができないレコード内にどんな値が含まれているかを確認したい場合は、INFILE ステートメントで TRUNCOVER オプションを指定します。

```
infile 'external-file' trunccover;
```

この場合、DATA ステップは、同じ入力レコードを読み込み、5 つのオブザベーションを作成します。SAS データセットを比較した表を次に示します。

表 2.2 INFILE ステートメントで異なるオプションを使用した場合の TESTNUM の値

OBS	FLOWOVER	MISCOVER	TRUNCOVER
1	22	.	1
2	4444	.	22
3	55555	.	333
4		.	4444
5		55555	55555

比較

- INFILE ステートメントでは、DATA ステップの INPUT ステートメントに対して入力ファイルが指定されます。FILE ステートメントでは、DATA ステップの PUT ステートメントに対して出力ファイルが指定されます。
- 通常、INFILE ステートメントは、外部ファイルからデータを読み込む場合に使用します。DATALINES ステートメントは、ジョブストリーム内の同ステートメントの後にデータが続くことを指定します。INFILE ステートメントと DATALINES ファイル指定を組み合わせると、INPUT ステートメントがインストリームデータを読み込む方法に影響を与える特定のデータ読み込みオプションを利用できるようになります。

例

例 1: 区切り文字の扱い方の変更

デフォルトでは、INPUT ステートメントはブランクを区切り文字として使用します。この例では、DATA ステップでは、カンマを区切り文字として使用します。

```
data num;  
  infile datalines dsd;  
  input x y z;  
  datalines;  
,2,3  
4,5,6
```

```
7,8,9
;
```

INFILE ステートメントの DATALINES 引数を指定すると、インストリームデータ行を読み込むための INFILE ステートメントオプションを使用できるようになります。DSD オプションを指定すると、カンマがデフォルトの区切り文字に設定されます。最初のデータ行の先頭にある値の前にカンマが記述されているため、1 番目のオブザベーション内の変数 X には欠損値が割り当てられ、変数 Y には値 2 が割り当てられます。

データで複数の区切り文字を使用している場合や、カンマ以外の単一区切り文字を使用している場合、DELIMITER=オプションを使用してその区切り文字値を指定します。次の例では、文字 **a** および **b** が区切り文字として使用されています。

```
data nums;
  infile datalines dsd delimiter='ab';
  input X Y Z;
  datalines;
1aa2ab3
4b5bab6
7a8b9
;

proc print; run;
```

実行後に作成された Nums データセットを PROC PRINT で出力した結果を次に示します。DSD オプションを指定すると、リスト入力は 2 つの連続する区切り文字を検出できるようになります。このため、1 番目と 2 番目のオブザベーション内の変数には欠損値が割り当てられます。DSD オプションを省略すると、文字 **a**、**b**、**aa**、**ab**、**ba**、または **bb** が区切り文字として機能するため、変数には欠損値が割り当てられません。

アウトプット 2.13 Nums データセット

Obs	X	Y	Z
1	1	.	2
2	4	5	.
3	7	8	9

文字列を区切り文字として使用するには、DLMSTR=オプションで区切り文字値を指定します。次の例では、文字列 **PRD** を区切り文字として使用します。この文字列にはすべて大文字が含まれていることに注意してください。DLMSOPT=オプションを使用することで、**PRD**、**Prd**、**PRd**、**PrD**、**pRd**、**pRD**、**prD**、**prd** をすべて有効な区切り文字にしています。

```
data test;
  infile datalines dsd dlmstr='PRD' dlmsopt='i';
  input X Y Z;
  datalines;
1PRD2PRd3
4PrD5Prd6
7pRd8pRD9
;
```

```
proc print data=test; run;
```

PROC PRINT による出力は、Test データセット内のすべてのオブザベーションを表示します。

アウトプット 2.14 Test データセット

Obs	X	Y	Z
1	1	2	3
2	4	5	6
3	7	8	9

次の DATA ステップでは、修飾リスト入力と DSD オプションを使用して、カンマで区切られたデータや、文字値の一部としてカンマを含んでいる可能性のあるデータを読み込みます。

```
data scores;
  infile datalines dsd;
  input Name : $9. Score
        Team : $25. Div $;
  datalines;
  Joseph,76,"Red Racers, Washington",AAA
  Mitchel,82,"Blue Bunnies, Richmond",AAA
  Sue Ellen,74,"Green Gazelles, Atlanta",AA
;
```

```
proc print; run;
```

実行後に作成された Scores データセットを PROC PRINT で出力した結果を次に示します。区切り文字(カンマ)は、Team の値の一部として格納されます。一方、引用符はそのような値としては格納されません。

アウトプット 2.15 Scores データセット

Obs	Name	Score	Team	Div
1	Joseph	76	Red Racers, Washington	AAA
2	Mitchel	82	Blue Bunnies, Richmond	AAA
3	Sue Ellen	74	Green Gazelles, Atlanta	AA

例 2: 欠損値と短いレコードをリスト入力で処理する

次の例では、リスト入力を使用してデータを読み込む場合に、欠損値を原因とする問題の発生を防ぐ方法を示します。この例の最初のデータ行には、5 つの値ではなく、3 つの値だけが含まれています。MISSOVER オプションを指定すると、最初のデータ行の 4 番目と 5 番目の値は欠損値に設定されます。

```
data weather;
  infile datalines missover;
```

```

input temp1-temp5;
datalines;
97.9 98.1 98.3
98.6 99.2 99.1 98.5 97.5
96.2 97.3 98.3 97.6 96.5
;

```

SAS は、最初のデータ行に含まれている 3 つの値を、TEMP1、TEMP2、TEMP3 の値として読み込みます。MISSOVER オプションを指定すると、最初のオブザベーション内の変数 TEMP4 および TEMP5 の値が欠損値に設定されます。これらの変数用の値が、現在の入力データレコード内には提供されていないためです。

MISSOVER オプションを省略するか、または FLOWOVER オプションを指定すると、SAS は入力ポインターを 2 番目のデータ行に移動し、変数 TEMP4 および TEMP5 の値をその行から読み込みます。DATA ステップの次の実行では、新しい行として、3 番目の行が読み込まれます。SAS ログには次のメッセージが表示されます。

NOTE: SAS went to a new line when INPUT statement
reached past the end of a line.

INFILE ステートメントに STOPOVER オプションを使用することもできます。STOPOVER オプションを使用すると、INPUT ステートメントで生データのレコードに十分な値が見つからない場合、DATA ステップの実行が停止します。

```
infile datalines stopover;
```

SAS では最初のデータレコードに TEMP4 値が見つからないため、_ERROR_ を 1 に設定し、データセットの構築を中止し、データライン 1 を印刷します。

例 3: 指定の文字列の直後に記述された可変長レコードをスキャンする

次の例では、TRUNCOVER オプションと SCANOVER オプションを組み合わせることで、電話帳から電話番号を取り出します。電話番号の前には、必ず "phone:" という文字列が記述されているものとします。電話番号は国際番号形式であるため、最大長は 32 文字になります。

```

filename phonebk host-specific-path;
data _null_;
file phonebk;
input line $80.;
put line;
datalines;
Jenny's Phone Book
Jim Johanson phone: 619-555-9340
Jim wants a scarf for the holidays.
Jane Jovalley phone: (213) 555-4820
Jane started growing cabbage in her garden.
Her dog's name is Juniper.
J.R. Hauptman phone: (49)12 34-56 78-90
J.R. is my brother.
;
run;

```

@'phone:'を使用すると、電話番号を含むファイルの行をスキャンし、電話番号が始まる位置にファイルポインターを移動できます。また、TRUNCOVER オプションと

SCANOVER オプションを組み合わせて使用することで、文字列'phone:'を含んでいない行をスキップし、電話番号のみを SAS ログに書き込みます。

```
data _null_;
  infile phonebk trunccover scanover;
  input @'phone:' phone $32.;
  put phone=;
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。

```
phone=619-555-9340
phone=(213) 555-4820
phone=(49)12 34-56 78-90
```

例 4: 可変長レコードを含むファイルの読み込み

次の例では、LENGTH=オプションと\$VARYING.入力形式を組み合わせて使用することで、可変長レコードを含むファイルを読み込んでいます。

```
data a;
  infile file-specification length=linelen lrecl=510 pad;
  input firstvar 1-10 @; /* assign LINELEN */
  varlen=linelen-10; /* Calculate VARLEN */
  input @11 secondvar $varying500. varlen;
run;
```

これらの動作は、直前の DATA ステップで行われます。

- INFILE ステートメントでは、変数 LINELEN を作成しますが、この変数には値を割り当てません。
- 最初の INPUT ステートメントを実行すると、SAS はレコード行の長さを判定し、変数 LINELEN に値を割り当てます。後置@が指定されているため、次の INPUT ステートメントの入力バッファにレコードが保持されます。
- 割り当てステートメントでは、2つの既知の長さ(FIRSTVAR の長さとレコード全体の長さ)を使用して、VARLEN の長さを決定します。
- 2番目の INPUT ステートメントでは、\$VARYING500.入力形式付きの VARLEN 値を使用して、変数 SECONDVAR の値を読み込みます。

詳細については、“[\\$VARYINGw. 入力形式](#)” ([SAS 出力形式と入力形式: リファレンス](#)) を参照してください。

例 5: 複数の外部ファイルからの読み込み

この DATA ステップは、DATA ステップの反復ごとに、2つの入力ファイルから読み込みます。SAS が1つのファイルから別のファイルへと処理を切り替える際に、各ファイルは開かれたままになります。入力ポインターは、現在の位置のまま移動しないため、次回 INPUT ステートメントを実行すると、ファイルの読み込みがその位置から開始されます。

```
data qtrtot(drop=jansale febsale marsale aprsale maysale junsale);
  infile quarter1; /* 1 */
  input name $ jansale febsale marsale; /* 2 */
  q1total = sum(jansale,febsale,marsale);

  infile quarter2; /* 3 */
```



```

input @7 aprsale maysale junsale;
q2total=sum(aprsale,maysale,junsale); /* 4 */
run;                               /* 5 */

```

- 1 最初のファイルの場所を特定するには、最初の INFILE ステートメントを指定します。
- 2 最初のファイルから値を読み込むには、最初の INPUT ステートメントを指定します。
- 3 2 つ目のファイルの場所を特定するには、2 つ目の INFILE ステートメントを指定します。
- 4 2 つ目のファイルから値を読み込むには、2 つ目の INPUT ステートメントを指定します。
- 5 SAS が最短の入力ファイルの最後のレコードに到達した時点で処理を停止します。

この DATA ステップでは、FILEVAR=オプションを使用して、DATA ステップの反復ごとにそれぞれ別のファイルを読み込みます。

```

data allsales;
  length fileloc myinfile $ 300;
  input fileloc $; /* 1 */
  infile datalines; /* 2 */
  infile dummy filevar=fileloc end=done /* 3 */
  do while(not done); /* 4 */
    input name $ jansale febsale marsale;
    output; /* 5 */
  end;
  put 'Finished reading ' myinfile;
  datalines;
external-file1
external-file2
external-file3
;

```

- 1 DATALINES ステートメントでインストリームデータを読み込むには、INPUT ステートメントを一時変数 fileloc とともに指定します。
- 2 INPUT ステートメントを指定して、DATA ステップの各反復において、変数 fileloc の値を読み込みます。
- 3 2 番目の INFILE ステートメントを 3 つのオプション(DUMMY、 FILEVAR=、および END=)で指定します。説明は次のとおりです。

DUMMY オプションは、INFILE ステートメントが慣習的なファイル参照を予想せず、代わりに FILEVAR=オプションを実行するようプレースホルダーの役割を果たします。filevar=fileloc ステートメントを使用して、SAS はファイル名を変数 fileloc に格納します。DATA ステップの各反復において、SAS は各外部ファイル名を読み込み、変数 fileloc に格納します。

END=オプションでは、一時変数 done を指定します。END=オプションは、最初に変数 done を 0 に設定します。これは、DATA ステートメント読み込む行がまだあることを示します。DATA ステップは、現在開いているファイルの最終行(ファイルの終わり)に到達すると、END=変数を 1 に設定し、これ以上読み込む行がないことを示します。この一時変数は、最後のファイルの最後のレコードを読み

込んだときに、DO ループが DATA ステップ処理を停止する条件として使われ
ます。

注: SAS では、最後の外部ファイルの最後のレコードを読み込んだときに、done
を 1 に設定します。

注: END=変数値(done)と FILEVAR=変数(fileloc)は出力データセットに書き込まれ
ません。

- 4 **DO WHILE ステートメント**を、done 変数を条件として指定します。変数 done が
1 に等しくない限り、DATA ステップは現在開いているデータセットの行を処理
し続けます。DATA ステップが DATALINES ステートメントに記載されている最
後のファイルの終わりを読み込むと、SAS は変数 done を 1 に設定し、DO ルー
プは現在のファイルからの行の読み込みを停止し、DATA ステップに制御を戻し
ます。その後、DATA ステップは再び繰り返し、fileloc の次の値(DATALINES ステ
ートメントの次のファイル名)を読み込みます。

- 5 **OUTPUT ステートメント**を指定します。

FILENAME=オプションは、現在の入力ファイルの名前を変数 MYINFILE に割り当て
ます。LENGTH ステートメントは、FILENAME=変数と FILEVAR=変数の長さが、こ
のファイル名を値として格納するために十分であることを保証します。PUT ステ
ートメントは、現在開かれている入力ファイルの物理名を SAS ログに出力します。

例 6: 外部ファイルの更新

この例では、SHAREBUFFERS オプションを指定した INFILE ステートメントと、
INPUT、FILE、PUT ステートメントを組み合わせることで、外部ファイルを
直接更新します。

```
data _null_;
  /* The INFILE and FILE statements */
  /* must specify the same file. */
  infile file-specification-1 sharebuffers;
  file file-specification-1;
  input state $ 1-2 phone $ 5-16;
  /* Replace area code for NC exchanges */
  if state='NC' and substr(phone,5,3)='333' then
    phone='910-'||substr(phone,5,8);
  put phone 5-16;
run;
```

例 7: コピーされたレコードの切り捨て

LENGTH=オプションは、PUT _INFILE_ ステートメントを使用して、入力ファイルを
別のファイルにコピーする場合に便利です。LENGTH=オプションを指定すると、コ
ピーされレコードを切り捨てることができます。たとえば、次のステートメントで
は、入力データレコードを出力ファイルに書き込む前に、各入力データレコードの
末尾から 20 列を切り捨てます。

```
data _null_;
  infile file-specification-1 length=a;
  input;
```

```

a=a-20;
file file-specification-2;
put _infile_;
run;

```

PUT _INFILE_ ステートメントでコピーした内容を切り捨てる場合、START=オプションを使用すると便利です。たとえば、各レコードの先頭から 10 列目までをコピーしたくない場合、次のステートメントを使用すると、入力バッファ内の各レコードの 11 列目から末尾までをコピーできます。

```

data _null_;
infile file-specification start=s;
input;
s=11;
file file-specification-2;
put _infile_;
run;

```

例 8: ポインター位置の出力

この DATA ステップでは、入力バッファ内の現在のポインターの位置を表す値を、変数 LINEPT および COLUMNPT に割り当てます。

```

data _null_;
infile datalines n=2 line=Linept col=Columnpt;
input name $ 1-15 #2 @3 id;
put linept= columnpt=;
datalines;
J. Brooks
40974
T. R. Ansen
4032
;

```

直前のステートメントは、DATA ステップを実行するたびに、これらの行を出力します。これは、PUT ステートメントの実行時には、入力ポインターが入力バッファ内の 2 番目の行に存在するためです。

```

Linept=2 Columnpt=9
Linept=2 Columnpt=8

```

例 9: 入力バッファ内のレコードを操作する

INFILE 変数には、常に、INPUT ステートメントを通じて読み込まれた最後のレコードが含まれています。次の例では、_INFILE_ 変数を使用して各種のタスクを実施する方法を示します。

- INPUT ステートメントを使用せずに解析したいレコード全体を読み込みます。
- SAS ログに書き込みたいレコード全体を読み込みます。
- この行が INPUT ステートメントにより解析される前に、入力レコードの内容を変更します。

次の例では、電話料金情報を含むファイルを作成します。数値データ(minutes および charge)は、山かっこ(<>)で囲んで指定します。

```

filename phonbill host-specific-filename;

```

```
data _null_;
  file phonbill;
  input line $80.;
  put line;
  datalines;
  City Number Minutes Charge
  Jackson 415-555-2384 <25> <2.45>
  Jefferson 813-555-2356 <15> <1.62>
  Joliet 913-555-3223 <65> <10.32>
  ;
run;
```

このコードでは、各レコードを読み込み、レコードを解析することにより、minute 値と charge 値を取り出します。

```
data _null_;
  infile phonbill firstobs=2;
  input;
  city = scan(_infile_, 1, ' ');
  char_min = scan(_infile_, 3, ' ');
  char_min = substr(char_min, 2, length(char_min)-2);
  minutes = input(char_min, BEST12.);
  put city= minutes=;
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。

```
city=Jackson minutes=25
city=Jefferson minutes=15
city=Joliet minutes=65
```

このコード内の INPUT ステートメントは、このファイルからレコードを読み込みます。_INFILE_ 自動変数を PUT ステートメントで使用するにより、レコードを SAS ログに書き込んでいます。

```
data _null_;
  infile phonbill;
  input;
  put _infile_;
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。

```
City Number Minutes Charge
Jackson 415-555-2384 <25> <2.45>
Jefferson 813-555-2356 <15> <1.62>
Joliet 913-555-3223 <65> <10.32>
```

このコードでは、最初の INPUT ステートメントで、読み込んだレコードを入力バッファ内に保持します。_INFILE_=オプションにより、数値データから山かっこ(<>)を取り除きます。2 番目の INPUT ステートメントで、バッファ内の値を解析します。

```
data _null_;
  length city number $16. minutes charge 8;
  infile phonbill firstobs=2;
  input @;
  _infile_ = compress(_infile_, '<>');
```

```
input city number minutes charge;
put city= number= minutes= charge=;
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。

```
city=Jackson number=415-555-2384 minutes=25 charge=2.45
city=Jefferson number=813-555-2356 minutes=15 charge=1.62
city=Joliet number=913-555-3223 minutes=65 charge=10.32
```

例 10: 複数のファイルの入力バッファへのアクセス

次の例では、_INFILE_ 自動変数と _INFILE_=オプションの両者を使用して複数のファイルを読み込んだ後、それぞれの入力バッファにアクセスしています。このコードを実行すると 4 つのファイルが作成されます。そのうちの 3 つはデータファイルであり、4 つ目のファイルにはすべてのデータファイルの名前が含まれます。2 番目の DATA ステップでは、file names ファイルを読み込み、各データファイルを開き、その内容を SAS ログに書き込みます。PUT ステートメントでは、ファイル名ファイルとデータファイル用に _INFILE_ 変数が必要となるため、_INFILE_ 変数の 1 つは *fname* を使用して参照されます。

```
data _null_;
  do i = 1 to 3;
    fname= 'external-data-file' || put(i,1.) || '.dat';
    file datfiles filevar=fname;
    do j = 1 to 5;
      put i j;
    end;
    file 'external-filenames-file';
    put fname;
  end;
run;
data _null_;
  infile 'external-filenames-file' _infile_=fname;
  input;
  infile datfiles filevar=fname end=eof;
  do while(^eof);
    input;
    put fname _infile_;
  end;
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。

NOTE: The infile 'external-filenames-file' is:

File Name=*external-filenames-file*,
RECFM=V, LRECL=256

NOTE: The infile DATFILES is:

File Name=*external-data-file1.dat*,
RECFM=V, LRECL=256

external-data-file1.dat 1 1

external-data-file1.dat 1 2

external-data-file1.dat 1 3

external-data-file1.dat 1 4

external-data-file1.dat 1 5

NOTE: The infile DATFILES is

File Name=*external-data-file2.dat*,
RECFM=V, LRECL=256

external-data-file2.dat 2 1

external-data-file2.dat 2 2

external-data-file2.dat 2 3

external-data-file2.dat 2 4

external-data-file2.dat 2 5

NOTE: The infile DATFILES is

File Name=*external-data-file3.dat*,
RECFM=V, LRECL=256

external-data-file3.dat 3 1

external-data-file3.dat 3 2

external-data-file3.dat 3 3

external-data-file3.dat 3 4

external-data-file3.dat 3 5

例 11: 外部ファイル読み込み時のエンコードの指定

この例では、外部ファイルから SAS データセットを作成します。外部ファイルのエンコーディングは UTF-8 であり、現在の SAS セッションのエンコーディングは Wlatin1 です。デフォルトでは、SAS は、外部ファイルのエンコーディングがセッションエンコーディングと同じであると見なします。このため、文字データは新しい SAS データセットに正しく書き込まれません。

外部ファイルの読み込み時に使用するエンコーディングを指定するには、ENCODING=オプションを指定します。外部ファイルのエンコーディングを UTF-8 に指定すると、新しい SAS データセットへの書き込み時に、外部ファイルが UTF-8 から現在のセッションエンコーディングにトランスコードされます。これで、新しいデータセットにデータが Wlatin1 のエンコーディングで正しく書き込まれるようになります。

```
libname myfiles 'SAS-library';
filename extfile 'external-file';
data myfiles.unicode;
  infile extfile encoding='utf-8';
  input Make $ Model $ Year;
run;
```

例 12: ディレクトリのすべてのファイルメンバーの読み込み

この例では、空白に設定された MEMVAR=変数を使用して、'c:\mydir\tmp'ディレクトリからすべてのファイルを読み込みます。

```
data _null_;
  length memname $ 1024;
  memname = '';
  infile 'c:\mydir\tmp' memvar=memname end=done;
```

```

put memname=;
do while (^done);
  input x;
  put _all_;
end;
run;

```

例 13: ファイルメンバーのリストの読み込み

この例では、c:\mydir\tmp'ディレクトリから、MEMVAR=オプションに割り当てられたファイルのリストを読み込みます。

```

data _null_;
  length mname $ 32;
  do mname = 'erase.dat', 'erase2.dat';
    infile 'c:\mydir\tmp' memvar=mname end=done;
    do while (^done);
      input x;
      put _all_;
    end;
  end;
  stop;
run;

```

例 14: URL リクエストからのリターンコードの読み込み

この例では、存在しない Web リソースにアクセスしようとしています。その結果、STATUS=変数には、ページが見つからないコード(404)が設定されています。

```

filename in url "http://www.sas.com/missing_page.html" lrecl=1000;
data _NULL_;
  infile in status=ioerr;
  input;
  if (ioerr = 404) then do;
    put 'File not found';
    stop;
  end;
run;

```

次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。

```
File not found
```

関連項目

- [“名前の長さの規則” \(SAS プログラマガイド: エッセンシャル\)](#)

ステートメント:

- [“FILENAME ステートメント” \(128 ページ\)](#)

- “FILENAME Statement: ACTIVEMQ Access Method” (*Application Messaging with SAS*)
- “FILENAME ステートメント: Azure Data Lake Storage アクセス方式” (SAS グローバルステートメント: リファレンス)
- “FILENAME Statement: JMS Access Method” (*Application Messaging with SAS*)
- “FILENAME ステートメント: S3 アクセスメソッド” (SAS グローバルステートメント: リファレンス)
- “INPUT ステートメント” (186 ページ)
- SAS Intelligence Platform: Application Server Administration Guide の “LOCKDOWN ステートメント”
- “PUT ステートメント” (289 ページ)

システムオプション:

- SAS Intelligence Platform: Application Server Administration Guide の “LOCKDOWN システムオプション”

INFORMAT ステートメント

変数に入力形式を関連付けます。

該当要素:	DATA ステップまたは PROC ステップ
カテゴリ:	CAS 情報
種類:	宣言
制限事項:	このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“ DATA Step Acceleration ” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。

構文

INFORMAT *variable-1* <...*variable-n*> <*informat*>;

INFORMAT <*variable-1*> <... *variable-n*> <DEFAULT=*default-informat*>;

INFORMAT *variable-1* <...*variable-n*> *informat* <DEFAULT=*default-informat*>;

引数

variable

1 つまたは複数の変数に関連付ける入力形式を指定します。*informat* を指定する場合、または他の引数を指定しない場合は、少なくとも *variable* を 1 つ指定する必要があります。入力形式として DEFAULT=を使用する場合、変数の指定は任意です。

ヒント 変数から入力形式の関連付けを取り消すには、この変数を INFORMAT ステートメントに入力形式を指定せずに使用します。INFORMAT ステートメントは SET ステートメントの後に指定します。[“例 3: 入力形式の取り消し” \(186 ページ\)](#)を参照してください。

informat

INFORMAT ステートメントにリストされている変数の値を読み込むときに使用する入力形式を指定します。

ヒント 入力形式が INFORMAT ステートメントを使用して変数に関連付けられ、INPUT ステートメントにて同じ変数に同じ入力形式が関連付けられていない場合、この入力形式は INPUT ステートメントにコロン(:)修飾子付きで指定された入力形式と同じように動作します。変数は、入力形式を使いリスト入力経由で読み込まれます。たとえば、コロン修飾子を入力形式とともに使用すると、8 バイトよりも長い文字の値や非標準の値を含む数値を読み込むことができます。詳細については、[“INPUT ステートメント: リスト” \(215 ページ\)](#)を参照してください。

参照 [SAS 出力形式と入力形式: リファレンス](#)

項目:

例 [“例 2: 数値および文字の入力形式を指定する” \(185 ページ\)](#)

DEFAULT= default-informat

INFORMAT ステートメントにリストされている変数の値を読み込むときに使用する一時的なデフォルトの入力形式を指定します。*variable* を指定しない場合、DATA ステップにあるすべての変数の読み込みに使用する一時的なデフォルトの入力形式として、DEFAULT=に指定した入力形式が適用されます。数値変数には数値の入力形式が適用され、文字変数には文字の入力形式が適用されます。このデフォルト入力形式は、現在の DATA ステップにのみ適用されます。

DEFAULT=に指定した入力形式の適用対象

- INFORMAT または ATTRIB ステートメントに指定されていない変数
- SAS データセット内で入力形式が恒久的に関連付けられていない変数
- 現在の DATA ステップにおいて、明示的に指定した入力形式を使用して読み込まれない変数

デフォルト DEFAULT=の指定を省略すると、デフォルトの数値の入力形式として *w.d* が使用され、デフォルトの文字の入力形式として *\$w.*が使用されます。

制限事項 この引数は DATA ステップのみで使用します。

ヒント DEFAULT=は、INFORMAT ステートメントのどの位置に指定してもかまいません。数値に対するデフォルト値、文字に対するデフォルト値、またはその両方を指定できます。

例 [“例 1: デフォルトの入力形式を指定する” \(185 ページ\)](#)

詳細

基本情報

DATA ステップ内の INFORMAT ステートメントは、変数に入力形式を恒久的に割り当てます。標準の SAS 入力形式または PROC FORMAT で定義したユーザー定義の入力形式を指定できます。1 つの INFORMAT ステートメントにて、同じ入力形式を複数の変数に関連付けることや、異なる入力形式を複数の変数に関連付けることができます。同じ変数が複数の INFORMAT ステートメントに指定されている場合、最後に割り当てられた入力形式が使用されます。

注意

INFORMAT ステートメントは事前に定義されていない文字変数の長さを定義するため、SET ステートメントの前に使用すると、DATA ステップ内の文字変数の値を切り捨てることができます。

INFORMAT ステートメントで入力形式を割り当てた場合の変数の処理

INFORMAT ステートメントを使用して変数に入力形式を割り当てると、修飾リスト入力で使用される入力形式と同じように動作します。変数はリスト入力の走査機能を使用して読み込まれますが、入力形式が適用されます。修飾リスト入力では、SAS は次の処理を行います。

- 入力形式の *w* の値を使用して、外部ファイルの列位置や入力フィールド幅を設定しない
- 入力形式の *w* の値を使用して、事前に定義されていない文字変数の長さを設定する
- 数値の入力形式の *w* の値は無視する
- 入力形式の *d* の値を、数値入力形式の場合と同じように使用する
- 入力データに埋め込まれているブランクは、INFILE ステートメントの DLM=オプションまたは DLMSTR=オプションを変更しない限り、区切り文字として処理する

フォーマット入力や列入力などの別の入力スタイルを使用するように INPUT ステートメントをコード化した場合、INFORMAT オプションの使用時にはその入力スタイルは使用されません。

比較

- ATTRIB ステートメントと INFORMAT ステートメントは、どちらも変数に入力形式を関連付けることができます。また、どちらのステートメントを使用しても変数に関連付けられた入力形式を変更することができます。PROC DATASETS で INFORMAT ステートメントを使用して、変数に関連付けられた入力形式の変更や削除を行うこともできます。
- ステートメントを使用すると、指定した変数が含まれる SAS データセットのディクショナリ情報が変更されます。一部の PROC ステップでも INFORMAT ステ

ートメントを使用できますが、ルールは異なります。詳細については、“[FORMAT プロシジャ](#)” (*Base SAS プロシジャガイド*)を参照してください。

例

例 1: デフォルトの入力形式を指定する

次の例では、INFORMAT ステートメントを使用してデフォルトの数値入力形式を関連付けます。

```
data tstinfmt;
  informat default=3.1;
  input x;
  put x;
  datalines;
111
222
333
;
```

PUT ステートメントの実行結果は次のようになります。

```
11.1
22.2
33.3
```

例 2: 数値および文字の入力形式を指定する

この例では、SAS 変数に文字の入力形式と数値の入力形式を関連付けます。文字変数は列位置 15 まですべて占めるものではありませんが、INPUT ステートメントは修飾リスト入力を使用してデータレコードを正しく読み込みます。

```
data name;
  informat FirstName LastName $15. n1 6.2 n2 7.3;
  input firstname lastname n1 n2;
  datalines;
Alexander Robinson 35 11
;
proc contents data=name;
run;
proc print data=name;
run;
```

出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

アウトプット 2.16 SAS 変数への数値および文字の入力形式の関連付け

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Informat
1	FirstName	Char	15	\$15.
2	LastName	Char	15	\$15.
3	n1	Num	8	6.2
4	n2	Num	8	7.3

アウトプット 2.17 PROC PRINT で生成したレポート

Obs	FirstName	LastName	n1	n2
1	Alexander	Robinson	0.35	0.011

例 3: 入力形式の取り消し

この例では既存の入力形式の関連付けを取り消します。INFORMAT ステートメントと SET ステートメントを指定する順序が重要です。

```
data rtest;
  set rtest;
  informat x;
run;
```

関連項目

ステートメント:

- [“ATTRIB ステートメント” \(52 ページ\)](#)
- [“INPUT ステートメント” \(186 ページ\)](#)
- [“INPUT ステートメント: リスト” \(215 ページ\)](#)

INPUT ステートメント

入力データレコードでの値の位置を指定し、読み込んだ値を対応する SAS 変数に割り当てます。

該当要素: DATA ステップ
カテゴリ: ファイル操作

種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (SAS In-Database Products: User's Guide)を参照してください。

構文

INPUT <*specification(s)*> <@ | @@>;

引数なし

引数を指定しない INPUT ステートメントは、*null INPUT* ステートメントと呼ばれます。null INPUT ステートメントは、次の場合に使用します。

- SAS 変数を作成せずに、入力データレコードを入力バッファに読み込む場合
- 後置@または後置@@で保持されている入力データレコードを解放する場合

この例については、“[例 2: null INPUT ステートメントを使用する](#)” (201 ページ)を参照してください。

注: INPUT ステートメントを使用して、引数なしでバイナリファイルを読み込むことはできません。

引数

specification(s)

次を指定できます。

variable

読み込む値を割り当てる変数名を 1 つ指定します。

(variable-list)

読み込む値を割り当てる変数名のリストを指定します。

要件 (variable-list)の後ろに(informat-list)を指定します。

参照項目: “[変数と入力形式をグループ化する方法](#)” (213 ページ)

“[変数リスト](#)” (SAS プログラマガイド: エッセンシャル)

例 “[例: 変数リストの作成](#)” (SAS プログラマガイド: エッセンシャル)

\$

変数の値を数値ではなく文字値として格納するように指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例 “[例 1: INPUT ステートメントで複数のスタイルを使用する](#)” (201 ページ)

pointer-control

入力バッファ内の指定した行または列に入力ポインターを移動させます。

参照項目: [“列ポインターコントロール” \(189 ページ\)](#)および[“行ポインターコントロール” \(191 ページ\)](#)

column-specifications

読み込む値が含まれる入力レコードの列を指定します。

ヒント 入力形式は無視されます。この方法では、標準の文字データと数値データのみ正しく読み込めます。

参照項目: [“列入力” \(193 ページ\)](#)

例 [“例 1: INPUT ステートメントで複数のスタイルを使用する” \(201 ページ\)](#)

format-modifier

修飾リスト入力を使用できるようにします。また、入力値にエラーが発生したときに SAS ログに出力する情報の量を制御します。

ヒント 修飾リスト入力を使用すると、単純リスト入力では読み込めないデータを読み込めます。

参照項目: [“リスト入力の使用が求められる場合” \(218 ページ\)](#)および[“エラーレポートの入力形式修飾子” \(192 ページ\)](#)

例 [“例 6: 文字変数でポインターの位置を指定する” \(204 ページ\)](#)

informat.

変数の値の読み込みに使用する入力形式を指定します。

ヒント 修飾リスト入力を使用すると、入力形式を指定したデータを読み込めます。修飾リスト入力は、データに入力形式が必要ですが、値の列位置が揃っていないためフォーマット入力ではデータを読み込めない場合に使用すると便利です。

参照項目: [“フォーマット入力” \(194 ページ\)](#)および[“リスト入力” \(193 ページ\)](#)

例 [“例 2: 入力形式リストを使用する” \(214 ページ\)](#)

(informat-list)

入力形式のリストを指定します。このリストは、前の位置に指定した変数リストの値の読み込みに使用します。

制限事項 *(informat-list)*は*(variable-list)*の後ろに指定する必要があります。

参照項目: [“変数と入力形式をグループ化する方法” \(213 ページ\)](#)

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、後置@と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

例 [“例 3: 入力バッファ内のレコードを保持する” \(202 ページ\)](#)

@@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、後置@@と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@@は、それぞれの入力行に複数のオブザベーションの値が含まれる場合や、DATA ステップの次の繰り返しでレコードをもう一度読み込む必要がある場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

例 [“例 4: DATA ステップの反復間を通してレコードを保持する” \(203 ページ\)](#)

列ポインターコントロール

@n

ポインターを列 n に移動させます。

範囲 正の整数

ヒント n が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。 n が 0 または負の場合、ポインターは列 1 に移動します。

例 @15 と指定すると、ポインターは列 15 に移動します。
input @15 name \$10.;

例 [“例 7: ポインターを後方\(左\)に移動する” \(205 ページ\)](#)

@numeric-variable

指定した *numeric-variable* の値が示す列にポインターを移動させます。

範囲 正の整数

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が 0 または負の整数の場合、ポインターを列 1 に移動させます。

例 変数 A の値に従って、ポインターは列 15 に移動します。
 a=15;
 input @a name \$10.;

例 [“例 5: 数値変数でポインターの位置を指定する” \(203 ページ\)](#)

@(*expression*)

指定した *expression* の値が示す列にポインターを移動させます。

制限 *expression* の実行結果は正の整数でなければなりません。
 事項

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。値が 0 または負の整数の場合、ポインターを列 1 に移動させます。

例 式の実行結果に従って、ポインターを列 15 に移動します。
 b=5;
 input @(b*3) name \$10.;

@'*character-string*'

指定した *character-string* を入力レコードから検索し、検出された文字列の終わ
 りから 1 番目の列にポインターを移動させます。

@*character-variable*

指定した *character-variable* の値が示す文字列を入力レコードから検索し、検出
 された文字列の終わりに 1 番目の列にポインターを移動させます。

例 このステートメントでは、文字変数 WEEKDAY を読み込みます。2 番目に指
 定した @1 に従って、ポインターを入力行の先頭に移動させます。変数
 SALES の値は、変数 WEEKDAY の値の後方の非ブランク列から読み込まれま
 す。

```
input @1 day 1. @5 weekday $10.  
      @1 @weekday sales 8.2;
```

例 [“例 6: 文字変数でポインターの位置を指定する” \(204 ページ\)](#)

@(*character-expression*)

指定した *character-expression* の値が示す文字列を入力レコードから検索し、検
 出された文字列の終わりに 1 番目の列にポインターを移動させます。

例 [“例 6: 文字変数でポインターの位置を指定する” \(204 ページ\)](#)

+*n*

n に指定した列数だけポインターを移動させます。

範囲 正の整数または 0

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用
 します。この値が入力バッファの長さを超える場合、ポインターは次の
 レコードの列 1 に移動します。

例 次のステートメントでは、ポインターを列 23 に移動させます。LENGTH
 変数の値として列 23 から 26 ままでを読み込みます。次に、ポインターを 5

列右に移動させ、WIDTH 変数の値として列 32 から 35 までを読み込みます。

```
input @23 length 4. +5 width 4.;
```

例 “例 7: ポインターを後方(左)に移動する” (205 ページ)

+numeric-variable

指定した *numeric-variable* の値が示す列数だけポインターを移動させます。

範囲 正の整数、負の整数、または 0

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が負の値の場合、ポインターを後方(左)へ移動させます。ただし、現在の列位置が 1 未満になる場合、ポインターは列 1 に移動します。値が 0 の場合、ポインターは移動しません。この値が入力バッファの長さを超える場合、ポインターは次のレコードの列 1 に移動します。

例 “例 7: ポインターを後方(左)に移動する” (205 ページ)

+(expression)

指定した *expression* の値が示す列数だけポインターを移動させます。

範囲 *expression* の実行結果が、正の整数、負の整数または 0 になる式を指定します。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*expression* の実行結果が負の値の場合、ポインターを左に戻します。ただし、現在の列位置が 1 未満になる場合、ポインターは列 1 に移動します。値が 0 の場合、ポインターは移動しません。この値が入力バッファの長さを超える場合、ポインターは次のレコードの列 1 に移動します。

行ポインターコントロール

#n

ポインターをレコード *n* に移動させます。

範囲 正の整数

操作 INFILE ステートメントの N=オプションの値によって、INPUT ステートメントで読み込むレコード数や、DATA ステップの繰り返しを終了した後の入力ポインターの位置は異なります。[N= \(154 ページ\)](#)オプションの説明を参照してください。

例 #2 と指定されているので、ポインターは 2 番目のレコードに移動してから、変数 ID の値として列 3 と 4 を読み込みます。

```
input name $10. #2 id 3-4;
```

#numeric-variable

指定した *numeric-variable* の値が示すレコードにポインターを移動させます。

範囲 正の整数

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

#(*expression*)

指定した *expression* の値が示すレコードにポインターを移動させます。

範囲 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

/

ポインターを次の入力レコードの列 1 に移動させます。

例 最初の入力レコードから変数 NAME と変数 AGE の値を読み込みます。次に、ポインターを 2 番目のレコードに移動し、変数 ID の値として列 3 と 4 を読み込みます。

```
input name age / id 3-4;
```

エラーレポートの入力形式修飾子

?

無効なデータ値が検出されたときに、無効なデータに関するメッセージを出力しないようにします。

参照項目: [“無効なデータの処理方法” \(199 ページ\)](#)

??

無効なデータ値が検出されたときに、メッセージと該当する入力行を出力しないようにします。この無効なオブザベーションに対しては、自動変数 `_ERROR_` の値は 1 に設定されません。

参照項目: [“無効なデータの処理方法” \(199 ページ\)](#)

詳細

INPUT を使用する必要がある場合

外部ファイルやインストリームデータから生データを読み込むには、INPUT ステートメントを使用します。データが外部ファイルに格納されている場合、INFILE ステートメントにそのファイルを指定できます。INFILE ステートメントは、データレコードを読み込む INPUT ステートメントよりも前に実行する必要があります。読み込むデータがインストリームデータの場合、ジョブストリーム内の DATALINES ステートメントに続けてデータ行を指定する必要があります。読み込むデータにセミコロンが含まれる場合、DATALINES4 ステートメントに続けてデータ行を指定します。生データを読み込む 1 つの DATA ステップには、複数の INPUT ステートメントを使用できます。

INPUT ステートメントの前に INFILE ステートメントの file-specification に DATALINES を指定すると、INFILE ステートメントを使用してインストリームデータを読み込むこともできます。INFILE ステートメントに DATALINES を指定すると、

INFILE ステートメントのほとんどのオプションをインストリームデータの読み込みに使用できます。

SAS データセットに格納されているデータを読み込むには、SET ステートメントを使用します。SAS 以外のソフトウェアで作成されたデータベースまたは PC ファイル形式のデータを読み込むには、LIBNAME ステートメントでデータにアクセスした後に、SET ステートメントを使用します。詳細については、[SAS/ACCESS for Relational Databases: リファレンス](#)および [SAS/ACCESS Interface to PC Files: リファレンス](#)を参照してください。

z/OS 固有: z/OS で生成され、PROC PRINTTO でキャプチャされたログファイルには、列 1 に ASA コントロール文字が含まれています。INPUT ステートメントに列入力または列ポインターコントロールを使用して、z/OS で生成されたログファイルを読み込む場合、この文字を考慮する必要があります。

入力スタイル

入力スタイルの概要

INPUT ステートメントでレコードの値を指定する方法は 4 つあります。

- 列
- リスト(単純および修飾)
- フォーマット
- 名前付き

各変数の値は、次のいずれかの入力スタイルを使用して読み込まれます。INPUT ステートメントには、入力レコードでのデータ値の配置に応じて、使用可能な入力スタイルのいずれかまたはすべてを指定することができます。ただし、名前付き入力が入力スタイルに使用されると、他の入力スタイルを使用することはできません。

列入力

列入力を使用する場合、INPUT ステートメントで、値の位置を示す列番号を変数名の後ろに指定します。番号は、入力データレコード内で変数の値が存在する位置を示しています。

```
input name $ 1-8 age 11-12;
```

この INPUT ステートメントを実行すると、次のデータレコードを読み込むことができます。

```
----+----1----+----2----+
Peterson 21
Morgan 17
```

NAME は文字変数なので、変数名と列番号の間に\$を指定しています。詳細については、[“INPUT ステートメント: 列” \(205 ページ\)](#)を参照してください。

リスト入力

リスト入力では、変数名を INPUT ステートメント内に単にリストします。各文字変数名の後ろには\$を指定します。

```
input name $ age;
```

この INPUT ステートメントを実行すると、空白で区切られているデータ値、または列位置が揃っているデータ値を読み込むことができます。ただし、それぞれの値の間には 1 つ以上の空白が必要です。

```
----+----1----+----2----+
Peterson 21
Morgan 17
```

詳細については、“[INPUT ステートメント: リスト](#)” (215 ページ)を参照してください。

フォーマット入力

フォーマット入力では、INPUT ステートメントで変数名の後に入力形式を指定します。入力形式では、データの種類や入力値のフィールド長を指示します。入力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を読み込むこともできます。

```
input name $char8. +2 income comma6.;
```

この INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```
----+----1----+----2----+
Peterson 21,000
Morgan 17,132
```

ポインターコントロールを +2 と指定し、入力ポインターを変数 INCOME の値が含まれるフィールドに移動させています。詳細については、“[INPUT ステートメント: フォーマット](#)” (210 ページ)を参照してください。

名前付き入力

入力名前付き 名前付き入力名前付き入力では、変数名の後に等号(=)を指定します。入力レコードから変数名と等号が検索されます。

```
input name= $ age=;
```

次の INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```
----+----1----+----2----+
name=Peterson age=21
name=Morgan age=17
```

詳細については、“[INPUT ステートメント: 名前付き](#)” (223 ページ)を参照してください。

1 つの INPUT ステートメントに複数のスタイルを指定する

1 つの INPUT ステートメントに異なる入力スタイルのいずれか、またはすべてを指定できます。

```
input idno name $18. team $ 25-30 startwght endwght;
```

次の INPUT ステートメントを実行すると、次のデータレコードが正しく読み込まれます。

```

----+----1----+----2----+----3----+----
023 David Shaw    red  189 165
049 Amelia Serrano yellow 189 165

```

変数 IDNO、STARTWGHT、ENDWGHT の値はリスト入力で、変数 NAME の値はフォーマット入力で読み込まれます。また、変数 TEAM の値は列入力で読み込まれます。

注: INPUT ステートメントに名前付き入力を使用されると、入力スタイルを変更することはできません。

ポインターコントロール

ポインターの概要

入力データレコードから入力バッファに値を読み込む場合、ポインターを使用して読み込み位置を制御します。INPUT ステートメントには、ポインターの動きを制御する 3 つの方法があります。

列ポインターコントロール

データレコードのデータ値を読み込むときに、ポインターの列位置をリセットします。

行ポインターコントロール

データレコードのデータ値を読み込むときに、ポインターの行位置をリセットします。

ラインホールド指定子

読み込んだ入力レコードが入力バッファに保持されるので、他の INPUT ステートメントでも保持した値を使用できます。デフォルトでは、INPUT ステートメントを実行すると、前回読み込んだレコードを解放してから、次のレコードを読み込みます。

列ポインターコントロールや行ポインターコントロールを使用すると、ポインターを移動させる行または列の絶対位置を指定したり、現在のポインターの位置の相対位置として移動先の列または行を指定できます。INPUT ステートメントで指定できるすべてのポインターコントロールをこのテーブルに示します。

表 2.3 INPUT ステートメントで指定できるポインターコントロール

ポインターコントロール	相対指定	絶対指定
列ポインターコントロール	$+n$	$@n$
	$+numeric-variable$	$@numeric-variable$
	$+(expression)$	$@(expression)$
		$@'character-string'$
		$@character-variable$

ポインターコントロール	相対指定	絶対指定
		@(<i>character-expression</i>)
行ポインターコントロール	/	# <i>n</i>
		# <i>numeric-variable</i>
		#(<i>expression</i>)
ラインホールド指定子	@	(なし)
	@@	(なし)

注: ポインターコントロールは、常に適用する変数の前に指定するようにしてください。

INFILE ステートメントに COLUMN=オプションや LINE=オプションを指定して、ポインターの現在の列位置や行の位置を確認することができます。

列ポインターコントロールと行ポインターコントロールの使用

列ポインターコントロールでは、入力値の読み込みを開始する列を指定します。

次の入力レコードに移動したり、オブザベーションごとの入力レコード数を定義するには、INPUT ステートメントで行ポインターコントロールを使用します。行ポインターコントロールには、読み込み対象とする入力レコードを指定します。複数のデータレコードを入力バッファに読み込むには、INFILE ステートメントの N=オプションに読み込むレコード数を指定します。N=オプションの指定を省略する場合は、特別な注意が必要になります。詳細については、[“1 つのオブザベーションにつき複数のレコードを読み込む” \(198 ページ\)](#)を参照してください。

ラインホールド指定子の使用

ラインホールド指定子を使用して、次の場合にポインターを現在の出力行に保持します。

- 複数の INPUT ステートメントで同じデータレコードを読み込む場合(後置@)
- 1 つの入力行に複数のオブザベーションの値が含まれる場合(後置@@)
- DATA ステップの次の反復でレコードをもう一度読み込む必要がある場合(後置@@)

後置@を使用すると、次の INPUT ステートメントで同じレコードを読み込むことができます。後置@@を使用すると、DATA ステップの反復間を通して、読み込んだレコードを保持して次の INPUT ステートメントで使用することができます。

通常、DATA ステップで INPUT ステートメントを実行するたびに、新しいデータレコードが入力バッファに読み込まれます。後置@を使用する場合は、次のようになります。

- ポインターの位置は変化しません。
- 入力バッファに新しいレコードは読み込まれません。
- DATA ステップの同じ反復内で次の INPUT ステートメントを実行すると、新しいレコードではなく同じレコードが読み込まれます。

後置@で保持したレコードは、次の場合に解放されます。

- null INPUT ステートメントを実行した場合:
input;
- 後置@を指定せずに INPUT ステートメントを実行した場合
- DATA ステップの次の反復を開始した場合

通常、後置@@を使用すると、DATA ステップの次の反復でも INPUT ステートメントは同じレコードを読み込みます。後置@@で保持したレコードは、次の場合に解放されます。

- ポインターが入力レコードの行端を超えた場合
- null INPUT ステートメントを実行した場合
input;
- DATA ステップの次の反復が開始され、この DATA ステップで後置@を指定した INPUT ステートメントが実行された場合
input @;

値を読み込んだ後のポインターの位置

1 つの INPUT ステートメントに複数の入力スタイルを組み合わせて使用する場合、値を読み込んだ後の入力ポインターの位置を理解しておくことが重要になります。列入力とフォーマット入力を使用すると、ポインターは INPUT ステートメントに指定された列を読み込んだあと、次の列で停止します。リスト入力を使用すると、ポインターはデータレコードを走査してデータ値を探してから、値の終わりを示すブランクを読み込みます。そのため、リスト入力で値を読み込んだ後、ポインターは値の終わりから 2 番目の列で停止します。

たとえば、リスト入力、列入力、フォーマット入力を使用して次のデータレコードを読み込むことができます。

```
----+----1----+----2----+----3
REGION1  49670
REGION2  97540
REGION3  86342
```

この INPUT ステートメントでは、リスト入力を使用してデータレコードを読み込みます。

```
input region $ jansales;
```

REGION の値を読み込んだ後、ポインターは列 9 で停止します。

```
----+----1----+----2----+----3
REGION1  49670
      ↑
```

次の INPUT ステートメントでは、列入力とフォーマット入力を使用してデータレコードを読み込みます。

■ 列入力

```
input region $ 1-7 jansales 12-16;
```

■ フォーマット入力

```
input region $7. +4 jansales 5.;
input region $7. @12 jansales 5.;
```

変数 REGION の値を読み込むため、この INPUT ステートメントでは、7 列読み込んでから列 8 で停止するようにポインターに指示します。

```
----+----1----+----2----+----3
REGION1  49670
      ↑
```

1 つのオブザベーションにつき複数のレコードを読み込む

#ポインターコントロールを使用する

INPUT ステートメントの#ポインターコントロールの後ろに指定した最大値によって、入力バッファに読み込む入力データレコード数が決まります。INFILE ステートメントに N=オプションを指定すると、このレコード数を変更することができます。たとえば、次のステートメントでは、#ポインターコントロールに指定した最大値は 3 になります。

```
input @31 age 3. #3 id 3-4 #2 @6 name $20.;
```

関連する INFILE ステートメントに N=オプションを指定していない場合、DATA ステップを実行するたびに、INPUT ステートメントは 3 つの入力レコードを読み込みます。

各オブザベーションに複数の入力レコードが存在しても、最後のレコードの値は読み込まない場合、INPUT ステートメントに#ポインターコントロールを使用するか、INFILE ステートメントに N=オプションを使用して最後に読み込む入力レコードを指定する必要があります。たとえば、各オブザベーションに 4 つのレコードが存在しても、最初の 2 つの入力レコードの値しか読み込まない場合、次の INPUT ステートメントを使用します。

```
input name $ 1-10 #2 age 13-14 #4;
```

/ポインターコントロールを指定してポインターを次のレコードに移動した場合、INPUT ステートメントに#ポインターコントロールを指定するか、INFILE ステートメントに N=オプションを指定して、入力バッファに読み込むレコード数を設定します。ポインターを前のレコードに戻すには、#ポインターコントロールを使用します。たとえば、次のステートメントでは、INFILE ステートメントに N=オプションを指定していない場合、#2 ポインターコントロールを指定してレコードを 2 つ読み込むように指示する必要があります。

```
input a / b #1 @52 c #2;
```

この INPUT ステートメントでは、変数 A に最初のレコードの値を割り当てます。ポインターを次の入力レコードに移動してから、変数 B に値を割り当てます。次に、ポインターを 2 番目のレコードから最初のレコードの列 1 に復帰した後に、列 52 に移動して変数 C に値を割り当てます。ポインターコントロールの#2 は、オブザベーションの入力レコードが 2 行にわたることを示すため、C の値を読み込むためにポインターが最初のレコードに復帰できます。

オブザベーションごとに入力レコード数が異なる場合、INFILE ステートメントの N=オプションにオブザベーションのレコード数の最大値を指定します。詳細については、[N=オプション \(154 ページ\)](#)を参照してください。

行の終わりを超えて読み込む

ポインターコントロールの@または+に指定した値が、現在のレコードの終端または終端を超えた位置に移動し、その列から次の値を読み込むように指示している場合、ポインターは、次のレコードの列 1 に移動してから値を読み込みます。次のメッセージが SAS ログに書き込まれます。

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.

INFILE ステートメントでデフォルトの動作(FLOWOVER オプション)を変更することができます。

INFILE ステートメントに STOPOVER オプションを指定すると、この状態をエラーと判断し、データセットの作成を終了します。

INFILE ステートメントに MISCOVER オプションを指定すると、ポインターがレコードの終端に到達した場合、INPUT ステートメントでまだ値が読み込まれていない変数には欠損値を割り当てます。

INFILE ステートメントに TRUNCOVER オプションを指定すると、INPUT ステートメントで読み込んだ最後の変数に可変長データが含まれている場合、列入力またはフォーマット入力を使用します。

レコードの前にポインターを移動する

列ポインターコントロールをレコードの開始位置よりも前に移動するように指示した場合、ポインターは列 1 に移動します。たとえば、次の INPUT ステートメントでは、最初の値を読み込んだ後、ポインターを列-2 に移動するように指示しています。

```
data test;
  input a @(a-3) b;
  datalines;
2
;
```

そのため、変数 A の値を読み込んだ後、ポインターは列 1 に移動します。変数 A と B には同じ値が割り当てられます。

無効なデータの処理方法

指定した変数の入力値に無効な文字が検出された場合、次の処理を実行します。

- 読み込み中の変数の値を欠損値に設定するか、INVALIDDATA=システムオプションに指定した値に設定します。詳細については、["INVALIDDATA= 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。
- 無効なデータに関するメッセージが SAS ログに出力されます。
- 無効な値が含まれる入力行と列番号を SAS ログに出力します。表示不能な文字は、16 進表記で表示されます。列番号を見やすくするために、入力行の上部に罫線が出力されます。

- 現在のオブザベーションの自動変数_ERROR_の値が 1 に設定されます。

エラーレポートのフォーマット修飾子を指定すると、SAS ログに出力される情報の量を制御することができます。?および??修飾子を指定すると、無効なデータを示すメッセージを出力しないようにします。ただし、??修飾子では、自動変数_ERROR_の値が 0 に設定されます。たとえば、次の 2 つのステートメントは同じ結果になります。

```
input x ?? 10-12;

input x ? 10-12;
_error_=0;
```

どちらの場合でも、変数 X の値が無効な場合は欠損値に設定されます。無効なデータの原因については、“[SAS が無効なデータを処理する方法](#)” ([SAS プログラマガイド: エssenシャル](#))を参照してください。

ファイルの終端

INPUT ステートメントがデータを最後まで読み込むと、ファイルの終端が検出されます。DATA ステップでファイルの終端に到達した後も他のレコードを読み込もうとすると、DATA ステップの実行が終了します。DATA ステップの実行を継続するには、INFILE ステートメントに END=オプションまたは EOF=オプションを使用します。その場合、ファイルの終端の検出時に、INPUT ステートメントの実行を停止しても DATA ステップの処理は継続するプログラムステートメントを作成することができます。詳細については、“[INFILE ステートメント](#)” ([143 ページ](#))を参照してください。

配列

INPUT ステートメントでは、配列参照を使用して入力データ値を読み込むことができます。配列参照を丸かっこで囲むと、ポインターコントロールに配列参照を使用できます。“[例 6: 文字変数でポインターの位置を指定する](#)” ([204 ページ](#))を参照してください。

配列の subscript にアスタリスク(*)を使用すると、明示的に定義済みの配列のすべての要素を読み込むことができます。1 次元配列または多次元配列を使用できます。subscript は中かっこ、大かっこ、丸かっこで囲んでください。このステートメントの形式を次に示します。

```
input array-name{*};
```

リスト入力、列入力、フォーマット入力で配列を使用できます。ただし、_TEMPORARY_で定義されている配列や、subscript にアスタリスクを使用する配列に値を読み込むことはできません。たとえば、次のステートメントでは変数 X1 から X100 を作成し、入力形式 2.を使用してデータ値を変数に割り当てます。

```
array x{100};
input x{*} 2.;
```

比較

- INPUT ステートメントでは、外部ファイルの生データ、または DATALINE ステートメントの後ろに指定されているインストリームのデータ行を読み込みます。

これらは、SAS への記述が必要です。SET ステートメントでは、SAS データセットを読み込みます。このデータセットには、データ値の記述情報が含まれていません。

- INPUT ステートメントはデータを読み込みます。PUT ステートメントはデータ値、テキスト文字列、またはその両方を SAS ログまたは外部ファイルに書き込みます。
- INPUT ステートメントは外部ファイルからデータを読み込みます。INFILE ステートメントは INPUT ステートメントで読み込むファイルや、ファイルを読み込む方法を制御するオプションを指定します。

例

例 1: INPUT ステートメントで複数のスタイルを使用する

この例では、1 つの INPUT ステートメントで複数の入力スタイルを使用します。

```
data club1;
  input Idno Name $18.
        Team $ 25-30 Startwght Endwght;
  datalines;
023 David Shaw    red  189 165
049 Amelia Serrano yellow 189 165
... more data lines ...
;
```

この表に入力スタイルの種類を示します。

変数	入力の種類
Idno、Startwght、Endwght	リスト入力
Name	フォーマット入力
Team	列入力

例 2: null INPUT ステートメントを使用する

この例では、引数を指定しない INPUT ステートメントを使用します。この DATA ステップを実行すると、SAS 変数を作成せずに、入力ファイルから出力ファイルにレコードがコピーされます。

```
data _null_;
  infile file-specification-1;
  file file-specification-2;
  input;
  put _infile_;
run;
```

例 3: 入力バッファー内のレコードを保持する

この例では、2 種類の入力データレコードを含むファイルを読み込み、読み込んだレコードから SAS データセットを作成します。1 つのデータレコードには、ある大学のコースに関する情報が保存されています。もう 1 つのデータレコードには、コースを専攻した学生の情報が保存されています。2 つのレコードを読み込み、出力形式が異なる複数の変数に値を割り当てるには、2 つの INPUT ステートメントを使用する必要があります。次に示すように、クラス情報を含むレコードの列 1 には C という値が格納されています。また、学生情報を含むレコードの列 1 には S という値が格納されています。

```
----+----1----+----2----+
C HIST101 Watson
S Williams 0459
S Flores 5423
C MATH202 Sen
S Lee 7085
```

どちらの INPUT ステートメントを使用するかを判断するため、このレコードの値を読み込むときに確認します。レコードに含まれる情報がクラスの情報が学生の情報を判断する変数のみを読み込む INPUT ステートメントを使用します。

```
data schedule(drop=type);
  retain Course Professor;
  input type $1. @;
  if type='C' then
    input course $ professor $;
  else if type='S' then
    do;
      input Name $10. Id;
      output schedule;
    end;
  datalines;
C HIST101 Watson
S Williams 0459
S Flores 5423
C MATH202 Sen
S Lee 7085
;
run;

proc print;
run;
```

最初の INPUT ステートメントは、各行の列 1 から変数 TYPE の値を読み込みます。INPUT ステートメントの末尾に後置@が指定されているので、DATA ステップの次の INPUT ステートメントでも同じ行を読み込みます。最初の INPUT ステートメントの後に記述されている IF-THEN ステートメントは、次の INPUT ステートメントで行の残りを読み込む前に、レコードに含まれる情報がクラスの情報が学生の情報を判断します。後置@を指定されていないこの INPUT ステートメントは、現在保持している行を解放します。RETAIN ステートメントでは、大学のコースの値が保持されます。この DATA ステップは、学生情報のレコードを読み込んだ後、オブザベーションを SCHEDULE データセットに書き込みます。

生成されたデータセット SCHEDULE の内容を PROC PRINT で出力した結果を次に示します。

アウトプット 2.18 データセット SCHEDULE

Obs	Course	Professor	Name	Id
1	HIST101	Watson	Williams	459
2	HIST101	Watson	Flores	5423
3	MATH202	Sen	Lee	7085

例 4: DATA ステップの反復間を通してレコードを保持する

この例は、1つの入力データレコードから複数のオブザベーションを作成する方法を示しています。各レコードには、変数 NAME と変数 AGE の値が複数含まれています。この DATA ステップでは、変数 NAME の値と変数 AGE の値をそれぞれ 1 つ読み込んでからオブザベーションを 1 つ出力します。レコード内のすべての入力値が処理されるまで、NAME と AGE の値の組み合わせを継続して読み込みます。

```
data test;
  input name $ age @@;
  datalines;
John 13 Monica 12 Sue 15 Stephen 10
Marc 22 Lily 17
;
```

この INPUT ステートメントでは後置 @@ が指定されているので、DATA ステップの反復間で入力ポインターが制御されます。生成される SAS データセットには、6 つのオブザベーションが含まれます。

例 5: 数値変数でポインターの位置を指定する

この例では、数値変数を使用してポインターの位置を指定します。生データファイルには、多国籍企業の複数の事業所別の従業員数が保存されています。入力データレコードは次のとおりです。

```
----+----1----+----2----+----3----+
8   New York   1 USA 14
5   Cary       1 USA 2274
3   Chicago    1 USA 37
22  Tokyo      5 ASIA 80
5   Vancouver  2 CANADA 6
9   Milano     4 EUROPE 123
```

最初の列は、事業所の所在地名の開始列位置を含みます。次の数値列は、地域カテゴリを示しています。また、各事業所の従業員数の前に地域名が示されています。

@*numeric-variable* ポインターコントロールと後置 @ を組み合わせて使用し、事業所の所在地名の開始列位置を指定します。レコードを読み込みには、INPUT ステートメントを 2 つ使用します。最初の INPUT ステートメントでは、@ *numeric-variable* ポインターコントロールの値を取得します。2 番目の INPUT ステートメントでは取得した値を使用して、ポインターの移動先となる列を指示します。

```
data office (drop=x);
  infile file-specification;
  input x @;
  if 1<=x<=10 then
```

```

        input @x City $9.;
    else do;
        put 'Invalid input at line ' _n_;
        delete;
    end;
run;

```

この DATA ステップでは、OFFICE データセットに 5 つのオブザベーションのみが書き込まれます。4 番目の入力データレコードは、変数 X の値が 10 を超えているので無効になります。そのため、2 番目の INPUT ステートメントは実行されません。そのかわり、PUT ステートメントによってメッセージが SAS ログに書き込まれます。また、DELETE ステートメントによってオブザベーションの処理が中止されます。

例 6: 文字変数でポインターの位置を指定する

この例では、文字変数を使用してポインターの位置を指定します。“[例 5: 数値変数でポインターの位置を指定する](#)” (203 ページ) で作成された OFFICE データセットには、文字変数 CITY が含まれています。この変数の値は、事業所の所在地を示しています。ここで、生データファイルから、他の値を読み込む必要が発生したと仮定します。もう 1 つ DATA ステップを追加することで、その値を検出するために、`@character-variable` ポインターコントロール、後置@、`@character-expression` ポインターコントロールを組み合わされます。

OFFICE データセットに含まれるオブザベーションが元の入力データレコードと同じ順で格納されている場合、次の DATA ステップを使用することができます。

```

data office2;
    set office;
    infile file-specification;
    array region {5} $ _temporary_
        ('USA' 'CANADA' 'SA' 'EUROPE' 'ASIA');
    input @city Location : 2. @;
    input @(trim(region{location})) Population : 4.;
run;

```

ARRAY ステートメントは、一時配列要素に初期値を割り当てます。この要素は、事業所の所在地の地域に対応しています。最初の INPUT ステートメントには、`@character-variable` ポインターコントロールが指定されています。各レコードから変数 CITY の値を示す文字列を走査します。次に、LOCATION の値を変数 CITY の次にある非ブランク列から読み込みます。LOCATION は、事業所の地域を示す数値になります。2 番目の INPUT ステートメントでは、`@character-expression` ポインターコントロールに配列参照を使用して、入力レコード内の変数 POPULATION の位置を指示します。また、この式では、TRIM 関数を使用して、文字値の末尾にあるブランクを取り除きます。このようにして、入力データの文字列と配列要素の値の完全一致を検出します。

生成されたデータセット OFFICE2 の内容を PROC PRINT で出力した結果を次に示します。

アウトプット 2.19 データセット OFFICE2

Obs	City	Location	Population
1	New York	1	14
2	Cary	1	2274
3	Chicago	1	37
4	Tokyo	5	80
5	Vancouver	2	6
6	Milano	4	123

例 7: ポインターを後方(左)に移動する

この例では、ポインターを戻すいくつかの方法を示します。

- 次の INPUT ステートメントでは、@ポインターコントロールを使用して列 26 から始まる変数 BOOK の値を読み込みます。次に、ポインターを同じ行の列 1 に戻し、変数 COMPANY の値を読み込みます。

```
input @26 book $ @1 company;
```

- 次の INPUT ステートメントでは、*+numeric-variable* または *+(expression)* を使用し、ポインターを 1 列戻します。次の 2 つのステートメントは同じ結果になります。

```
m=-1;
input x 1-10 +m y 2.;
input x 1-10 +(-1) y 2.;
```

関連項目

ステートメント:

- [“ARRAY ステートメント” \(40 ページ\)](#)
- [“INPUT ステートメント: 列” \(205 ページ\)](#)
- [“INPUT ステートメント: フォーマット” \(210 ページ\)](#)
- [“INPUT ステートメント: リスト” \(215 ページ\)](#)
- [“INPUT ステートメント: 名前付き” \(223 ページ\)](#)
- [“LIST ステートメント” \(244 ページ\)](#)

INPUT ステートメント: 列

指定した列から入力値を読み込み、読み込んだ値を対応する SAS 変数に代入します。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。

構文

INPUT *variable* <\$> *start-column*<--*end-column*>
<.*decimals*> <@ | @@>;

引数

variable

読み込む値を割り当てる変数を 1 つ指定します。

\$

変数の値が数値ではなく文字列であることを指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

start-column

読み込む値が含まれる入力レコードの開始列を指定します。

--*end-column*

読み込む値が含まれる入力レコードの最終列を指定します。

ヒント 変数の値が 1 列だけの場合、*end-column* の指定を省略することができます。

例 *end-column* が省略されているため、文字変数 GENDER の値は列 16 にのみ存在します。
input name \$ 1-10 pulse 11-13 waist 14-15 gender \$ 16;

.decimals

値を除算する 10 のべき乗を指定します。データに小数点が含まれる場合、小数点値は無視されます。

ヒント 入力値に小数点が明示的に含まれている場合、INPUT ステートメントで指定した小数点以下の桁数より優先されます。

例 “例 2: 10 進数を使用した各入力データの読み込み” (208 ページ)

“例 3: 10 進数を使用した入力レコードの読み込み” (209 ページ)

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目 [“ポインターコントロール” \(195 ページ\)](#)
目:

@@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

詳細

列入力の使用が求められる場合

列入力を使用する場合、INPUT ステートメントで、値の位置を示す列番号を変数名の後ろに指定します。列入力でデータを読み込む場合、データ値には次の属性が必要です。

- すべての入力データレコードで同じ列位置に表示される
- 標準的な数値形式または文字形式で構成される ¹

列入力にはこんな便利な機能があります。

- 文字値の中に埋め込みブランクを含めることができます。
- 文字値の長さは、1 - 32,767 文字です。
- 入力値がレコードのどの位置にあっても、任意の順序で読み込むことができます。
- 値または値の一部を繰り返し読み込むことができます。たとえば、次の INPUT ステートメントでは、ID の値として列 10 から 15 までを読み込んだ後、GROUP の値として列 13 を読み込みます。

1. 標準のデータ値や非標準のデータ値については、[SAS プログラマガイド: エssenシャル](#)を参照してください。

```
input id 10-15 group 13;
```

- フィールド内の前置ブランクと後置ブランクは無視されます。そのため、数値に 0 を示すブランクが含まれていたり、文字値の先頭と末尾にあるブランクを保持する場合は、入力形式を使用して値を読み込みます。詳細については、“[INPUT ステートメント: フォーマット](#)” (210 ページ) を参照してください。

欠損値

欠損データにプレースホルダーは必要ありません。INPUT ステートメントでは、ブランクのフィールドを欠損値として読み込みます。また、それ以外の値は正確に読み込みます。数値または文字列のフィールドにピリオドが 1 つだけ含まれている場合、この変数の値は欠損値に設定されます。

データ行の読み込み

DATALINES ステートメントの後ろに指定したデータレコード(インストリームデータ)には、80 の倍数を示す固定長になるようにブランクが追加されます。80 番目以降の列のデータを読み込むか切り捨てるかは、CARDIMAGE システムオプションの設定によって異なります。

可変長レコードの読み込み

デフォルトでは、可変長データレコードの読み込みには、FLOWOVER オプションを使用します。レコードに含まれる値の数が必要な数よりも少ない場合、INPUT ステートメントは次のデータレコードから値を読み込みます。可変長データを読み込むには、INFILE ステートメントに TRUNCOVER オプションを指定することをお勧めします。TRUNCOVER オプションは、レコードにブランクを追加して固定長レコードに変更する PAD オプションよりも便利です。詳細については、“[行の終わりを越えて読み込む](#)” (169 ページ) を参照してください。

例

例 1: 列入力での入力レコードの読み込み

次の DATA ステップは、列入力で入力データレコードを読み込む方法を示しています。

```
data scores;
  input name $ 1-18 score1 25-27 score2 30-32
        score3 35-37;
  datalines;
Joseph      11  32  76
Mitchel     13  29  82
Sue Ellen   14  27  74
;
```

例 2: 10 進数を使用した各入力データの読み込み

次の INPUT ステートメントでは、小数点以下 2 桁と指定して数値変数の入力データを読み込みます。

入力データ	ステートメント	結果
----+---1	input number 1-5 .2;	
2314		23.14
2		.02
400		4.00
-140		-1.40
12.236		12.23 1
12.2		12.2 1

1 INPUT ステートメントに指定した小数点以下の桁数は、入力値に小数点が含まれている場合は無効になります。

例 3: 10 進数を使用した入力レコードの読み込み

この DATA ステップでは、.decimals 引数を使用して入力行から値を読み込み、明示的な 10 進数がまだ定義されていないデータに 10 進数を挿入します。明示的な 10 進数を含むデータは変更されませんが、変換後の出力データのいずれかに発生する最大有効桁数に合わせてデータがパディングされます。

```
data product1 noobs;  
  input partcost 1-5 .2;  
  datalines;  
6857  
21  
400  
3.2  
12.56  
;  
proc print data=product1 noobs; run;
```

partcost
68.57
0.21
4.00
3.20
12.56

関連項目

ステートメント:

- [“INPUT ステートメント” \(186 ページ\)](#)

INPUT ステートメント: フォーマット

指定した形式で入力値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“DATA Step Acceleration” (SAS In-Database Products: User's Guide)を参照してください。
参照項目:	“SAS 入力形式” (SAS 出力形式と入力形式: リファレンス)

構文

INPUT *<pointer-control> variable informat. <@ | @@>;*

INPUT *<pointer-control>(variable-list) (informat-list)*
<@ | @@>;

INPUT *<pointer-control>(variable-list) (<n> informat.)*
<@ | @@>;

引数

pointer-control

入力バッファー内の指定した行または列に入力ポインターを移動させます。

参照項目: [“列ポインターコントロール” \(189 ページ\)](#)および[“行ポインターコントロール” \(191 ページ\)](#)

variable

読み込む値を割り当てる変数を 1 つ指定します。

要件 *(variable-list)*の後ろに*(informat-list)*を指定します。

例 [“例 1: フォーマット入力でポインターコントロールを使用する” \(214 ページ\)](#)

(variable-list)

読み込む値を割り当てる変数名のリストを指定します。

参照項目: [“変数と入力形式をグループ化する方法” \(213 ページ\)](#)

例 [“例 2: 入力形式リストを使用する” \(214 ページ\)](#)

informat.

変数の値の読み込みに使用する SAS 入力形式を指定します。

ヒント 実際の入力値に含まれる小数点は、数値入力形式による小数点の指定よりも優先されます。

参照項目: [SAS 出力形式と入力形式: リファレンス](#)の SAS 入力形式目:

例 [“例 1: フォーマット入力でポインターコントロールを使用する” \(214 ページ\)](#)

(informat-list)

入力形式のリストを指定します。このリストは、前の位置に指定した変数リストの値の読み込みに使用します。

INPUT ステートメントでは、(informat-list)に次を指定できます。

informat.

変数の値の読み込みに使用する入力形式を指定します。

pointer-control

値の読み込み位置を指示するポインターコントロール(@、#、/、+)のうちの 1 つを指定します。

n*

入力形式リストで n*に続けて指定した入力形式を n 回繰り返します。

例 次のステートメントは、7.2 入力形式を使用して GRADES1、GRADES2、GRADES3 を読み込み、5.2 入力形式を使用して GRADES4、GRADES5 を読み込みます。

```
input (grades1-grades5)(3*7.2, 2*5.2);
```

制限事項 (informat-list)は(variable-list)の後ろに指定する必要があります。

参照項目: [“変数と入力形式をグループ化する方法” \(213 ページ\)](#)

例 [“例 2: 入力形式リストを使用する” \(214 ページ\)](#)

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自

動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目 [“ラインホールド指定子の使用” \(196 ページ\)](#)
目:

@@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、後置@@と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

詳細

フォーマット入力の使用が求められる場合

フォーマット入力を使用する場合、変数名の後ろに入力形式を指定することにより、変数の値を読み込む方法を定義します。入力形式では、データの種類や入力値のフィールド長を指示します。入力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を読み込むこともできます。¹ SAS 入力形式については、[“入力形式について” \(SAS 出力形式と入力形式 リファレンス\)](#)を参照してください。

通常のフォーマット入力を使用する場合、対応する値が入力データに表示されるのと同じ順序で変数を指定する必要があります。ポインターコントロールを使用すると、変数を任意の順序で読み込むことができます。詳細については、[“INPUT ステートメント” \(186 ページ\)](#)を参照してください。

欠損値

通常、フォーマット入力では、数値の欠損値は 1 つのピリオドで示されます。また、文字の欠損値は 1 つの空白で示されます。フォーマット入力で空白がどのように解釈されるかは、使用する入力形式によって異なります。たとえば、\$CHAR.w と指定すると、空白は値の一部として読み込まれます。ただし、BZ.w と指定すると、空白は 0 に変換されます。

可変長レコードの読み込み

デフォルトでは、可変長データレコードの読み込みには、FLOWOVER オプションを使用します。レコードに含まれる値の数が必要な数よりも少ない場合、INPUT ステートメントは次のデータレコードから値を読み込みます。可変長データを読み込むには、INFILE ステートメントに TRUNCOVER オプションを指定することをお勧めし

1. 標準のデータ値や非標準のデータ値の情報については、[SAS プログラマガイド: エssenシャル](#)を参照してください。

ます。詳細については、“[行の終わりを越えて読み込む](#)” (169 ページ)を参照してください。

変数と入力形式をグループ化する方法

読み込む値があるパターンで配置されている場合、入力形式リストをグループ化することができます。グループ化した入力形式リストは、次の 2 つのリストで構成されます。

- 読み込み対象の変数名を丸かっこで囲んで指定したリスト
- ブランク区切りまたはカンマ区切りの対応する入力形式を丸かっこで囲んで指定したリスト

入力形式リストを使用すると INPUT ステートメントを簡潔に記述できます。これは、入力形式リストはすべての変数が読み込まれるまで繰り返し適用され、簡略化された番号付きの変数名が使用されるためです。入力形式リストを使用すると、個々の変数を指定する必要がなくなります。

例えば、5 つの変数 SCORE1～SCORE5 の値が、空白を挟まずに 1 つの値につき 4 列で格納されている場合、この INPUT ステートメントで値を読み込むことができます。

```
input (score1-score5) (4. 4. 4. 4.);
```

しかし、入力形式よりも多くの変数を指定した場合、INPUT ステートメントは入力形式リストを再利用して残りの変数を読み込みます。前述のステートメントは次のように簡略化できます。

```
input (score1-score5) (4.);
```

INPUT ステートメントには必要な数だけ入力形式リストを使用できますが、入力形式リストはネストさせないでください。変数リストにあるすべての値が読み込まれると、INPUT ステートメントは入力形式リストに残っている指示を無視します。この例については、“[例 3: 必要な数以上の入力形式が指定されている場合](#)” (214 ページ)を参照してください。

入力形式リストに n^* 修飾子を使用すると、後ろに指定した入力形式を n 回繰り返します。次に例を示します。

```
input (name score1-score5) ($10. 5*4.);
```

入力形式の保存方法

INPUT に指定した入力形式は、SAS データセットに保存されません。INFORMAT ステートメントや ATTRIB ステートメントに指定した入力形式は恒久的に保存されます。そのため、後続の DATA ステップでは、恒久的に保存された入力形式を使用してデータ値を読み込むことができます。入力形式を指定したり、PROC FSEDIT を使用して正しい形式でデータを入力する必要はありません。

比較

フォーマット入力に変数を読み込むときのポインターの動きは、列入力のポインターの動きと似ています。ポインターは入力形式に指定された長さを移動した後、次の列で停止します。列位置が揃っていないデータを入力形式を使用して読み込むに

は、**修飾リスト入力**を使用します。修飾リスト入力を使用すると、リスト入力の走査機能を使用できます。**“リスト入力の使用が求められる場合” (218 ページ)**を参照してください。

例

例 1: フォーマット入力でポインターコントロールを使用する

次の INPUT ステートメントは、入力形式とポインターコントロールを使用します。

```
data sales;
  infile file-specification;
  input item $10. +5 jan comma5. +5 feb comma5.
    +5 mar comma5.;
run;
```

このステートメントは、次の入力データレコードを読み込むことができます。

```
----+----1----+----2----+----3----+----4
trucks      1,382  2,789  3,556
vans        1,265  2,543  3,987
sedans       2,391  3,011  3,658
```

ITEM の値は、レコードの最初の 10 列から読み込まれます。ポインターは 11 列の位置で停止します。末尾にあるブランクを取り除いてから、ITEM の値がプログラムデータベクトルに書き込まれます。次に、ポインターを 5 列右に移動してから、INPUT ステートメントで COMMA5. 入力形式を使用して JAN の値を読み込みます。この入力形式は、カンマを含む数値の読み込みを実行し、フィールド長として 5 列を使用します。その後も、ポインターを 5 列ずつ右に移動してから、INPUT ステートメントで COMMA5. 入力形式を使用して FEB と MAR の値を読み込みます。

例 2: 入力形式リストを使用する

次の INPUT ステートメントは、文字入力形式 \$10. で変数 NAME の値を読み込みます。また、数値入力形式 4. で SCORE1 から SCORE5 までの変数の値を読み込みます。

```
data scores;
  input (name score1-score5) ($10. 5*4.);
  datalines;
Whittaker 121 114 137 156 142
Smythe    111 97  122 143 127
;
```

例 3: 必要な数以上の入力形式が指定されている場合

次の入力形式リストには、この INPUT ステートメント実行時に必要となる数よりも多い入力形式が指定されています。

```
data test;
  input (x y z) (2.,+1);
  datalines;
2 24 36
0 20 30
```


;

この INPUT ステートメントでは、入力形式 2. を使用して X の値を読み込みます。列ポインターコントロールは+1 と指定されているため、ポインターを 1 列だけ前方(右)に移動します。次に、入力形式 2. を使用して Y の値を読み込みます。列ポインターコントロールは+1 と指定されているため、再度ポインターを 1 列だけ前方(右)に移動します。最後に、入力形式 2. を使用して Z の値を読み込みます。3 回目の繰り返しでは、INPUT ステートメントはポインターコントロールの移動を指示する+1 を無視します。

関連項目

ステートメント:

- [“INPUT ステートメント” \(186 ページ\)](#)
- [“INPUT ステートメント: リスト” \(215 ページ\)](#)

INPUT ステートメント: リスト

入力データレコードの入力値を走査し、対応する SAS 変数に割り当てます。

該当要素: DATA ステップ

カテゴリ: ファイル操作

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。
このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User’s Guide\)](#)を参照してください。

構文

INPUT *<pointer-control>* *variable* *<\$>* *<&>* *<@ | @@>*;

INPUT *<pointer-control>* *variable* *<: | & | ~>*
<informat.> *<@ | @@>*;

引数

pointer-control

入力バッファ内の指定した行または列に入力ポインターを移動させます。

参照項目: [“列ポインターコントロール” \(189 ページ\)](#)および[“行ポインターコントロール” \(191 ページ\)](#)

例 [“例 2: ブランクが埋め込まれた文字データの読み込み” \(221 ページ\)](#)

variable

読み込む値を割り当てる変数を 1 つ指定します。

\$

変数の値を数値ではなく文字値として格納するように指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例 [“例 1: 単純リスト入力を使用した位置が揃っていないデータの読み込み” \(220 ページ\)](#)

&

文字値に 1 つ以上のブランクが含まれている場合に指定します。このフォーマット修飾子を指定すると、次の非ブランク列から値を、連続する 2 つのブランク、変数に定義された長さ、入力行の最後のいずれかにポインターが到達するまで読み込みます。

制限事項 & 修飾子は適用対象となる変数名と\$記号の後ろに指定する必要があります。

ヒント & 修飾子の後ろに入力形式を指定しても、連続する 2 つのブランクというフォーマット修飾子の終了条件は変わりません。

参照項目: [“修飾リスト入力” \(219 ページ\)](#)

例 [“例 2: ブランクが埋め込まれた文字データの読み込み” \(221 ページ\)](#)

:

INPUT ステートメントで変数の値の読み込みに使用する入力形式を指定できます。文字変数の場合、このフォーマット修飾子を指定すると、次の非ブランクの列から値を、次のブランクの列、長さを指定した変数、データ行の最後のいずれかにポインターが到達するまで読み込みます。数値変数の場合、このフォーマット修飾子を指定すると、次の非ブランク列の列から値を、次のブランクの列、データ行の最後のいずれかにポインターが到達するまで読み込みます。

ヒント 変数の長さが定義されていない場合、入力形式の長さに従って値を読み込んでから格納されます。

ポインターは、次のブランク列に到達するまで読み込みを継続します。ただし、フィールドの長さが入力形式の長さよりも長い場合、入力形式の長さまで切り捨てられます。

参照項目: [“修飾リスト入力” \(219 ページ\)](#)

例 [“例 3: 入力形式を使用した位置が揃っていないデータの読み込み” \(221 ページ\)](#)

[“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” \(222 ページ\)](#)

~

単一引用符、二重引用符、区切り文字を特別な方法で扱うように指示します。このフォーマット修飾子を指定すると、区切り文字ではなく文字として、引用符で囲まれた文字値内の区切り文字を読み込みます。また、変数に値を書き込む際、この引用符を保持します。

制限事項 INFILE ステートメントに DSD オプションを使用する必要があります。使用しない場合、INPUT ステートメントはこのオプションは無視します。

参照項目: [“修飾リスト入力” \(219 ページ\)](#)

例 [“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” \(222 ページ\)](#)

informat.

変数の値の読み込みに使用する入力形式を指定します。

ヒント 実際の入力値に含まれる小数点は、数値入力形式による小数点の指定より優先されます。

参照項目: [SAS 出力形式と入力形式: リファレンスの SAS 入力形式](#)

例 [“例 3: 入力形式を使用した位置が揃っていないデータの読み込み” \(221 ページ\)](#)

[“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” \(222 ページ\)](#)

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

@@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

詳細

リスト入力の使用が求められる場合

リスト入力では、入力データレコードでフィールドが表示されるのと同じ順序で、INPUT ステートメントに変数名を指定する必要があります。SAS ではデータ行を走査して次の値を検出しますが、値を区切るブランクは無視されます。リスト入力では、データが特定の列位置に存在していなくてもかまいません。ただし、区切り文字を変更しないかぎり、値と値の間は 1 つ以上のブランクで区切る必要があります。デフォルトでは、データ値の区切り文字は 1 つのブランク、または入力レコードの最後に設定されています。リスト入力では、あるデータをスキップして後続のデータを読み込むことはできませんが、データレコード内の指定した位置より後ろにあるすべてのデータを無視することはできます。ただし、ポインターコントロールを使用すると、データ値を読み込む順序を変更することができます。

リスト入力には、次の 2 種類あります。

- 単純リスト入力
- 修飾リスト入力

フォーマット修飾子を使用すると単純リスト入力の複数の制限を取り除くことができるため、修飾リスト入力により INPUT ステートメントを幅広い用途に使用できるようになります。参照: [“修飾リスト入力” \(219 ページ\)](#)。

単純リスト入力

単純リスト入力では、INPUT ステートメントで読み込めるデータの種類の制限があります。

- デフォルトでは、入力値は 1 つ以上のブランクで区切る必要があります。ブランク以外の区切り文字を使用する場合は、INFILE ステートメントに DLM=オプション、DLMSTR=オプション、DSD オプションのいずれかを使用します。
- 欠損値は、ブランクや連続する 2 つの区切り文字ではなく、ピリオドで表示します。
- 文字入力値には 8 バイトを超えた値を使用できません。ただし、あらかじめ LENGTH、ATTRIB、INFORMAT の各ステートメントでそれ以上の変数の長さ指定している場合を除きます。
- 区切り文字を変更しない場合、文字値にブランクを使用することはできません。
- 標準の数値形式または文字形式のデータでなければなりません。¹

1. 標準のデータ値や非標準のデータ値の情報については、[SAS プログラマガイド: エssenシャル](#)を参照してください。

修飾リスト入力

フォーマット修飾子を使用すると、リスト入力をより多目的に使えるようになります。フォーマット修飾子を次に示します。

フォーマット修飾子	用途
&	埋め込みのブランクを含む文字値を読み込みます。
で定義された値は保持されません。	入力形式が提供する追加指示を必要とする、列位置が不揃いのデータ値を読み込みます。 ¹
~	引用符で囲んだ文字値内の区切り文字を通常の文字として読み込みます。引用符は保持されます。

¹ フォーマット入力とポインターコントロールを使用して、列位置が揃っているデータ値を読み込みます。

たとえば、:(コロン)修飾子と入力形式を併用すると、8 バイトを超える文字値や非標準の値を含む数値を読み込むことができます。

リスト入力ではブランクが区切り文字として解釈されるので、ブランクを含む値を読み込むには、修飾リスト入力を使用します。&修飾子を使用すると、連続していないブランクを含む文字値を読み込みます。ただし、データ値は 2 つ以上のブランクで区切られている必要があります。リスト入力を使用して先頭、末尾、内部のどこかにブランクを含むデータを読み込むには、INFILE ステートメントに DLM=オプションまたは DLMSTR=オプションを使用して他の文字を区切り文字に指定します。
“例 5: 修飾リスト入力を使用した区切られたデータの読み込み” (222 ページ)を参照してください。区切り文字がブランクで、先頭、末尾、内部のどこかにブランクを含む入力データには、列入力またはフォーマット入力のいずれかを使用することをお勧めします。引用符で区切値が囲まれている場合、INFILE ステートメントに DSD オプションを指定すると、リスト入力を使用できます。

比較

修飾リスト入力とフォーマット入力の違い

修飾リスト入力には走査機能があるため、入力形式を使用して、列位置が不揃いのデータを読み込みます。フォーマット入力では、列入力と同じようにポインターを移動させて変数の値を読み込みます。ポインターは入力形式に指定された長さを移動した後、次の列で停止します。

次の DATA ステップでは、修飾リスト入力で最初のデータを読み込み、フォーマット入力をで 2 番目のデータを読み込みます。

```
data jansales;
  input item : $10. amount comma5.;
datalines;
trucks 1,382
vans 1,235
sedans 2,391
```

;

ITEM は修飾リスト入力で読み込まれます。INPUT ステートメントは、ポインターがブランクに到達した時点で読み込みを停止します。ポインターは、このフィールドの末尾から 2 列先(右)に移動します。この位置からフォーマット入力で AMOUNT の値を読み込みます。

一方、フォーマット入力は、フィールドの総幅を読み込むまで読み込みを続けます。次の INPUT ステートメントでは、フォーマット入力を使用して両方のデータ値を読み込みます。

```
input item $10. +1 amount comma5.;
```

フォーマット入力でデータを正しく読み込むには、次に示すように 2 番目のデータ値が最初の値から 10th 列目に出現する必要があります。

```
----+----1----+----2
trucks  1,382
vans    1,235
sedans  2,391
```

また、フォーマット入力で ITEM の値を読み込んだ後、ポインターコントロールを +1 と指定し、AMOUNT の値の開始位置となる列にポインターを移動する必要があります。

データに引用符が含まれている場合

INFILE ステートメントに DSD オプションを使用して区切り文字をカンマに変更した場合、値を変数に書き込む前にデータに含まれる引用符が取り除かれます。また、INPUT ステートメントでチルダ(~)修飾子を使用すると、引用符を値の一部として保持することができます。

例

例 1: 単純リスト入力を使用した位置が揃っていないデータの読み込み

次の DATA ステップの INPUT ステートメントは、単純リスト入力を使用して入力データレコードを読み込みます。

```
data scores;
  input name $ score1 score2 score3 team $;
  datalines;
Joe 11 32 76 red
Mitchel 13 29 82 blue
Susan 14 27 74 green
;
```

次の INPUT ステートメントは、前述のデータ行にある最初の 4 つのフィールドだけを読み込みます。この例で示すように、レコード内のすべてのデータを読み込む必要はありません。

```
input name $ score1 score2 score3;
```

例 2: ブランクが埋め込まれた文字データの読み込み

次の DATA ステップの INPUT ステートメントでは、& フォーマット修飾子とリスト入力を組み合わせて、ブランクを含む文字値を読み込みます。

```
data list;
  infile file-specification;
  input name $ & score;
run;
```

このステートメントは、次の入力データレコードを読み込むことができます。

```
----+----1----+----2----+----3----+
Joseph 11 Joergensen red
Mitchel 13 Mc Allister blue
Su Ellen 14 Fischer-Simon green
```

INPUT ステートメントに適用する&修飾子の変数の後ろに指定されています。このフォーマット修飾子は NAME の後ろにあるので、入力データレコードの NAME フィールドと SCORE フィールドの間は 2 つ以上のブランクを使用して区切られている必要があります。

次に示すように、フォーマット修飾子と入力形式を併用することもできます。

```
input name $ & +3 lastname & $15. team $;
```

また、この INPUT ステートメントでは読み込むデータは同じですが、入力レコードにあるすべてのデータを読み込む必要はないことを示しています。列ポインターコントロールが+3 と指定されているので、ポインターは SCORE フィールドの位置を通過して LASTNAME フィールドと TEAM フィールドの値を読み込みます。

例 3: 入力形式を使用した位置が揃っていないデータの読み込み

次の DATA ステップでは、入力形式付きの修飾リスト入力でデータ値を読み込みます。

```
data jansales;
  input item : $10. amount;
  datalines;
trucks 1382
vans 1235
sedans 2391
;
```

\$10.入力形式では、文字変数の値として最大 10 文字まで読み込むことができます。

例 4: 入力形式付きのリスト入力を使用したカンマ区切りデータの読み込み

次の DATA ステップでは、INFILE ステートメントに DELIMITER=オプションを指定して、ブランクのかわりにカンマで区切られた値をリスト入力を読み込みます。例では入力形式を使用して日付を読み込み、出力形式を使用してその日付を書き込みます。

```
data scores2;
  length Team $ 14;
  infile datalines delimiter=',';
  input Name $ Score1-Score3 Team $ Final_Date:MMDDYY10.;
```



```

format final_date weekdate17.;
datalines;
Joe,11,32,76,Red Racers,2/3/2007
Mitchell,13,29,82,Blue Bunnies,4/5/2007
Susan,14,27,74,Green Gazelles,11/13/2007
;
proc print data=scores2;
var Name Team Score1-Score3 Final_Date;
title 'Soccer Player Scores';
run;

```

アウトプット 2.20 カンマで区切られたデータの出力

Soccer Player Scores						
Obs	Name	Team	Score1	Score2	Score3	Final_Date
1	Joe	Red Racers	11	32	76	Sat, Feb 3, 2007
2	Mitchell	Blue Bunnies	13	29	82	Thu, Apr 5, 2007
3	Susan	Green Gazelles	14	27	74	Tue, Nov 13, 2007

例 5: 修飾リスト入力を使用した区切られたデータの読み込み

次の DATA ステップでは、INFILE ステートメントに DSD オプションを指定し、INPUT ステートメントにチルダ(~)フォーマット修飾子を指定します。ここでは、文字データに含まれる引用符は保持されます。また、引用符で囲まれた文字列に含まれる区切り文字は、区切り文字ではなく文字として読み込まれます。

```

data scores;
infile datalines dsd;
input Name : $9. Score1-Score3
      Team ~ $25. Div $;
datalines;
Joseph,11,32,76,"Red Racers, Washington",AAA
Mitchel,13,29,82,"Blue Bunnies, Richmond",AAA
Sue Ellen,14,27,74,"Green Gazelles, Atlanta",AA
;
proc print; run;

```

実行後に作成された SCORES データセットを PROC PRINT で出力した結果を次に示します。TEAM の値には引用符が含まれています。

アウトプット 2.21 SCORES データセット

Obs	Name	Score1	Score2	Score3	Team	Div
1	Joseph	11	32	76	"Red Racers, Washington"	AAA
2	Mitchel	13	29	82	"Blue Bunnies, Richmond"	AAA
3	Sue Ellen	14	27	74	"Green Gazelles, Atlanta"	AA

関連項目

ステートメント:

- [“INFILE ステートメント” \(143 ページ\)](#)
- [“INPUT ステートメント” \(186 ページ\)](#)
- [“INPUT ステートメント: フォーマット” \(210 ページ\)](#)

INPUT ステートメント: 名前付き

変数名と等号の後ろに続くデータ値を読み込み、読み込んだ値を対応する SAS 変数に割り当てます。

該当要素:	DATA ステップ
カテゴリ:	ファイル操作
種類:	実行
制限事項:	このステートメントは CAS で実行する DATA ステップではサポートされていません。 このステートメントは、In-database DATA ステップではサポートされていません。 詳細については、 “DATA Step Acceleration” (SAS In-Database Products: User’s Guide) を参照してください。

構文

INPUT *<pointer-control>* *variable=* *<\$>* *<@ | @@>;*

INPUT *variable=* *<\$>* *start-column<-end-column>*
<@ | @@>;

引数

pointer-control

入力バッファ内の指定した行または列に入力ポインターを移動させます。

参照項目: [“列ポインターコントロール” \(189 ページ\)](#)および[“行ポインターコントロール” \(191 ページ\)](#)

variable=

INPUT ステートメントで値を読み込む変数名を指定します。入力データレコードでは、次の形式でフィールドを記述します。

variable=value

ヒント 変数に入力形式を関連付けるには、INFORMAT ステートメントを使用します。

参照項目: [SAS 出力形式と入力形式 リファレンス](#)の SAS 入力形式

例 [“例 3: 名前付き入力と他の入力スタイルを使用する” \(227 ページ\)](#)

\$

変数の値を数値ではなく文字値として格納するように指定します。

ヒント この変数が文字変数と事前に定義されている場合、\$を指定する必要はありません。

例 [“例 3: 名前付き入力と他の入力スタイルを使用する” \(227 ページ\)](#)

start-column

INPUT ステートメントで変数の入力データレコードの走査を開始する列位置を指定します。変数名が始まる列位置でなくてもかまいません。

-end-column

変数のデフォルトの長さを指定します。

@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの同一の反復内で保持されます。このラインホールド指定子は、**後置@**と呼ばれます。

制限事項 後置@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@を指定すると、次の INPUT ステートメントによって、現在の入力レコードの開放や入力バッファへの次のレコードの読み込みが自動的に実行されなくなります。後置@は、同じレコードを何度も読み込む必要がある場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

@@

次の INPUT ステートメントの実行時に使用できるように入力行を保持します。DATA ステップの反復間を通して保持されます。このラインホールド指定子は、**後置@@**と呼ばれます。

制限事項 後置@@は、INPUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント 後置@@は、入力行に複数のオブザベーションの値が含まれる場合に便利です。

参照項目: [“ラインホールド指定子の使用” \(196 ページ\)](#)

詳細

名前付き入力の使用が求められる場合

名前付き入力では、変数名と等号の後ろに変数の値が続く入力データレコードを読み込みます。INPUT ステートメントでは、入力ポインタの現在位置にある入力データレコードを読み込みます。入力データレコードの先頭に INPUT ステートメントの名前付き入力では読み込めないデータ値が含まれる場合、他の入力スタイルを指定してそのデータ値を読み込むようにしてください。ただし、いったん INPUT ステートメントで名前付き入力での読み込みを開始した場合は、残りのすべての値をこの形式で読み込む必要があります。[“例 3: 名前付き入力と他の入力スタイルを使用する” \(227 ページ\)](#)を参照してください。

INPUT ステートメントの変数は、データレコードに値が表示されるのと同じ順序で指定する必要はありません。また、レコードに含まれる各フィールドに対して変数を指定する必要もありません。ただし、他のステートメント(ATTRIB、FORMAT、INFORMAT、または LENGTH などのステートメント)で使用する変数を INPUT ステートメントに指定していなくても、入力データにその値が含まれる場合は、INPUT ステートメントによって値が自動的に読み込まれます。その場合、変数が初期化されていないことを示すメッセージが SAS ログに出力されます。

名前付き入力のすべてのデータ値に変数を指定しない場合、自動変数 `_ERROR_` の値が 1 に設定され、メッセージが SAS ログに出力されます。

```
data list;
  input name=$ age=;
  datalines;
  name=John age=34 gender=M
  ;
```

SAS ログに出力されるメッセージには、GENDER が定義されていないので自動変数 `_ERROR_` の値が 1 に設定されたことが示されます。

制限事項

- 名前付き入力での読み込みを開始した後に、他の入力スタイルに切り替えたり、ポインタコントロールを使用することはできません。入力データレコードに含まれる残りの値はすべて、`variable=value` の形式で記述されている必要があります。名前付き入力の形式で記述されていない値は、無効なデータとして扱われます。
- 現在の入力行の後にも名前付き入力の値が続く場合、入力行の最後にスラッシュ (/) を使用します。スラッシュを指定すると、ポインタを次の行に移動させてから名前付き入力による読み込みを続けます。ここでは、2 つの入力ラインに分割された入力レコードの例を示します。

```
name=John /
age=34
```

- ブランクが中に含まれる文字値の読み込みに名前付き入力を使用する場合、リスト入力と同じように、データ値の前後にブランクを 2 つ追加します。[“例 4: 値にブランクが含まれる文字変数を読み込む” \(227 ページ\)](#)を参照してください。
- アスタリスクや式の subscript を使用した配列参照はできません。

例

例 1: リスト入力と名前付き入力を使用する

次の DATA ステップでは、リスト入力と名前付き入力を使用して、入力データレコードを読み込みます。

この INPUT ステートメントでは、ID 変数の値をリスト入力を読み込みます。残りの変数である NAME、GENDER、AGE、DOB の値は名前付き入力を読み込みます。また、INPUT ステートメントで読み込む変数 NAME の文字値が 8 文字で切り捨てられないように、LENGTH ステートメントを指定しています。

```
data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8.;
  input id name= gender= age= dob=;
  datalines;
4798 name=COLIN gender=m age=23 dob=16/02/75
2653 name=MICHELE gender=f age=46 dob=17/02/73
;
proc print data=list; run;
```

Obs	name	gender	dob	id	age
1	COLIN	m	5525	4798	23
2	MICHELE	f	4796	2653	46

例 2: 名前付き入力で変数を任意の順序で使用する

この DATA ステップでも、前述の例と同じデータを使用し、リスト入力と名前付き入力を入力データレコードを読み込みます。ただし、この例では、最初に読み込む ID の値を除き、読み込む 2 行の入力データでの値の順序は異なります。

```
data list;
  length name $ 20 gender $ 1;
  informat dob ddmmyy8.;
  input id dob= name= age= gender=;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE dob=17/02/73 age=46 gender=f
;
proc print data=list; run;
```

Obs	name	gender	dob	id	age
1	COLIN	m	5525	4798	23
2	MICHELE	f	4796	2653	46

例 3: 名前付き入力と他の入力スタイルを使用する

次の DATA ステップでは、リスト入力と名前付き入力を使用して入力データレコードを読み込みます。この例では、NAME 変数と GENDER 変数の長さタイプを INPUT ステートメントで直接指定しています。

この INPUT ステートメントでは、最初の変数 ID の値をリスト入力を読み込みます。残りの変数である NAME、GENDER、DOB の値は名前付き入力を読み込みます。これら変数の値は、データレコードにある順序で読み込まれるわけではありません。NAME=に\$20.と入力形式を指定しているので、INPUT ステートメントでの読み込み時に文字値が 8 文字で切り捨てられることはありません。INFORMAT ステートメントで変数 DOB を参照するように指定されているので、INPUT ステートメントで DOB=フィールドの値を読み込みます。このステートメントでは、AGE=フィールドはともにスキップされます。DOB が初期化されていないこと、AGE が定義されていないこと、自動変数_ERROR_が 1 に設定されたことを示すメッセージが SAS ログに出力されます。

```
data list;
  input id name=$20. gender=$;
  informat dob ddmmyy8.;
  datalines;
4798 gender=m name=COLIN age=23 dob=16/02/75
2653 name=MICHELE age=46 gender=f
;
proc print data=list; run;
```

Obs	id	name	gender	dob
1	4798	COLIN	m	5525
2	2653	MICHELE	f	.

例 4: 値にブランクが含まれる文字変数を読み込む

次の DATA ステップでは、値にブランクが含まれる文字変数を名前付き入力を読み込みます。

変数 HEADER の値である AGE=60 AND UP の前後には、2 個のスペースが挿入されています。このフィールドには、等号も含まれています。

```
data list2;
  informat header $30. name $15.;
  input header= name=;
  datalines;
header= age=60 AND UP name=PHILIP
;
proc print data=list2; run;
```

Obs	header	name
1	age=60 AND UP	PHILIP

関連項目

ステートメント:

- [“INPUT ステートメント” \(186 ページ\)](#)

KEEP ステートメント

出力 SAS データセットに含める変数を指定します。

該当要素:	DATA ステップ
カテゴリ:	CAS In-Database 処理 情報
種類:	宣言

構文

KEEP *variable-list*;

引数

variable-list

出力データセットに書き込む変数の名前を指定します。

ヒント SAS で使用可能な任意の形式で変数をリストします。

詳細

KEEP ステートメントでは、DATA ステップで指定した変数のみを 1 つまたは複数のデータセットに書き込みます。KEEP ステートメントは、同一の DATA ステップ内で作成されたすべての SAS データセットに適用されます。また、ステップのどの位置に指定してもかまいません。KEEP または DROP ステートメントが存在しない場合、DATA ステップで作成されたすべてのデータセットにすべての変数が含まれます。

同じ変数が DROP ステートメントと KEEP ステートメントの両方にリストされている場合、ステートメントの順序に関係なく、DROP が KEEP よりも優先され、変数は削除されます。

注: 同一の DATA ステップで KEEP ステートメントと DROP ステートメントの両方を使用しないでください。

比較

- SAS PROC ステップでは、KEEP ステートメントは使用できません。KEEP=データセットオプションは使用できます。
- KEEP ステートメントは DATA ステートメントで指定したすべての出力データセットに適用されます。データセットごとに異なる変数を書き込むには、KEEP=データセットオプションを使用します。
- DROP ステートメントは、出力データセットから除外する変数を指定する同種のステートメントです。
- KEEP ステートメントでは出力データセットに含める変数を指定し、DROP ステートメントでは出力データセットから除外する変数を選択します。サブセット化 IF ステートメントでは、オブザベーションを選択します。
- KEEP ステートメントと RETAIN ステートメントを混同しないでください。RETAIN ステートメントは、DATA ステップの繰り返し間に変数の値を保持するものです。KEEP ステートメントは変数の値には影響を与えず、出力データセットに含める変数の指定のみ実行します。

例

例 1: KEEP ステートメントの基本的な使用法

これらの例では、KEEP ステートメントで変数をリストする場合の正しい構文を示します。

```
keep name address city state zip phone;  
keep rep1-rep5;
```

例 2: データセットでの変数の保持

この例では、KEEP ステートメントを使用して、出力データセットに変数 NAME および AVG のみを含めます。AVG の計算に使用される変数 SCORE1 から SCORE20 は、データセット AVERAGE には書き込まれません。

```
data average;  
  keep name avg;  
  infile file-specification;  
  input name $ score1-score20;  
  avg=mean(of score1-score20);  
run;
```

関連項目

データセットオプション:

- [“KEEP= データセットオプション” \(SAS データセットオプション: リファレンス\)](#)

ステートメント:

- “DROP ステートメント” (98 ページ)
- “IF ステートメント: サブセット化” (137 ページ)
- “RETAIN ステートメント” (344 ページ)

LABEL ステートメント

説明を示すラベルを変数に割り当てます。

該当要素: DATA ステップまたは PROC ステップ
 カテゴリ: CAS
 情報
 In-Database 処理
 種類: 宣言

構文

形式 1: 1 つ以上の変数に記述的なラベルを作成します。

LABEL *variable-1* = '*text-string*' <...*variable-n* = '*text-string*'>;

形式 2: 1 つ以上の変数のラベルを削除します。

LABEL *variable-1* = '' <...*variable-n*=''>; | *variable-1* = <...*variable-n*=>;

引数

variable

ラベルを付ける変数を指定します。複数の変数のラベルは、1 つの LABEL ステートメントで指定できます。

```
data test;
  L = 5;
  W = 10;
  label L = 'Length' W = 'Width';
run;
```

text-string

最大 256 バイトのラベルを指定します。

```
data test;
  L = 5;
  label L = 'Length';
run;
```

制 限 ラベルにセミコロン(;)や等号(=)が含まれている場合は、ラベルのテキストを単一引用符または二重引用符で囲む必要があります。

```
label N = 'Number = ';
```


事項 ラベルに単一引用符(')が含まれている場合は、ラベルのテキストを二重引用符で囲む必要があります。

```
label Name = "Owner's Last Name";
```

JAVAIMG のデバイスでは、変数名の置き換えを一部サポートしています。変数に指定したラベルのテキストが許可されたスペースを超える場合、軸や凡例のタイトルをラベリングするプロシジャのかわりに、変数名が使用されます。SAS/GRAPH GPLOT プロシジャを除き、JAVAIMG デバイスが指定されていない場合、変数名は使用されません。

注 ラベルにセミコロン、引用符、等号が含まれていない限り、引用符は必要ありません。

LABEL ステートメントは、最初の変数の後に等号(=)があるものと想定します。それ以外の文字が見つかった場合、エラーが発生します。LABEL ステートメントは、最初の変数の直後に続く等号の後から、等号が直後に続く別の変数に達するまでの区間に存在するすべての文字を、引用符の有無にかかわらず、ラベルの一部と見なします。次の例では、変数 x のラベルは、**this is x y 4 =this is y** になります。なぜなら、等号が直後に続く変数は z であるためです。

```
data test;
  x = 1;
  y = 1;
  z = 1;
  label x = 'this is x' y 4 = 'this is y' z = 'This is z';
run;
```

次のような LABEL ステートメントでも、変数 x のラベルは前述の例と同じになります。

```
label x = this is x y 4 = this is y z = This is z;
```

参照項目: “文字定数と文字変数” (SAS プログラマガイド: エッセンシャル).

変数からラベルを削除します。ブランク 1 つを引用符で囲むと、指定済みのラベルが取り消されます。1 つの LABEL ステートメントで複数の変数からラベルを削除できます。

```
data test;
  L=5; W=10;
  label L = ' ' W = ' ';
run;
```

また、等号の後に何も指定しないでラベルを削除することもできます。

```
data test;
  L=5; W=10;
  label L = W = ;
run;
```

詳細

DATA ステップ内で LABEL ステートメントを使用すると、変数を含む SAS データセットのディスクリプター情報が変更され、ラベルとその変数は恒久的に関連付けられます。1 つの LABEL ステートメントで複数の変数にラベルを割り当てることができます。

比較

ATTRIB ステートメントと LABEL ステートメントは、どちらも変数へのラベルの割り当てや、変数が割り当てられたラベルの変更を実行することができます。

LABEL ステートメントは、DATA ステップおよび一部の PROC ステップ内で使用できます。ラベルが DATA ステップまたは **PROC DATASETS** で変数に割り当てられている場合、そのラベルは出力データセットディスクリプターに永久的に割り当てられます。**PROC PRINT** などの一部の PROC では、プロシジャ中に使用するためにラベルを変数に一時的に関連付けることができます。

この例では、レポート作成時にラベルを作成するさまざまな方法を示します。

DATA ステップで LABEL ステートメントを指定して、変数 MSRP に永続的なラベルを割り当てます。

```
data work.cars;
  set sashelp.cars;
  label MSRP = Sticker Price;
run;
```

PROC DATASETS ステートメントで LABEL ステートメントを指定して、変数 Invoice に永続的なラベルを割り当てます。

```
proc datasets library = work;
  modify cars;
  label Invoice = Dealer Cost;
quit;
```

PROC PRINT **LABEL ステートメント**, を使用すると、出力レポートで変数名のかわりにラベルを表示できます。出力に変数名ではなくラベル名を表示するには、PROC PRINT ステートメントで LABEL ステートメントを指定します。次に、DATA ステップ LABEL ステートメントを指定して、変数 Type にラベルを割り当てます。

```
proc print data=work.cars label;
  label Type = Style;
  var Make Model MSRP Invoice Type;
run;
```

出力レポートには、Sticker Price、Dealer Cost、Style という 3 つの新しいラベルが含まれています。

Obs	Make	Model	Sticker Price	Dealer Cost	Style
1	Acura	MDX	\$36,945	\$33,337	SUV
2	Acura	RSX Type S 2dr	\$23,820	\$21,761	Sedan
3	Acura	TSX 4dr	\$26,990	\$24,647	Sedan

PROC CONTENTS を使用すると、出力データセットに永久的に割り当てられたラベルを表示できます。この例では、データセット work.cars の永続ラベルと変数名を含む記述子情報が表示されます。

```
proc contents data=work.cars;
run;
```

新しい永久ラベル Sticker Price と Dealer Cost が、Invoice 変数と MSRP 変数に割り当てられるようになりました。PROC PRINT ステートメントの実行中に Type 変数に割り当てられた Style ラベルは、出力データセットに保持されません。

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
9	Cylinders	Num	8		
5	DriveTrain	Char	5		
8	EngineSize	Num	8		Engine Size (L)
10	Horsepower	Num	8		
7	Invoice	Num	8	DOLLAR8.	Dealer Cost
15	Length	Num	8		Length (IN)
11	MPG_City	Num	8		MPG (City)
12	MPG_Highway	Num	8		MPG (Highway)
6	MSRP	Num	8	DOLLAR8.	Sticker Price
1	Make	Char	13		
2	Model	Char	40		
4	Origin	Char	6		
3	Type	Char	8		
13	Weight	Num	8		Weight (LBS)
14	Wheelbase	Num	8		Wheelbase (IN)

PROC ステップ内で LABEL ステートメントを使用する方法の詳細については、[Base SAS プロシジャガイド](#)を参照してください。

例

例 1: ラベルの指定

いくつかの LABEL ステートメントの例を次に示します。

```
label compound=Type of Drug;
label date="Today's Date";
label n='Mark's Experiment Number';
label score1="Grade on April 1 Test"
      score2="Grade on May 1 Test";
```

例 2: ラベルの取り消し

定義済みのラベルを取り消す例を次に示します。

```
data rtest;
set rtest;
label x='';
run;
```

関連項目

ステートメント:

- [“ATTRIB ステートメント” \(52 ページ\)](#)

Label: ステートメント

他のステートメントから参照されるステートメントを識別します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	宣言

構文

label: statement;

引数

label

SAS 名を指定します。SAS 名の後ろにはコロン(:)が必要です。*label* 引数は必ず指定する必要があります。

statement

null ステートメント(;)など、実行ステートメントを指定します。*statement* 引数は必ず指定する必要があります。

制限事項 DATA ステップ内では、同一のラベルを複数のステートメントに指定できません。

DATA ステップ内のステートメントにラベルを設定する場合、同じステップ内のステートメントやオプションからはそのラベルを参照する必要があります。

ヒント null ステートメントにもラベルを設定できます。
ABC;

詳細

ステートメントラベルは、GOTO ステートメント、LINK ステートメント、FILE ステートメントの HEADER=オプション、INFILE ステートメントの EOF=オプションのいずれかの出力先を識別します。

比較

LABEL ステートメントは、記述的なラベルを変数に割り当てます。ステートメントラベルは、同一の DATA ステップ内で GOTO ステートメントなどの他のステートメントから参照される 1 つのステートメントまたはステートメントのグループを識別します。

例: 別のステートメントへの移動

この例では、Stock=0 であれば、GOTO ステートメントは SAS を reorder というラベルが設定されているステートメントに移動します。Stock の値が 0 でなければ、RETURN ステートメントまで処理を続行した後、次のオブザベーションを処理するために DATA ステップの先頭に戻ります。

```
data Inventory Order;
  input Item $ Stock @;
  /* go to label reorder: */
  if Stock=0 then go to reorder;
  output Inventory;
  return;
  /* destination of GOTO statement */
  reorder: input Supplier $;
  put 'ORDER ITEM ' Item 'FROM ' Supplier;
  output Order;
  datalines;
  milk 0 A
  bread 3 B
  ;
```

関連項目

ステートメント:

- [“GOTO ステートメント” \(135 ページ\)](#)
- [“LINK ステートメント” \(242 ページ\)](#)

ステートメントオプション:

- [“EOF=label” \(150 ページ\)](#)(INFILE ステートメント)
- [“HEADER=label” \(114 ページ\)](#)オプション(FILE ステートメント)

LEAVE ステートメント

現在のループの処理を中止し、シーケンス内の次のステートメントから再開します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

LEAVE;

引数なし

LEAVE ステートメントは現在の DO ループや SELECT グループの処理を中止し、この DO ループや SELECT グループの後に続く次のステートメントから DATA ステップの処理を再開します。

詳細

LEAVE ステートメントを使用すると、条件に基づいて DO ループまたは SELECT グループを途中で終了することができます。

比較

- LEAVE ステートメントは、現在のループの処理を終了します。CONTINUE ステートメントは、ループの現在の繰り返し処理を中止し、次の繰り返し処理から再開します。
- DO ループまたは SELECT グループ内で LEAVE ステートメントを使用します。CONTINUE ステートメントは DO ループのみで使用できます。

例: 指定した条件で DO ループの処理を中止する

次の DATA ステップでは、LEAVE ステートメントを使用して DO ループの処理を中止します。この例では、IF/THEN ステートメントを使用して BONUS の値をチェッ

クします。BONUS の値が指定した最大値である 500 になると、LEAVE ステートメントは DO ループの処理を中止します。

```
data week;
  input name $ idno start_yr status $ dept $;
  bonus=0;
  do year= start_yr to 1991;
    if bonus ge 500 then leave;
    bonus+50;
  end;
  datalines;
Jones 9011 1990 PT PUB
Thomas 876 1976 PT HR
Barnes 7899 1991 FT TECH
Harrell 1250 1975 FT HR
Richards 1002 1990 FT DEV
Kelly 85 1981 PT PUB
Stone 091 1990 PT MAIT
;
```

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“SELECT ステートメント” \(352 ページ\)](#)

LENGTH ステートメント

文字変数や数値変数を格納する場合はバイト数を、VARCHAR 変数を格納する場合は文字数を指定します。

該当要素: DATA ステップ

カテゴリ: CAS
In-Database 処理
情報

種類: 宣言

注意: **小数を含んでいる数値の短縮化を防ぎます。** 変数に小数が含まれる場合、数値変数の有効桁数はその数値変数の長さに密接に関連します。整数を含む変数の長さは、各動作環境向けの SAS ドキュメント内のルールに従って短くすることができます。ただし、小数を含む変数の長さを短くすると、重要な有効桁数が削除される場合があります。

構文

LENGTH *variable-specification(s)* <DEFAULT=*n*>;

引数

variable-specification

この引数は必須です。次の形式を使用します。

variable(s) <\$ *length* | *length* | VARCHAR(*length*) | VARCHAR(*)>

variable

長さを割り当てる 1 つまたは複数の変数(DATA ステップ内の任意の変数を含む)を指定します。出力データセットからは除外される変数も含められます。

制限事項 配列参照は指定できません。

ヒント 指定した変数が文字変数または VARCHAR 変数の場合、長さはプログラムデータベクトルと出力データセットに適用されます。指定した変数が数値変数の場合、長さは出力データセットにのみ適用されます。

\$

直前にある変数が、CHAR データタイプの文字変数であることを指定します。

length

変数値を格納するための数値定数を指定します。数値変数と文字変数の場合、この定数は変数に格納される最大バイト数です。

範囲 文字変数の場合、1～32767 バイト。数値変数の場合、3～8 バイト。数値変数に指定できる長さの最小値は、ユーザーのシステムが使用する浮動小数点形式によって異なります。ほとんどのシステムでは IEEE 浮動小数点形式が使用され、最小値は 3 バイトです。

制限事項 CAS では、8 バイトより短い数値変数は 8 バイトとして扱われます。

VARCHAR

直前にある変数が VARCHAR 型であることを指定します。

length

VARCHAR 変数に格納される最大文字数を指定します。

たとえば、length v varchar(100);は、VARCHAR 変数に最大 100 文字が格納されることを意味します。

範囲 VARCHAR 変数の場合、1 文字から 536,870,911 文字までです。

制限事項 VARCHAR 変数に文字定数を割り当てる場合、その文字定数は 32767 バイトまでに制限されます。

要件 DATA ステップで変数を読み込んだり書き出したりする際に、変数を VARCHAR データ型として保持したい場合は、CAS エンジンが必要となります。

参照 変数の自動宣言の詳細については、“データ型の暗黙的な宣言のサ
項目: ポート” ([SAS Cloud Analytic Services: ユーザーガイド](#))を参照して
ください。

VARCHAR 変数が自動的に文字値に変換される仕組みについては、
“暗黙的および明示的なデータ型の変換の VARCHAR サポート”
([SAS Cloud Analytic Services: ユーザーガイド](#))および“暗黙的な型変
換を使用した VARCHAR の長さ” ([SAS Cloud Analytic Services: ユー
ザーガイド](#))を参照してください。

*

許可される最大長:の 536,870,911 文字をサポートするように指定しま
す。

DEFAULT=*n*

新しく作成する数値変数の値を保存するときに使用するデフォルトのバイト数
を変更します。

デフォルト 8 バイト

範囲 3 から 8 バイト

詳細

一般情報

通常、変数の長さはこの情報によって決定されます。

- 変数が数値か文字のいずれであるか
- 変数の作成方法
- LENGTH ステートメントまたは ATTRIB ステートメントの有無
- プログラムの処理に使用される SAS セッションエンコーディング

長さを割り当てるルールに基づいて、LENGTH ステートメントで割り当てた文字の
長さと数値の長さを ATTRIB ステートメントで変更できます。またその逆も実行で
きます。

VARCHAR 変数は、その値を保持するために必要なメモリを割り当てます。
VARCHAR 変数が使用するメモリの最大量は、SAS セッションエンコーディングによ
り異なります。VARCHAR(100)と宣言された変数は、100 文字まで格納できます。
LATIN1 のようなシングルバイトの SAS セッションエンコーディングでは、変数は
その値に最大 100 バイトを使用します。

UTF-8 の SAS セッションエンコーディングでは、1 つの文字が最大 4 バイトになる
ため、VARCHAR(100)として宣言された変数によって最大 400 バイトが使用される
可能性があります。

長さの変数への代入については、“[変数の変更](#)” ([SAS プログラマガイド: エッセンシャル](#)
[/](#))を参照してください。

比較

ATTRIB ステートメントを使用すると、文字変数と数値変数うの他の属性と同時に長さを割り当てることができます。

例

この例では、LENGTH ステートメントを使用して文字変数 NAME の長さを 25 バイトに設定します。また、この LENGTH ステートメントは、新しく作成する数値変数を保存するときに使用するデフォルトのバイト数を 8 バイトから 4 バイトに変更します。TRIM 関数は、LASTNAME の末尾にあるブランクを削除してから、結合します。

- カンマ(,)
- ブランク
- FIRSTNAME の値

LENGTH ステートメントを省略すると、変数 NAME の長さは 32 バイトに設定されます。

```
data testlength;
  informat FirstName LastName $15. n1 6.2;
  input firstname lastname n1 n2;
  length name $25 default=4;
  name=trim(lastname)||','||firstname;
  datalines;
Alexander Robinson 35 11
;
proc contents data=testlength;
run;
proc print data=testlength;
run;
```

出力結果は、PROC CONTENTS の実行結果の一部と PROC PRINT で生成したレポートを示しています。

アウトプット 2.22 TESTLENGTH に対する PROC CONTENTS の実行結果の一部

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Informat
1	FirstName	Char	15	\$15.
2	LastName	Char	15	\$15.
3	n1	Num	4	6.2
4	n2	Num	4	
5	name	Char	25	

アウトプット 2.23 変数の長さの設定

Obs	FirstName	LastName	n1	n2	name
1	Alexander	Robinson	0.35000	11	Robinson, Alexander

関連項目

- PROC ステップで LENGTH ステートメントを使用する方法の詳細については、[Base SAS プロシジャガイド](#)を参照してください。
- “名前の長さの規則” ([SAS プログラマガイド: エッセンシャル](#))

ステートメント:

- “ATTRIB ステートメント” (52 ページ)

LIBNAME ステートメント

SAS ライブラリへのライブラリ参照名(ショートカット名)の関連付けや関連付けの取り消し、1 つまたはすべてのライブラリ参照名の関連付けの取り消しを実行します。また、SAS ライブラリの属性のリスト、SAS ライブラリの連結や SAS カタログの連結を実行します。

注: [LIBNAME ステートメント](#)は SAS グローバルステートメントに移動しました。

LIBNAME ステートメント: CVP エンジン

ファイルのトランスコードが必要なときに文字データの切り捨てが発生しなくなるように、文字変数の長さを拡張する文字変数埋め込み(CVP)エンジンのライブラリ参照名を関連付けます。

注: [LIBNAME ステートメント: CVP エンジン](#)は SAS グローバルステートメントに移動しました。

LIBNAME ステートメント: JMP エンジン

ライブラリ参照名を JMP データテーブルに関連付け、JMP データテーブルの読み込みや書き込みを実行できるようにします。

注: [LIBNAME ステートメント: JMP エンジン](#)は SAS グローバルステートメントに移動しました。

LIBNAME ステートメント:JSON エンジン

SAS ライブラリ参照名を JSON ドキュメントと関連付けて、JSON データと、指定された任意の MAP が有効になることを保証します。

注: [LIBNAME ステートメント:JSON エンジン](#)は SAS グローバルステートメントに移動しました。

LIBNAME ステートメント: WebDAV サーバーアクセス

SAS ライブラリにライブラリ参照名を関連付け、WebDAV (Web-based Distributed Authoring And Versioning)サーバーにアクセスできるようにします。

注: [LIBNAME ステートメント: WebDAV サーバーアクセス](#)は SAS グローバルステートメントに移動しました。

LINK ステートメント

指定したステートメントラベルにプログラム実行移動します。RETURN ステートメントが後に続く場合、LINK ステートメントの直後のステートメントに実行を戻します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

LINK *label*;

引数

label

LINK の出力先となるステートメントラベルを指定します。*label* 引数は必ず指定する必要があります。

詳細

LINK ステートメントは、LINK ステートメントに指定したステートメントラベルに SAS を移動します。移動先から RETURN ステートメントまでの間にあるステートメントの実行を続けます。RETURN ステートメントは、LINK ステートメントの後に続くステートメントにプログラム制御を移動します。

LINK ステートメントと出力先は同じ DATA ステップ内になければなりません。LINK ステートメントでは、出力先はステートメントラベルで指定します。

LINK ステートメントでは、別の LINK ステートメントを含むステートメントグループに移動することができます。この配置方法はネストと呼ばれます。無限ループが発生しないように、ネストする LINK ステートメントのデフォルト数が SAS によって設定されています。RETURN ステートメントを間にはさまずに、最大 10 個までの LINK ステートメントをネストできます。複数の LINK ステートメントが実行された場合、RETURN ステートメントは最後に実行した LINK ステートメントの次のステートメントに SAS を戻します。ネストする LINK ステートメントの数を増やすには、DATA ステートメントで/STACK オプションを使用します。

比較

LINK ステートメントと GOTO ステートメントでは、後に続く RETURN ステートメントのアクションが異なります。LINK ステートメントの後に RETURN ステートメントを指定すると、LINK ステートメントの直後のステートメントに実行を戻します。GOTO ステートメントの後に RETURN ステートメントを指定し、GOTO ステートメントの前に LINK ステートメントがない場合は、DATA ステップの先頭に実行を戻します。このとき、GOTO ステートメントの前に LINK ステートメントがある場合は、LINK ステートメントの直後の最初のステートメントから実行を続けます。通常、LINK ステートメントは RETURN ステートメントを明示的に指定して使用しますが、GOTO ステートメントは RETURN ステートメントを指定せずに使用します。

同じプログラム内の複数のポイントでステートメントグループを実行する場合、LINK ステートメントを使用するとコーディングが簡単になり、プログラムロジックも理解しやすくなります。プログラム内でステートメントグループを 1 つのポイントでのみ実行する場合、LINK-RETURN ロジックよりも DO グループロジックを使用するほうが簡単です。

例: プログラム実行の移動

この例では、変数 TYPE の値が aluv になる場合、LINK ステートメントはラベル CALCU に割り当てたステートメントにプログラム実行を移動します。RETURN ステートメントが検出されるまでこのプログラムが実行されます。RETURN ステートメントが検出されると、プログラム実行は LINK の後に続く最初のステートメントに移動されます。割り当てステートメントを実行し、オブザベーションを書き込みます。その後、DATA ステップの先頭に戻って、次のレコードを読み込みます。TYPE の値が aluv ではない場合、割り当てステートメントを実行し、オブザベーションを書き込みます。その後、DATA ステップの先頭に戻ります。

```
data hydro;
```

```

input type $ depth station $;
  /* link to label calcu: */
if type ='aluv' then link calcu;
date=today();
  /* return to top of step */
return;
calcu: if station='site_1'
      then elevatn=6650-depth;
else if station='site_2'
      then elevatn=5500-depth;
      /* return to date=today(); */
return;
datalines;
aluv 523 site_1
uppa 234 site_2
aluv 666 site_2
...more data lines...
;

```

関連項目

ステートメント:

- [“DATA ステートメント” \(68 ページ\)](#)
- [“DO ステートメント” \(88 ページ\)](#)
- [“GOTO ステートメント” \(135 ページ\)](#)
- [“Label: ステートメント” \(234 ページ\)](#)
- [“RETURN ステートメント” \(350 ページ\)](#)

LIST ステートメント

現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。

該当要素: DATA ステップ

カテゴリ: アクション

CAS

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

LIST;

引数なし

LIST ステートメントを使用すると、現在処理中のオブザベーションの入力データレコードを SAS ログに書き込みます。

詳細

LIST ステートメントは、INPUT ステートメントで読み込まれるデータのみを処理します。SET、MERGE、MODIFY、UPDATE の各ステートメントで読み込まれるデータは処理されません。

SAS ログでは、列の位置を示すルーラーが表示してから、レコードを表示します。

可変長レコード(RECFM=V)の場合、入力行の最後にレコード長を書き込みます。固定長レコード(RECFM=F)の長さは、読み込むデータ量がレコード長(LRECL)と一致している場合には書き込みません。

比較

この表で、LIST ステートメントと PUT ステートメントを比較します。

アクション	LIST ステートメント	PUT ステートメント
書き込みのタイミング	DATA ステップの繰り返しの最後に実行	すぐに実行
書き込み内容	入力した内容と同一の入力データレコード	指定した変数またはリテラル
書き込み先	SAS ログのみ	SAS ログ、SAS 出力先、外部ファイル
処理可能なステートメント	INPUT ステートメントのみ	すべてのデータ読み込みステートメント
16 進数の値の処理	書き込みできない文字が検出された場合は自動的に値を 16 進数で出力する	16 進数の出力形式が指定された場合にのみ文字を 16 進数で表示する

例

例 1: 欠損データを含むレコードをリストする

この例では、LIST ステートメントを使用して、欠損データが含まれる入力レコードを SAS ログに書き込みます。INPUT ステートメントの行ポインターコントロールが #3 と指定されているため、1 つのオブザベーションを作成するのに 3 つのレコードを読み込みます。そのため、W2AMT の値が欠損しているたびに、現在の入力レコードが 3 行にわたって SAS ログに書き込まれます。

```
data employee;
  input ssn 1-9 #3 w2amt 1-6;
  if w2amt=. then list;
  datalines;
23456789
JAMES SMITH
356.79
345671234
Jeffrey Thomas
.
;
```

アウトプット 2.24 欠損データを表示するログ

```
RULE:-----1-----2-----3-----4-----5-----
9      345671234
10     Jeffrey Thomas
11     .
```

番号 9、10、と 11 は SAS ログ内の行番号を示しています。

例 2: 可変長レコードのレコード長を表示する

この例では、可変長 ID 番号を含むの外部ファイルを入力として使用します。RECFM=V オプションは INFILE ステートメントで指定され、LIST ステートメントは SAS ログにレコードを書き出します。この例の RECFM=V オプションが示すように、ファイルが可変長のレコードを持つ場合、SAS ログにリストされる各レコードの最後にレコード長が書き込まれます。

```
data employee;
  infile 'your-external-file' recfm=v;
  input id $;
  list;
run;
```

アウトプット 2.25 可変長レコードとレコード長を表示するログ

```
RULE:-----1-----2-----3-----4-----5-----
1      23456789 8
2      123456789 9
3      555555555 10
4      345671234 9
5      2345678910 10
6      2345678 7
```


例 3: 入力データの 16 進数値のリスト

/HEXLISTALL DATA ステートメントオプションにより、LIST ステートメントで入力データの全行を 16 進数形式で SAS ログに書き込むことができます。この例では、有効なデータ行、印刷不可能な文字を含むデータ行、および無効なデータ行を含むファイルが作成されます。LIST ステートメントを伴うデフォルトの DATA ステートメントは、印刷不可能な文字の入力データを 16 進形式で SAS ログに表示するために指定されます。次に、/HEXLISTALL 引数を使用して DATA ステートメントを指定します。LIST ステートメントは、すべての入力データを 16 進形式で SAS ログに表示します。

ここでは、f_out.txt というファイルが作成されます。ファイルには 3 行分のデータが入っています。

```
filename f_out 'f_out.txt' recfm=f lrecl=30; /* assign fileref */
data;
  file f_out;
  output;
  put '123456789012345678901234567890';
  put '0102030405060708090A0B0C0D0E0F10111213141516171819202A2B2C2D'x;
  put 'Not a number          ';
run;
```

DATA ステートメントのデフォルト設定は、LIST ステートメントで使用され、ファイルの内容を SAS ログに書き込みます。デフォルトでは、入力ファイルの 2 行目にある印刷不可能な文字を 16 進数形式で表示します。2 行目と 3 行目にある無効なデータには、注が表示されます。

```
data _null_;
  infile f_out length=len;
  input num;
  list;
run;
```

アウトプット 2.26 印刷不可能な文字の 16 進数値の SAS ログ

```
RULE:      +---+---1---+---2---+---3---+---4---+---5---+---6---
+---7---+---8---+---
1          123456789012345678901234567890
NOTE: Invalid data for num in line 2 1-25.

RULE:      +---+---1---+---2---+---3---+---4---+---5---+---6---
+---7---+---8---+---

2  CHAR ..... *+, -
   ZONE 0000000000000001111111112222
   NUMR 123456789ABCDEF01234567890ABCD
len=30 num=. _ERROR_=1 _N_=2
NOTE: Invalid data for num in line 3 1-3.
3      Not a number
len=30 num=. _ERROR_=1 _N_=3
```

DATA ステートメントに/HEXLISTALL 引数が追加されます。LIST ステートメントは、入力データのすべての行の 16 進数形式を SAS ログに書き込むようになりました。2 行目と 3 行目にある無効なデータには、注が表示されます。

```
data _null_ /HEXLISTALL;
  infile f_out length=len;
```

```
input num;
list;
run;

filename f_out; /* deassign fileref */
```

アウトプット 2.27 全入力データの 16 進数値の SAS ログ

```
RULE:      +---+---+1---+---+2---+---+3---+---+4---+---+5---+---+6---+
+---+7---+---+8---+---+

1  CHAR  123456789012345678901234567890
   ZONE  33333333333333333333333333333333
   NUMR  123456789012345678901234567890
NOTE: Invalid data for num in line 2 1-25.

2  CHAR  ..... *+,-
   ZONE  0000000000000000111111111122222
   NUMR  123456789ABCDEF01234567890ABCD
len=30 num=._ERROR_=1 _N_=2
NOTE: Invalid data for num in line 3 1-3.

3  CHAR  Not a number
   ZONE  4672626766672222222222222222222
   NUMR  EF4010E5D2520000000000000000000
len=30 num=._ERROR_=1 _N_=3
```

関連項目

ステートメント:

- [“DATA ステートメント” \(68 ページ\)](#)
- [“INPUT ステートメント” \(186 ページ\)](#)
- [“PUT ステートメント” \(289 ページ\)](#)

%LIST ステートメント

現在のセッション中に入力された行を表示します。

注: [%LIST ステートメント](#)は SAS グローバルステートメントに移動しました。

LOCK ステートメント

既存の SAS ファイルに対する排他的ロックの取得、リスト出力、または解除を行います。

注: [LOCK ステートメント](#)は SAS グローバルステートメントに移動しました。

LOCKDOWN ステートメント

サーバープロセスの内部からホスト動作環境へのアクセスを制限することにより、SAS Viya プラットフォーム上の SAS/CONNECT サーバーと SAS Compute Server を保護します。

注: SAS Viya プラットフォームの LOCKDOWN ステートメントは、SAS Viya Administration: Programming Run-Time Servers に移動しました。参照: [“References” \(SAS Viya: Programming Run-Time Servers\)](#)。
LOCKDOWN については、[SAS LOCKDOWN ステートメント](#)を参照してください。

LOSTCARD ステートメント

1 オブザベーションあたりのレコードが複数あるデータに欠損レコードや無効なレコードが検出された際に、入力データを再同期します。

該当要素: DATA ステップ

カテゴリ: アクション
CAS

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

LOSTCARD;

引数なし

LOSTCARD ステートメントを使用すると、現在のグループに欠損レコードが含まれている場合、次のグループからレコードを読み込まないようにします。

詳細

LOSTCARD を使用する必要がある場合

複数のレコードを読み込んで 1 つのオブザベーションを作成する場合、データの最後に到達するまでレコードの欠損は検出されません。データ内に欠損レコードが存在すると、SAS データセットの後続のオブザベーションの値に誤りが生じる可能性

があります。LOSTCARD ステートメントを使用すると、現在のグループのレコード数が予測よりも少ない場合、次のグループからデータを読み込まないようにします。

LOSTCARD ステートメントは、1 オブザベーションあたりの入力データのレコード数が固定されている場合や、オブザベーションの各レコードに同じ値の ID 変数が使用されている場合に便利です。通常、LOSTCARD ステートメントは、IF-THEN ステートメントの THEN 句や SELECT グループのステートメントなどの条件付き処理の中で使用されます。

LOSTCARD ステートメントの実行

LOSTCARD ステートメントを実行すると、次に示す複数のステップが実行されます。

- 1 LOSTCARD メッセージ、ルーラー、現在のオブザベーションを作成しようとして読み込んだすべてのレコードの 3 つが SAS ログに書き込まれます。
- 2 読み込み中のレコードグループにある最初のレコードを破棄し、オブザベーションには書き込まずに、DATA ステップの先頭に処理を戻します。
- 3 自動変数_N_の値に 1 を追加しません。(通常は、DATA ステップの先頭に戻ったときに自動変数_N_の値を 1 ずつ増やしていきます。)
- 4 グループ内の 2 番目のレコードから読み込みを開始して、オブザベーションを作成します。INPUT ステートメントに指定されている数のレコードを読み込みます。
- 5 LOSTCARD の IF 条件が真である場合、ステップ 1 から 4 を繰り返します。ログを見やすくするため、指定したレコードグループに対してメッセージとルーラーが出力されるのは 1 度だけです。また、レコードが、次のオブザベーションの作成で使用される場合でも 1 度しか出力されません。
- 6 LOSTCARD の IF 条件が真でない場合、オブザベーションが作成され、SAS データセットに書き込まれます。

例: 入力データの再同期

この例では、条件付きの作成に LOSTCARD ステートメントを使用して、欠損データの特定と入力データの再同期を実行します。

```
data inspect;
  input id 1-3 age 8-9 #2 id2 1-3 loc
        #3 id3 1-3 wt;
  if id ne id2 or id ne id3 then
  do;
    put 'DATA RECORD ERROR: ' id= id2= id3=;
    lostcard;
  end;
  datalines;
301 32
301 61432
301 127
302 61
302 83171
400 46
```

```

409 23145
400 197
411 53
411 99551
411 139
;

```

この DATA ステップでは、オブザベーションを書き込む前に入力レコードを 3 つ読み込みます。レコード 1(変数 ID)の ID 番号が 2 番目のレコード(ID2)または 3 番目のレコード(ID3)と一致しない場合、レコードが正しく入力されていないか欠損しています。IF-THEN DO ステートメントでは、ID 番号が無効な場合に PUT ステートメントに指定したメッセージを出力し、LOSTCARD ステートメントを実行するように指定しています。

この例では、2 番目のオブザベーション(ID3=400)の 3 番目のレコードが欠損しています。3 番目のオブザベーションの 2 番目のレコードが誤って入力されています(ID=400 となるのが正しいですが、ID2=409 となっています)。そのため、データセットには ID301 と ID411 の 2 つのオブザベーションが含まれます。ID=302 または ID=400 のオブザベーションはデータセットに含まれません。この DATA ステップを実行すると、PUT ステートメントと LOSTCARD ステートメントにより、次のステートメントがログに書き込まれます。

```

DATA RECORD ERROR: id=302 id2=302 id3=400
NOTE: LOST CARD.
RULE:----+----1----+----2----+----3----+----4----+----5----+----
14 302 61
15 302 83171
16 400 46
DATA RECORD ERROR: id=302 id2=400 id3=409
NOTE: LOST CARD.
17 409 23145
DATA RECORD ERROR: id=400 id2=409 id3=400
NOTE: LOST CARD.
18 400 197
DATA RECORD ERROR: id=409 id2=400 id3=411
NOTE: LOST CARD.
19 411 53
DATA RECORD ERROR: id=400 id2=411 id3=411
NOTE: LOST CARD.
20 411 99551

```

番号 14、15、16、17、18、19、20 は SAS ログ内の行番号を示しています。

関連項目

ステートメント:

- [“IF-THEN/ELSE ステートメント” \(140 ページ\)](#)

MERGE ステートメント

複数の SAS データセットにある複数のオブザベーションを 1 つのオブザベーションに結合します。

該当要素:	DATA ステップ
カテゴリ:	CAS ファイル操作
種類:	実行
制限事項:	このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“ DATA Step Acceleration ” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。
注:	MERGE ステートメントを使用して読み込まれた変数は PDV で保持されます。読み込まれた変数のデータタイプも保持されます。詳細については、“ RETAIN ステートメント ” (344 ページ)を参照してください。
参照項目:	“MODIFY ステートメントと KEY=オプションを使用したデータの更新” (SAS プログラマガイド: エッセンシャル) “1 対 1 マージ” (SAS プログラマガイド: エッセンシャル) “マッチマージ” (SAS プログラマガイド: エッセンシャル) “1 対 1 読み取りと 1 対 1 マージ” (SAS プログラマガイド: エッセンシャル) “データセットを結合するための SAS 言語要素の比較” (SAS プログラマガイド: エッセンシャル) “DATA ステップでの BY グループの処理” (SAS プログラマガイド: エッセンシャル)
例:	“データの結合” (SAS プログラマガイド: エッセンシャル) “データの結合” (SAS プログラマガイド: エッセンシャル)

構文

```
MERGE SAS-data-set-1 <(data-set-options)>
      SAS-data-set-2 <(data-set-options) >
      <...SAS-data-set-n <(data-set-options)>>
      <END=variable>;
```

引数

SAS-data-set

オブザベーションを読み込む既存の SAS データセットを少なくとも 2 つ指定します。データセットは個別に指定するか、データセットのリストで指定します。また、両方を組み合わせても指定できます。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することが

できます。物理パス名は単一引用符または二重引用符で囲む必要があります。

追加する SAS データセットを指定できます。

参照 “MERGE ステートメントでデータセットリストを使用する” (254 ページ)
項目:

(data-set-options)

SAS データセット名の後ろに、1 つまたは複数の SAS データセットオプションを丸かっこで囲んで指定します。

注 データセットオプションは、処理対象のオブザベーションを DATA ステップに読み込むときに実行するアクションを指定します。データセットオプションのリストについては、[SAS データセットオプション: リファレンス](#)を参照してください。

ヒント データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

END=variable

作成する一時変数の名前を指定します。この変数の値にはファイル終端指示子が格納されます。

注 この変数は 0 に初期化されますが、MERGE ステートメントが最後のオブザベーションを処理するときに 1 に設定されます。各入力データセットに存在するオブザベーションの数が異なる場合は、MERGE ステートメントによってすべてのデータセットの最後のオブザベーションが処理されるときに、END=変数が 1 に設定されます。

ヒント END=変数は、作成される SAS データセットには追加されません。

詳細

概要

MERGE ステートメントには柔軟性があり、SAS プログラミングでさまざまな方法で使用されます。次のセクションでは、MERGE ステートメントの基本的な使用方法を説明します。応用として、2 つ以上の BY 変数の使用、3 つ以上のデータセットのマージ、少数のオブザベーションを別のデータセットの全オブザベーションへのマージなどがあります。

重要 match-merge の動作を完全に理解するには、次のドキュメントをお読みください。

- “マッチマージ” (SAS プログラマガイド: エッセンシャル)
- “1 対 1 マージ” (SAS プログラマガイド: エッセンシャル)

- “1 対 1 読み取りと 1 対 1 マージ” (SAS プログラマガイド: エッセンシャル)
 - “データセットを結合するための SAS 言語要素の比較” (SAS プログラマガイド: エッセンシャル)
 - “データの結合” (SAS プログラマガイド: エッセンシャル)
 - “データの結合” (SAS プログラマガイド: エッセンシャル)
- “DATA ステップでの BY グループの処理” (SAS プログラマガイド: エッセンシャル)

MERGE ステートメントでデータセットリストを使用する

MERGE ステートメントでは、データセットのリストを使用できます。データセットリストを使用すると、現在存在するデータセットのグループを簡単に示せます。このデータセットリストには、名前接頭辞リストまたは番号付き範囲リストを指定する必要があります。

名前接頭辞リストは、指定した文字列で始まるすべてのデータセットを示します。たとえば、merge SALES1;と指定すると、“SALES1”で始まる SALES1、SALES10、SALES11、SALES12 などのデータセットがすべてマージされます。

番号付き範囲リストでは、連番である最後の文字を除き、同じ名前のデータセットが存在する必要があります。番号付き範囲リストでは、開始値と終了値に任意の数値を使用してもかまいません。たとえば、次のリストは同じデータセットを示します。

```
sales1 sales2 sales3 sales4
sales1-sales4
```

注: 最初のデータセット名にある数値接尾辞の先頭に 0 が置かれる場合、最後のデータセット名の数値接尾辞の桁数は最初のデータセット名の桁数と一致するか、または上回る必要があります。このように指定されていない場合はエラーが発生します。たとえば、データセットリストを sales001-sales99 または sales01-sales9 と指定するとエラーが発生します。ただし、このデータセットリストの有効な指定は、sales001-sales999 です。最初のデータセット名にある数値接尾辞の先頭に 0 が置かれない場合、最初と最後のデータセット名に使用する数値接尾辞の桁数を一致させる必要はありません。たとえば、データセットリストの指定として sales1-sales999 は有効です。

番号付きデータセットリストを使用する場合に考慮すべきその他のルールを次に示します。

- 範囲のグループを複数指定できます。
merge cost1-cost4 cost11-cost14 cost21-cost24;
- 番号付き範囲リストと名前接頭辞リストを組み合わせで指定できます。
merge cost1-cost4 cost2: cost33-37;
- 個々のデータセットとデータセットリストを組み合わせで指定できます。
merge cost1 cost10-cost20 cost30;
- データセットリストを囲んだ引用符は無視されます。


```
/* these two lines are the same */
merge sales1-sales4;
merge 'sales1'n-'sales4'n;
```

- データセット名のブランクは無効です。引用符を使用する場合、末尾にあるブランクは無視されます。

```
/* blanks in these statements will cause errors */
merge sales 1-sales 4;
merge 'sales 1'n - 'sales 4'n;
/* trailing blanks in this statement will be ignored */
merge 'sales1'n - 'sales4'n;
```

- 数値接尾辞に使用できる最大数は、2147483647 です。

```
/* this suffix will cause an error */
merge prod2000000000-prod2934850239;
```

- 物理パス名を使用できません。

```
/* physical pathnames will cause an error */
%let work_path = %sysfunc(pathname(WORK));
merge "&work_path\dept.sas7bdat"-"&work_path\emp.sas7bdat";
```

1 対 1 のマージ

1 対 1 のマージとは、複数の SAS データセットからオブザベーションを読み込んで結合し、新しいデータセットの単一のオブザベーションにする処理です。1 対 1 のマージを実行するには、BY ステートメントを指定せずに MERGE ステートメントを使用します。MERGE ステートメントに指定したすべてのデータセットの最初のオブザベーションを結合してから、新しいデータセットに最初のオブザベーションを作成します。次に、すべてのデータセットの 2 番目のオブザベーションを結合して、新しいデータセットに 2 番目のオブザベーションを作成します。同様の作業が繰り返されます。1 対 1 マージでは、新しいデータセットに作成されるオブザベーションの数は、MERGE ステートメントに指定した最大データセットのオブザベーションの数と一致します。1 対 1 マージについては、例 1 を参照してください。

詳細については、“[1 対 1 マージ](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

注意

1 対 1 マージを使用してデータセットを結合する場合は注意が必要です。 1 対 1 マージでは、予期しない結果が発生する場合があります。この方法を使用する前に、データセットのサンプルを使用してプログラムのテストを実行してください。

マッチマージ

マッチマージでは、複数の SAS データセットから、共通する変数の値に基づいてオブザベーションを取り出し、結合して、新しいデータセットの 1 つのオブザベーションにします。新しいデータセットに含まれるオブザベーションの数は、すべてのデータセットの各 BY グループに含まれるオブザベーションのうち、最も数が多いオブザベーションの合計になります。マッチマージを実行するには、MERGE ステートメントの直後に BY ステートメントを使用する必要があります。BY ステートメントに指定する変数は、すべてのデータセットに共通している必要があります。DATA ステップでは、1 つの BY ステートメントのみ、MERGE ステートメントに指定できます。MERGE ステートメントにリストされるデータセットは、BY ステートメントに

リストされる変数の値に基づいて並べ替えられているか、適切なインデックスが含まれている必要があります。

詳細については、“[マッチマージ](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

注: MERGE ステートメントは、1 対 1 のマッチマージではデカルト積を生成しません。そのかわり、MERGE ステートメントは、少なくとも 1 つのデータセット内の BY グループにオブザベーションが存在する間、1 対 1 のマッチマージを実施します。1 つのデータセット内にある BY グループのすべてのオブザベーションを読み込んだ後、別のデータセット内にまだオブザベーションが存在する場合、特定 BY グループに関してすべてのオブザベーションが読み込まれるまで、1 対 1 のマッチマージが実行されます。

比較

- MERGE ステートメントでは、複数のデータセットにあるオブザベーションが結合されます。UPDATE ステートメントでは、2 つのデータセットにあるオブザベーションが結合されます。UPDATE ステートメントでは、マスターデータセットにある選択したオブザベーションの値も変更または更新されます。また、UPDATE ステートメントによりオブザベーションが追加される場合があります。
- UPDATE ステートメントと同様に、MODIFY ステートメントでも、2 つの SAS データセットにあるオブザベーションが結合され、マスターデータセットにある選択したオブザベーションの値も変更または更新されます。
- 2 つ以上の SET ステートメントを使用してオブザベーションを読み込んだ結果は、BY ステートメントを指定せずに MERGE ステートメントを使用してオブザベーションを読み込んだ結果と似ています。ただし、SET ステートメントを使用すると、すべてのデータセットからオブザベーションをすべて読み込む前に、オブザベーションの数が同一ではない場合は処理が中止されます。これに対して、MERGE ステートメントに指定した全データセットの全オブザベーションの処理は続けられます。

詳細については、“[データセットを結合するための SAS 言語要素の比較](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

例

例 1: 1 対 1 のマージ

次の例では、2 つのデータセットにあるオブザベーションを結合し、新しいデータセットに 1 つのオブザベーションを作成する方法を示しています。

```
data benefits.qtr1;
  merge benefits.jan benefits.feb;
run;
```

例 2: マッチマージ

次の例では、2 つのデータセットにあるオブザベーションを BY ステートメントに指定した変数の値に基づいて結合し、新しいデータセットに 1 つのオブザベーションを作成する方法を示しています。

```
data inventory;
  merge stock orders;
  by partnum;
run;
```

例 3: データセットリストを使用したマージ

この例では、データリストを使用してマージ対象のデータセットを定義します。

```
data d008; job=3; emp=19; run;
data d009; job=3; sal=50; run;
data d010; job=4; emp=97; run;
data d011; job=4; sal=15; run;
data comb;
  merge d008-d011;
  by job;
run;
proc print data=comb;
run;
```

例 4: BY 値と IN=データセットオプションを使用した 3 つのテーブルのマージ

```
DATA CAFE(KEEP=NAME PLACE CNUM);
  INPUT NAME $ ;
  PLACE = 'CAFE ' ;
  CNUM = 'C' || LEFT(PUT(_N_,2.));
  DATALINES;
ANDERSON
COOPER
DIXON
FREDERIC
FREDERIC
PALMER
RANDALL
RANDALL
SMITH
SMITH
SMITH
;
RUN;

DATA VENDING (KEEP=NAME PLACE VNUM);
  INPUT NAME $ ;
  PLACE = 'VENDING ' ;
  VNUM = 'V' || LEFT(PUT(_N_,2.));
  DATALINES;
CARTER
DANIELS
GARY
```

```
GARY  
HODGE  
PALMER  
RANDALL  
RANDALL  
SMITH  
SMITH  
SPENCER  
SPENCER  
SPENCER  
SPENCER  
;  
RUN;
```

```
DATA SNACK (KEEP=NAME PLACE SNUM);  
  INPUT NAME $ ;  
  PLACE = 'SNACK ' ;  
  SNUM = 'S' || LEFT(PUT(_N_,2.));  
  DATALINES;  
BARRETT  
COOPER  
DANIELS  
DIXON  
DIXON  
FREDERIC  
GARY  
HODGE  
HODGE  
PALMER  
RANDALL  
RANDALL  
SMITH  
SMITH  
SMITH  
SMITH  
SPENCER  
SPENCER  
;  
RUN;
```

```
DATA ALL;  
  MERGE CAFE(IN=CAFEIN) SNACK(IN=SNACKIN) VENDING(IN=VENDIN);  
  BY NAME;  
  CIN=CAFEIN; SIN=SNACKIN; VIN=VENDIN;  
RUN;
```

```
PROC PRINT;  
  TITLE 'MERGED DATA';  
RUN;
```

MERGED DATA								
Obs	NAME	PLACE	CNUM	SNUM	VNUM	CIN	SIN	VIN
1	ANDERSON	CAFE	C1			1	0	0
2	BARRETT	SNACK		S1		0	1	0
3	CARTER	VENDING			V1	0	0	1
4	COOPER	SNACK	C2	S2		1	1	0
5	DANIELS	VENDING		S3	V2	0	1	1
6	DIXON	SNACK	C3	S4		1	1	0
7	DIXON	SNACK	C3	S5		1	1	0
8	FREDERIC	SNACK	C4	S6		1	1	0
9	FREDERIC	CAFE	C5	S6		1	1	0
10	GARY	VENDING		S7	V3	0	1	1
11	GARY	VENDING		S7	V4	0	1	1
12	HODGE	VENDING		S8	V5	0	1	1
13	HODGE	SNACK		S9	V5	0	1	1
14	PALMER	VENDING	C6	S10	V6	1	1	1
15	ANDALL	VENDING	C7	S11			1	1

例 5: BY 値と IN=データセットオプションを使用した 2 つのテーブルのマージ

```
data have_a;
  input ID amount_a;
  datalines;
1 10
3 15
4 20
7 15
9 12
10 14
;
```

```
data have_b;
  input ID amount_b;
  datalines;
2 15
3 20
4 10
5 12
7 20
8 15
9 10
11 20
```

```

;

data want;
  merge have_a(in=inA) have_b(in=inb);
  by id;
  length joinType $ 2;
  joinType = cats(inA, inB);
run;

proc print data=want;
run;
quit;

```

MERGED DATA				
Obs	ID	amount_a	amount_b	joinType
1	1	10	.	10
2	2	.	15	01
3	3	15	20	11
4	4	20	10	11
5	5	.	12	01
6	7	15	20	11
7	8	.	15	01
8	9	12	10	11
9	10	14	.	10
10	11	.	20	01

関連項目

- “データの結合” (SAS プログラマガイド: エッセンシャル)

ステートメント:

- “BY ステートメント” (57 ページ)
- “MODIFY ステートメント” (261 ページ)
- “SET ステートメント” (356 ページ)
- “UPDATE ステートメント” (380 ページ)

MISSING ステートメント

入力データ内で数値データの特異欠損値を表す文字を割り当てます。

注: [MISSING ステートメント](#)は SAS グローバルステートメントに移動しました。

MODIFY ステートメント

既存のデータセットにあるオブザベーションの置き換え、削除、追加を実行します。ただし、オブザベーションのコピーは作成しません。

該当要素: DATA ステップ

カテゴリ: ファイル操作

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。

このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

このステートメントでは、変数の追加など、SAS データセットのディスクリプター部分を変更することはできません。

注: パスワードで保護されたデータセットを変更する場合、DATA ステートメントではなく MODIFY ステートメントの適切なデータセットオプション(ALTER=または PW=)にパスワードを指定します。

MODIFY ステートメントを使用して読み込まれた変数は PDV で保持されます。詳細については、[“RETAIN ステートメント” \(344 ページ\)](#)を参照してください。

参照項目: [“変更” \(SAS プログラマガイド: エッセンシャル\)](#)

注意: **MODIFY ステートメントを含む DATA ステップの実行中にシステムが異常終了すると、SAS データセットが破損する可能性があります。** ネイティブ SAS データファイルのオブザベーションに誤ったデータ値が含まれたり、データファイルが読み込めなくなる場合があります。ビューで参照する DBMS テーブルには影響はありません。

構文

形式 1: **MODIFY** *master-data-set* <(data-set-options)> *transaction-data-set* <(data-set-options)>
 <CUROBS=*variable*> <NOBS=*variable*> <END=*variable*>
 <UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;
BY *by-variable*;

- 形式 2: **MODIFY** *master-data-set* <(data-set-options)> **KEY=index** </ UNIQUE>
 <KEYRESET=variable> <NOBS=variable> <END=variable>;
- 形式 3: **MODIFY** *master-data-set* <(data-set-options)> <NOBS=variable> **POINT=variable**;
- 形式 4: **MODIFY** *master-data-set* <(data-set-options)> <NOBS=variable> <END=variable>;

引数

master-data-set

変更対象となる SAS データセット(マスターデータセット)を指定します。

制限事項 このデータセットは、DATA ステートメントにも指定する必要があります。

順次アクセスまたはマッチングアクセスの場合、マスターデータセットには、SAS データファイル、SAS/ACCESS ビュー、SQL ビュー、LIBNAME ステートメントの DBMS エンジン指定できます。DATA ステップビューと SQL プロシジャのパススルー機能を使用したビューは指定できません。

POINT=オプションを指定したランダムアクセスの場合、マスターデータセットには SAS データファイルか、SAS データファイルを参照する SQL ビューを指定する必要があります。

KEY=オプションを指定したダイレクトアクセスの場合、マスターデータセットには SAS データファイルか、LIBNAME ステートメントの DBMS エンジン指定できます。SAS ファイルを指定する場合、インデックス付きである必要があります。KEY=オプションにインデックス名を指定する必要があります。

DBMS の場合、KEY=オプションにキーワード DBKEY を設定し、インデックスとして使用する列名を DBKEY=データセットオプションに指定する必要があります。指定した列名は、DBMS に渡す WHERE 式の生成に使用されません。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

(data-set-options)

SAS データセット名の後ろに、1 つまたは複数の SAS データセットオプションを丸かっこで囲んで指定します。

注 データセットオプションは、処理対象のオブザベーションを DATA ステップに読み込むときに実行するアクションを指定します。データセットオプションのリストについては、[SAS データセットオプション: リファレンス](#)を参照してください。

ヒント データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

transaction-data-set

マッチングアクセスで使用する値を含む SAS データセットを指定します。この値はマスターデータセットの更新に使用されます。

制限事項 このデータセットは、DATA ステップに BY ステートメントが含まれている場合にのみ指定してください。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

by-variable

オブザベーションの識別に使用する、1 つまたは複数の変数名を指定します。

CUROBS=variable

データセットから読み込んだオブザベーション数を含む変数を作成および名前を付与

END=variable

作成する一時変数の名前を指定します。この変数の値にはファイル終端指示子が格納されます。

制限事項 この引数は、POINT=オプションを指定した MODIFY ステートメントには使用しないでください。POINT=オプションは、MODIFY ステートメントでランダムアクセスを使用することを示しています。ランダムアクセスの場合、END=オプションに指定した変数の値が 1 に設定されることはありません。

注 この変数の初期値は 0 です。この変数の値が 1 に設定されるのは、MODIFY ステートメントによって、変更するデータセットの最後のオブザベーションが読み込まれた場合(順次アクセスの場合)、またはトランザクションデータセットの最後のオブザベーションが読み込まれた場合(マッチングアクセスの場合)です。また、この変数の値は、MODIFY ステートメントで KEY=オプションに指定した値に一致する値が見つからなかった場合にも 1 に設定されます(ランダムアクセスの場合)。

この変数はどのデータセットにも追加されません。

KEY=index

変更する SAS データセットの単一インデックスまたは複合インデックスを指定します。KEY=オプションを指定すると、他のソースに含まれる同一の変数名のインデックス値に基づいて、SAS データセットからオブザベーションを取得します。

デフォルト KEY=オプションの値が見つからない場合、自動変数_ERROR_の値は 1 に設定されます。また、自動変数_IORC_は、SYSRC 自動呼び出しマクロの二重モニック_DSENOM に対応する値を受け入れます。参照: [“自動変数_IORC_と SYSRC 自動呼び出しマクロ” \(269 ページ\)](#)。

制限事項 KEY=オプションの処理は、SAS/ACCESS Engine によって異なります。詳細については、SAS/ACCESS のドキュメントを参照してください。

ヒ KEYRESET=オプションを使って、読み込むデータセットのインデックスの
ン 先頭から KEY=オプションに指定された値の検索を開始するかどうかをコ
ト ントロールします。

インデックス値のソースの例として、SET ステートメントに指定した別の SAS データセットや INPUT ステートメントで読み込まれた外部ファイルなどがあります。

マスターデータセットにインデックス値の重複がある場合、そのインデックス値を持つ最初のオブザベーションしか更新されません。これを回避するには、DO ループ処理を使用して、KEY=オプションに指定したデータセットの SET ステートメントがマスターデータセットの重複インデックス値に対して繰り返し実行されるようにします。

トランザクションデータセットの重複インデックス値が連続している場合、UNIQUE オプションを指定して、マスターデータセットでのインデックス値の検索が、常にインデックスの先頭から実行されるようにします。さらに、合計ステートメントを使用して、トランザクションデータセットの重複インデックス値のオブザベーションがマスターデータセットのオブザベーションに追加されるようにする必要があります。UNIQUE オプションを指定しない場合、重複インデックス値のうち、最初に検出されたトランザクションデータセットのオブザベーションのみを使用してマスターデータセットを更新します。

トランザクションデータセットの重複インデックスが連続していない場合、検索は毎回インデックスの先頭から開始されるので、重複インデックス値がマスターデータセットにそれぞれ適用されます。合計ステートメントを作成すると、トランザクションデータセットの重複インデックス値のオブザベーションがマスターに追加されます。

参
照
項
目:

["KEYRESET=variable" \(264 ページ\)](#)

[UNIQUE \(266 ページ\)](#)

例 ["例 5: インデックスにより検索されるオブザベーションの変更" \(278 ページ\)](#)

["例 6: 重複するインデックス値の処理" \(279 ページ\)](#)

["例 7: I/O の制御" \(281 ページ\)](#)

KEYRESET=variable

読み込むデータセットのインデックスの先頭から KEY=オプションに指定された値の検索を開始するかどうかをコントロールします。KEYRESET 変数の値が 1 の場合、インデックスの一番上から検索します。KEYRESET 変数の値が 0 の場合、インデックスの検索はリセットされないで、最後の検索のつづきから検索します。

操作 KEYRESET=オプションは UNIQUE オプションに似ていますが、KEYRESET=オプションでは KEY = 検索をインデックスの最初からにするかどうかを決められます。

参照項目: ["KEY=index" \(263 ページ\)](#)

["UNIQUE" \(266 ページ\)](#)

NOBS=variable

作成する一時変数の名前を指定します。この一時変数の値は、入力データセットに含まれるオブザベーションの合計数になります。特定の SAS ビューと TAPE や XML エンジンのような順次エンジンに対しては、SAS がオブザベーション数を検知することができません。この場合、NOBS=オプションに指定した変数の値には、動作環境で使用できる最大の正の整数値が設定されます。

注 コンパイル時に、データセットのディスクリプター情報を読み込んでから、NOBS=オプションの変数の値を自動的に割り当てます。そのため、MODIFY ステートメントの実行前にも NOBS=オプションの変数を参照できます。この変数は DATA ステップで使用できますが、新しいデータセットには追加されません。

ヒント NOBS=オプションと POINT=オプションは互いに独立しています。

例 ["例 4: オブザベーション番号により検索されるオブザベーションの変更" \(276 ページ\)](#)

POINT=variable

オブザベーション番号によるランダム(ダイレクト)アクセスで SAS データセットを読み込みます。variable は、読み込むオブザベーションの番号を値とする変数名です。POINT=オプションに指定した変数は DATA ステップのどの位置にでも指定できますが、SAS データセットには追加されません。

制限事項 POINT=オプションは次のものとは併用できません。

- BY ステートメント
- WHERE ステートメント
- WHERE=データセットオプション
- 移送形式のデータセット
- テープまたはディスクに記録された順次データセット
- 他社製のリレーショナルデータベース管理システムにあるテーブル

圧縮データセットで POINT=オプションを使用できるのは、POINTOBS=データセットオプションを YES(デフォルト値)に設定してこのデータセットを作成した場合のみです。

SAS バージョン 7 以降を使用している場合のみ、圧縮ファイルにランダムアクセスメソッドを使用できます。

要件 POINT=引数を使用する場合、次のプログラミング構成のどちらかまたは両方を使用してください。

- STOP ステートメント
- POINT=オプションに指定した変数の値が無効かどうかを確認するプログラミングロジック

POINT=オプションでは指定したオブザベーションのみを読み込むため、順次アクセスでファイルを読み込む場合とは違い、ファイル終端条件を読み取ることができません。ファイル終端条件を検出すると DATA ステップを自動的に終了させるため、POINT=オプションの使用時に DATA ステップを終了させる別の方法を準備しないと、DATA ステップで無限ループが発生する場合があります。

ヒント POINT=オプションに指定した変数の値がオブザベーションの番号と一致しない場合、自動変数_ERROR_の値が 1 に設定されます。

例 [“例 4: オブザベーション番号により検索されるオブザベーションの変更” \(276 ページ\)](#)

UNIQUE

変更するデータファイルの先頭から KEY=オプションに指定した値の検索を開始します。

制限事項 UNIQUE オプションは、KEY=オプションを指定する場合にのみ使用できます。

ヒント UNIQUE オプションは、トランザクションデータセットの中に KEY=オプションに指定したインデックス値の重複が連続して存在する場合に使用します。このオプションを使用すると、トランザクションデータセットにある重複した値ごとに、マスターデータセットでの一致する値の検索をインデックスファイルの先頭から開始します。この場合、合計ステートメントを使用する必要があります。合計ステートメントを使用しないと、重複した値による上書きが発生するため、最後のトランザクションの値のみがマスターオブザベーションに反映されます。

参照 [“KEYRESET=variable” \(264 ページ\)](#)

項目:

例 [“例 6: 重複するインデックス値の処理” \(279 ページ\)](#)

UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある既存の変数の値を置き換えるかどうかを指定します。

MISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えないように指示します。

NOMISSINGCHECK

欠損値があるかどうかのチェックを実行しません。トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えられるようにします。

デフォルト	MISSINGCHECK
要件	オブザベーションの一致を検出する基準を指示する BY ステートメントに、UPDATEMODE オプションを指定する必要があります。
ヒント	ただし、特殊欠損値は例外です。この場合、MISSINGCHECK が有効な場合でも、マスターデータセットの値は特殊欠損値で置き換えられます。

詳細

マッチングアクセス(形式 1)

マッチングアクセスメソッドでは、BY ステートメントを使用して、トランザクションデータセットのオブザベーションとマスターデータセットのオブザベーションを突き合わせて対応させます。BY ステートメントには、トランザクションデータセットとマスターデータセットの両方に含まれる変数を指定します。

MODIFY ステートメントでトランザクションデータセットからオブザベーションを読み込むと、動的 WHERE 処理を実行し、マスターデータセットの中で対応するオブザベーションを探します。マスターデータセットのオブザベーションには、次のいずれかの処理が実行されます。

- マスターデータセットの値をトランザクションデータセットの値で置き替える
- マスターデータセットから削除する
- マスターデータセットに追加する

“例 3: トランザクションデータセットを使用したオブザベーションの変更” (275 ページ)には、マッチングアクセスメソッドの使用例が紹介されています。

BY 値の重複(形式 1)

マスターデータセットに重複が存在する場合でも、トランザクションデータセットに重複が存在する場合でも、処理に影響します。

- マスターデータセットに重複が存在する場合、重複する値のうち、最初に検出された値のみが更新されます。これは、生成される WHERE ステートメントではマスターデータセット内で常に条件に一致する最初の値を探すためです。
- トランザクションデータセットに重複が存在する場合、重複した値は上書きされます。これを回避するには、合計ステートメントを作成して重複する値をすべてマスターデータセットに追加する必要があります。合計ステートメントを使用しないと、重複している値は上書きされるので、最後のオブザベーションの値だけがマスターデータセットのオブザベーションに反映されます。

インデックス付きの値によるダイレクトアクセス(形式 2)

この方法では、MODIFY ステートメントに KEY=オプションを指定する必要があります。KEY=オプションには、変更するデータセットに含まれるインデックス付きの変数の名前を指定する必要があります。もう 1 つのデータソース(通常、SET ステート

メントに指定した SAS データセットか、INPUT ステートメントで読み込んだ外部ファイル)から、インデックス付きの変数に値を提供する同じ名前の変数を得ます。MODIFY ステートメントは、インデックス値を使用して変更対象のデータセットに格納されているオブザベーションを検出します。

“例 5: インデックスにより検索されるオブザベーションの変更” (278 ページ)では、インデックス付きの値を使用したダイレクトアクセス方式の例を紹介しています。

インデックス値の重複(形式 2)

- マスターデータセットにインデックス付きの変数の値が重複して存在する場合、最初に検出された値のみを対象にして取得、変更、置き換えを実行します。DO ループ処理を使用して KEY=オプションを指定した SET ステートメントを繰り返し実行すると、重複するすべての値がトランザクションの値で更新されます。
- データソースの同一名の変数に重複した値が連続せずに存在する場合、MODIFY ステートメントは、データソースのインデックス値に一致する値が最初に検出されたマスターデータセットのオブザベーションに対して、重複するトランザクションを繰り返し適用します。そのため、重複するトランザクションの最後の値のみがマスターオブザベーションに反映されます。これを回避するには、合計ステートメントを作成し、重複するトランザクションのそれぞれの値をマスターオブザベーションに追加する必要があります。
- データソースの同一名の変数に重複した値が連続して存在する場合、データソースの中で最初に検出されたオブザベーションの値がマスターデータセットに適用されます。ただし、重複する値のうち、データソースの中で 2 番目に検出された値に対応するオブザベーションをマスターデータセットで探そうとすると、エラーが発生するため DATA ステップが終了します。このエラーを回避するには、MODIFY ステートメントに UNIQUE オプションを指定します。UNIQUE オプションを使用すると、マスターデータセットの先頭に戻ってから一致するインデックス値を取得します。この場合、合計ステートメントを記述して、重複する値をすべて合計する必要があります。合計ステートメント記述しない場合、最後に検出された値のみがマスターオブザベーションに反映されます。

“例 6: 重複するインデックス値の処理” (279 ページ)では、重複するインデックス値を扱う例を紹介しています。

- 両方のデータセットに重複するインデックス値が存在する場合、SQL を使用すると、トランザクションデータセットの重複した値をマスターデータセットの重複した値と 1 対 1 で対応させることができます。

オブザベーション番号によるダイレクトアクセス(ランダムアクセス)(形式 3)

MODIFY ステートメントに POINT=オプションを使用すると、もう 1 つのデータソース(マスターデータセットを除く)の変数名を指定することができます。この変数の値がマスターデータセットで変更するオブザベーションの番号になります。MODIFY ステートメントは POINT=オプションに指定した変数の値を使用して、データセットに格納されている変更対象のオブザベーションを取得します。(圧縮データセットに POINT=オプションを使用できるのは、データセットが POINTOBS=データセットオプションを指定して作成された場合に限られます。)

プログラミング時には、POINT=オプションの変数の値を検証すること、また、自動変数_ERROR_の値のステータスを確認することをお勧めします。

“例 4: オブザベーション番号により検索されるオブザベーションの変更” (276 ページ)では、オブザベーション番号によるダイレクトアクセス(ランダムアクセス)方式について説明しています。

注意

POINT=オプションを指定すると、無限ループが発生する可能性があります。 POINT=オプションを使用する場合は、DATA ステップを正常に停止できないと、DATA ステップで無限ループが発生する可能性があるので注意してください。STOP ステートメントを使用するか、POINT=オプションに指定した変数の値が無効かどうかを確認するプログラミングロジックを使用するか、またはこの両方を使用してください。

順次アクセス(形式 4)

順次アクセスメソッドは、MODIFY ステートメントで最も簡単な形式ですが、ダイレクトアクセスメソッドに比べると制御できる要素は少なくなります。順次アクセスメソッドを使用する場合、NOBS=オプションおよび END=オプションを使用してデータセットを変更できます。ただし、POINT=オプションまたは KEY=オプションは使用しません。

MODIFY ステートメントを使用する前にデータセットを準備する

MODIFY ステートメントの使用時にパフォーマンスを改善したり、必要な結果を取得するために準備できることがあります。詳細については、“データの操作” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

自動変数_IORC_と SYSRC 自動呼び出しマクロ

自動変数_IORC_には、MODIFY ステートメントで I/O 操作を実行するたびにリターンコードが格納されます。自動変数_IORC_の値を検証するもっとも簡単な方法は、SYSRC 自動呼び出しマクロで提供されるニーモニックコードを使用することです。それぞれのニーモニックコードには、特定の条件が記述されています。そのため、ニーモニックを使用すると、DATA ステッププログラムで発生した問題を簡単に検証できるようになります。次のコードを使用すると便利です。

_DSENMR

トランザクションデータセットのオブザベーションがマスターデータセット上に存在しないことを示します(MODIFY ステートメントと BY ステートメントを併用した場合にのみ使用)。異なる BY 値を持つ連続するオブザベーションがマスターデータセットに存在しない場合、どちらのオブザベーションにも _DSENMR が返されます。

_DSEMTR

指定した BY 値を持つ複数のトランザクションデータセットのオブザベーションがマスターデータセット上に存在しないことを示します(MODIFY ステートメントと BY ステートメントを併用した場合にのみ使用)。同じ BY 値を持つ連続したオブザベーションがマスターデータセットに存在しない場合、最初のオブザベーションに対しては_DSENMR、後続のオブザベーションに対しては_DSEMTR が返されます。

_DSENM

変更対象のデータセットに KEY=オプションまたは POINT=オプションで要求したオブザベーションが含まれていないことを示します。

_SENOCHN

オブザベーションに対して OUTPUT ステートメントまたは REPLACE ステートメントを実行しようとしています、このオブザベーションに含まれているキー値と同じ値が、一意のキー値が必要な既存のインデックス付きデータセットに存在することを示しています。

_SOK

オブザベーションが検出されたことを示しています。

注: IORCMSG 関数は、自動変数_IORC_の現在の値に関連するエラーメッセージを指定された形式で返します。

“例 7: I/O の制御” (281 ページ)では、自動変数_IORC_および SYSRC 自動呼び出しマクロの使用方法について説明しています。

DATA ステップで MODIFY が使用される場合のオブザベーションの書き込み

DATA ステップに MODIFY ステートメントが含まれている場合にオブザベーションを SAS データセットに書き出す方法は、他にどのステートメントが使用されているかによって異なります。次の方法が考えられます。

明示的なステートメントなし

現在のオブザベーションを SAS データセットの元の位置に書き込みます。このアクションは、DATA ステップの最後のアクションとして実行されます(REPLACE ステートメントが DATA ステップの最後のステートメントとして記述されているのと同じように処理)。

OUTPUT ステートメント

OUTPUT ステートメントにデータセットが指定されていない場合、DATA ステップに指定したすべてのデータセットの最後に現在のオブザベーションを書き込みます。データセットが指定されている場合、このステートメントは指定されたデータセットの最後に現在のオブザベーションを書き込みます。このアクションは、DATA ステップで OUTPUT ステートメントが検出された時点で実行されます。

REPLACE <data-set-name>ステートメント

指定された 1 つまたは複数のデータセットに現在のオブザベーションを再度書き込みます。また、引数を指定しない場合は、DATA ステートメントに指定されている各データセットに現在のオブザベーションを再度書き込みます。このアクションは、REPLACE ステートメントが検出された時点で実行されます。

REMOVE <data-set-name>ステートメント

指定された 1 つまたは複数のデータセットから現在のオブザベーションを削除します。また、引数を指定しない場合は、DATA ステートメントに指定されている各データセットから現在のオブザベーションを削除します。データセットを管理しているエンジンの特性に応じて、削除が物理的に実行される場合も、論理的に実行される場合もあります。

これらのステートメントを使用する場合は、次に注意してください。

- OUTPUT ステートメント、REPLACE ステートメント、REMOVE ステートメントがどれも指定されていない場合、デフォルトのアクションは REPLACE ステートメントになります。

- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。1 つのオブザベーションに適用する OUTPUT、REPLACE、REMOVE の各ステートメントを複数記述することができます。ただし、いったん OUTPUT、REPLACE、REMOVE のいずれかのステートメントを実行した場合は、次の REPLACE ステートメントまたは REMOVE ステートメントを実行する前に、MODIFY ステートメントを再度実行する必要があります。

この例の条件ロジックに示すように、OUTPUT ステートメントおよび REPLACE ステートメントを使用できます。これは、オブザベーションごとに REPLACE ステートメントまたは OUTPUT ステートメントのどちらか 1 つのみが実行されるためです。

```
data master;
  modify master trans; by key;
  if _iorc_=0 then replace;
  else
    output;
run;
```

この例に示すように、同じオブザベーションに対して複数の REPLACE 演算を実行しないでください。

```
data master;
  modify master;
  x=1;
  replace;
  replace;
run;
```

オブザベーションごとに複数の OUTPUT ステートメントを記述できます。ただし、複数の OUTPUT ステートメントを使用する場合は注意が必要です。OUTPUT ステートメントを 1 つだけ使用する場合でも、無限ループが発生する可能性があります。

```
data master;
  modify master;
  output;
run;
```

- DATA ステップで OUTPUT、REPLACE、REMOVE のいずれかのステートメントを使用すると、オブザベーションのデフォルトの置き換えより優先されます。DATA ステップでこれらのステートメントのいずれかを使用する場合は、実行するアクションを明示的に記述する必要があります。
- OUTPUT ステートメントと REPLACE ステートメントの両方、または REMOVE ステートメントを指定したオブザベーションに対して実行する場合、オブザベーションポインターの位置を正しく保つために OUTPUT 操作を最後に実行する必要があります。

“例 8: オブザベーションの置換と削除、オブザベーションの他の SAS データセットへの書き込み” (283 ページ)では、OUTPUT、REMOVE、REPLACE の各ステートメントを使用してオブザベーションを書き込む方法について説明しています。

欠損値と MODIFY ステートメント

デフォルトでは、UPDATEMODE=MISSINGCHECK オプションが有効です。そのため、トランザクションデータセットの欠損値によって、マスターデータセットの既存の値が置き換えられることはありません。そのため、すべての変数ではなく一部

の変数のみを更新する場合、またオブザベーションごとに更新する変数が異なる場合は、変更しない変数を欠損値に設定します。トランザクションデータに含まれる欠損値でマスターデータセットに含まれる既存の値を置き換える場合は、`UPDATEMODE=NOMISSINGCHECK` を使用します。

`UPDATEMODE=MISSINGCHECK` が有効な場合でも、トランザクションデータセットで特殊欠損値の文字を使用すると、既存の値を欠損値に置き換えることができます。特殊欠損値を含むトランザクションデータセットを作成するには、`DATA` ステップで `MISSING` ステートメントを使用します。トランザクションデータセットに A から Z までの特殊欠損値の 1 つを定義すると、マスターデータセットに含まれる数値変数はその値に更新されます。

マスターデータセットの値を通常の欠損値に設定する場合は、トランザクションデータセット内でアンダースコア(_)を 1 つ使用して欠損値を示します。このように指定すると、マスターデータセットの値は、数値の欠損値がピリオド(.)に設定され、文字の欠損値は空白に設定されます。

特殊欠損値文字の定義方法および使用方法の詳細については、“[MISSING ステートメント](#)” (261 ページ)を参照してください。

データセットオプションを用いた MODIFY の使用

プログラムでデータセットオプション(`KEEP=`オプションなど)を使用する場合は、`MODIFY` ステートメントでマスターデータセットに対してこれらのオプションを指定します。`DATA` ステートメントでデータセットオプションを使用すると、予期しない結果が生じることがあります。

SAS/SHARE 環境で MODIFY を使用する

SAS/SHARE 環境では、`MODIFY` ステートメントは更新モードでオブザベーションにアクセスします。つまり、`MODIFY` ステートメントでオブザベーションを読み込んだときから、`REPLACE` ステートメントまたは `REMOVE` ステートメントを実行するまでオブザベーションはロックされます。実行した時点でオブザベーションのロックは解除されます。`MODIFY` ステートメントで再度読み込むまで、オブザベーションにアクセスすることはできません。`MODIFY` ステートメントは更新モードでデータセットを開きますが、制御レベルは使用するステートメントによって異なります。たとえば、`KEY=`オプションと `POINT=`オプションはメンバーレベルのロックを実行します。詳細については、[SAS/SHARE ユーザーガイド](#)を参照してください。

比較

- `DATA` ステップで `MERGE`、`SET`、`UPDATE` のいずれかのステートメントを使用すると、新しい SAS データセットが作成されます。新しいデータセットのデータセットディスクリプターは、元のデータセットのディスクリプターとは異なります(変数の追加、削除、ラベルの変更などによる)。ただし、`DATA` ステップで `MODIFY` ステートメントを使用する場合は、新しいデータセットは作成されません。そのため、データセットディスクリプターが変更されることはありません。

DBMS の置き換えルールの詳細については、SAS/ACCESS のマニュアルを参照してください。

- MODIFY ステートメントを BY ステートメントと併用する場合、MODIFY ステートメントは UPDATE ステートメントとほとんど同じように機能します。ただし、次の状況は除きます。

- マスターデータセットとトランザクションデータセットのどちらも並び替えやインデックスの作成を行う必要はありません。(BY ステートメントを指定した MODIFY ステートメントでは、動的 WHERE 処理がトリガーされます。)

注: 並び替えまたはインデックスの作成を実行していない SAS データセットを MODIFY ステートメントで変更する場合、動的 WHERE 処理はシステムに負担をかけることがあります。マスターデータセットで並び替えまたはインデックスの作成を実行しておく、特にサイズが大きいファイルの場合には処理時間を削減できます。

- 重複した BY 変数の値を持つオブザベーションがマスターデータセットとトランザクションデータセットの両方に存在する場合があります。MODIFY ステートメントでの重複する値の処理方法については、“BY 値の重複(形式 1)” (267 ページ)を参照してください。
- MODIFY ステートメントでは、UPDATE ステートメントと同じように、データセットのディスクリプター情報を変更することはできません。そのため、このステートメントでは、変数の追加や削除、変数ラベルの変更などは実行できません。

例

例 1: 例で使用する入力データセットの説明

例では、INVTY.STOCK データセットを変更します。INVTY.STOCK データセットには、次の変数が含まれています。

PARTNO

文字変数です。個々のツール番号を識別する一意の値が格納されます。

DESC

文字変数です。各ツールのテキストでの説明が格納されます。

INSTOCK

数値変数です。社内で保有している各ツールの在庫数が格納されます。

RECDATE

数値変数です。どの変数 INSTOCK の値が現在のものかを示す SAS 日付の値が格納されます。

PRICE

数値変数です。各ツールの単価が格納されます。

さらに、データセット INVTY.STOCK には、変数 PARTNO の値をキーとする単一インデックスが格納されています。次の DATA ステップを実行すると、データセット INVTY.STOCK が作成されます。

```
libname invty 'SAS-library';
data invty.stock(index=(partno));
input PARTNO $ DESC $ INSTOCK @17
```

```

RECDATE date7. @25 PRICE;
format recdate date7.;
datalines;
K89R seal 34 27jul95 245.00
M4J7 sander 98 20jun95 45.88
LK43 filter 121 19may96 10.99
MN21 brace 43 10aug96 27.87
BC85 clamp 80 16aug96 9.55
NCF3 valve 198 20mar96 24.50
KJ66 cutter 6 18jun96 19.77
UYN7 rod 211 09sep96 11.55
JD03 switch 383 09jan97 13.99
BV1E timer 26 03jan97 34.50
;

```

例 2: すべてのオブザベーションの変更

この例では、データセット INVTY.STOCK のすべてのレコードにある日付を現在の日付で置き換えます。また、データセット INVTY.STOCK に格納されているすべてのオブザベーションに対して、変数 RECDATE の値を現在の日付に置き換えます。

```

data invty.stock;
  modify invty.stock;
  recdate=today();
run;
proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;

```

アウトプット 2.28 RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	34	08DEC10	245.00
M4J7	sander	98	08DEC10	45.88
LK43	filter	121	08DEC10	10.99
MN21	brace	43	08DEC10	27.87
BC85	clamp	80	08DEC10	9.55
NCF3	valve	198	08DEC10	24.50
KJ66	cutter	6	08DEC10	19.77
UYN7	rod	211	08DEC10	11.55
JD03	switch	383	08DEC10	13.99
BV1E	timer	26	08DEC10	34.50

MODIFY ステートメントは、更新処理を行うためにデータセット INVTY.STOCK を開きます。DATA ステップの 1 回の繰り返しで、データセット INVTY.STOCK のオブザベーションを 1 つ読み込みます。また、コードに記述された処理を実行します。この場合、DATA ステップを繰り返すたびに、変数 RECDATE の値を TODAY 関数の結果で置き換えます。DATA ステップの最後に暗示的な REPLACE ステートメントを

指定すると、各オブザベーションがデータセット INVTY.STOCK の元の位置に書き込まれます。

例 3: トランザクションデータセットを使用したオブザベーションの変更

この例では、新しく受け取った在庫数をデータセット INVTY.STOCK に追加し、在庫を受け取った日付を更新します。WORK ライブラリにあるトランザクションデータセット ADDINV には新しいデータが含まれています。

ADDINV データセットは、更新された情報を格納するデータセットです。また、ADDINV データセットには、次の変数が格納されています。

PARTNO

文字変数です。データセット INVTY.STOCK のインデックス付き変数 PARTNO に対応しています。

NWSTOCK

数値変数です。新しく受け取った各ツールの在庫数を示しています。

データセット ADDINV は、MODIFY ステートメントに指定されている 2 番目のデータセットです。そのため、データセット ADDINV をトランザクションデータセットとして使用し、このデータセットの各オブザベーションを順番に読み込みます。BY ステートメントには共通する変数 PARTNO が指定されています。そのため、MODIFY ステートメントは、データセット ADDINV の変数 PARTNO の値に最初に一致する値を、データセット INVTY.STOCK の変数 PARTNO の値から探します。一致する値が検出されたオブザベーションごとに、DATA ステップによって変数 RECDATE の値が本日の日付に変更されます。また、変数 INSTOCK の値がこれまでの変数 INSTOCK の値と ADDINV に格納されている変数 NWSTOCK の値の合計に置き換えられます。MODIFY ステートメントでは、データセットディスクリプターが変更されるので、変数 NWSTOCK はデータセット INVTY.STOCK に追加されません。そのため、DROP ステートメントで変数 NWSTOCK を指定する必要はありません。

この例では、データセット ADDINV をトランザクションデータセットとして指定します。このデータセットにはデータセット INVTY.STOCK を変更するための情報が格納されています。共通する変数を BY ステートメントに指定します。この変数の値を基にして、データセット INVTY.STOCK に格納されているオブザベーションを検索します。

次の DATA ステップを実行すると、データセット ADDINV が作成されます。

```
data addinv;
  input PARTNO $ NWSTOCK;
  datalines;
K89R 55
M4J7 21
LK43 43
MN21 73
BC85 57
NCF3 90
KJ66 2
UYN7 108
JD03 55
BV1E 27
;
```

次の DATA ステップでは、データセット ADDINV の値を使用してデータセット INVTY.STOCK を更新します。

```
libname invty 'SAS-library';
data invty.stock;
  modify invty.stock addinv;
  by partno;
  RECDATE=today();
  INSTOCK=instock+nwstock;
  if _iorc_=0 then replace;
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

アウトプット 2.29 INSTOCK フィールドおよび RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	119	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

例 4: オブザベーション番号により検索されるオブザベーションの変更

この例では、データセット NEWP を読み込みます。次に、データセット INVTY.STOCK の中で更新対象とするオブザベーション番号を TOOL_OBS の値に基づいて特定してから、更新を実行します。ここでは、変数 PRICE の値を変数 NEWP の値で置き換えるため、割り当てステートメントを使用して更新作業を明示的に指定しています。

データセット NEWP には、次の 2 つの変数が格納されています。

TOOL_OBS

ツール会社のマスターデータセット INVTY.STOCK にある各ツールのオブザベーション番号が格納されています。

NEWP

各ツールの新価格が格納されています。

次の DATA ステップを実行すると、データセット NEWP が作成されます。

```
data newp;
  input TOOL_OBS NEWP;
  datalines;
1 251.00
2 49.33
3 12.32
4 30.00
5 15.00
6 25.75
7 22.00
8 14.00
9 14.32
10 35.00
;
```

この DATA ステップは、INVTY.STOCK を更新します。

```
libname invty 'SAS-library';
data invty.stock;
  set newp;
  modify invty.stock point=tool_obs
    nobs=max_obs;
  if _error_=1 then
  do;
    put 'ERROR occurred for TOOL_OBS=' tool_obs /
      'during DATA step iteration' _n_ /
      'TOOL_OBS value might be out of range.';
    _error_=0;
    stop;
  end;
  PRICE=newp;
  RECDATE=today();
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```

アウトプット 2.30 RECDATE フィールドおよび PRICE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	34	08DEC10	251.00
M4J7	sander	98	08DEC10	49.33
LK43	filter	121	08DEC10	12.32
MN21	brace	43	08DEC10	30.00
BC85	clamp	80	08DEC10	15.00
NCF3	valve	198	08DEC10	25.75
KJ66	cutter	6	08DEC10	22.00
UYN7	rod	211	08DEC10	14.00
JD03	switch	383	08DEC10	14.32
BV1E	timer	26	08DEC10	35.00

例 5: インデックスにより検索されるオブザベーションの変更

この例では、KEY=オプションを使用して、データセット ADDINV の変数 PARTNO の値とデータセット INVTY.STOCK の変数 PARTNO のインデックス付きの値で一致する値を探し、取得するオブザベーションを特定します。データセット ADDINV は、“例 3: トランザクションデータセットを使用したオブザベーションの変更” (275 ページ) で作成されています。

KEY=オプションによって、MODIFY ステートメントで更新対象のオブザベーションに直接アクセスできるインデックス値が指定されます。動的 WHERE 処理は実行されません。この例では、マスターデータセット INVTY.STOCK の変数 INSTOCK の値に、トランザクションデータセット ADDINV の変数 NWSTOCK の値を追加します。

```
libname invty 'SAS-library';

data invty.stock;
  set addinv;
  modify invty.stock key=partno;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK';
run;
```


アウトプット 2.31 インデックスを使用した INSTOCK フィールドと RECDATE フィールドの更新結果

INVTY.STOCK				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	119	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

例 6: 重複するインデックス値の処理

この例では、SET データセットに格納されている重複した変数の値を MODIFY ステートメントで処理する方法を示します。この SET データセットによって、マスターデータセットのインデックス値が提供されます。

データセット NEWINV は、更新された情報が格納するデータセットです。また、データセット NEWINV には、次の変数が格納されています。

PARTNO

文字変数です。データセット INVTY.STOCK のインデックス付き変数 PARTNO に対応しています。データセット NEWINV では、変数 PARTNO に重複した値が存在しています。この場合、**M4J7** が 2 つ含まれています。

NWSTOCK

数値変数です。新しく受け取った各ツールの在庫数を示しています。

次の DATA ステップを実行すると、データセット NEWINV が作成されます。

```
data newinv;
  input PARTNO $ NWSTOCK;
  datalines;
K89R 55
M4J7 21
M4J7 26
LK43 43
MN21 73
BC85 57
NCF3 90
KJ66 2
UYN7 108
JD03 55
BV1E 27
;
```

次を実行すると、データセット NEWINV で 2 番目に検出される **M4J7** に一致するオブザベーションをデータセット INVTY.STOCK で探そうとしたときにエラーが発生し、DATA ステップが異常終了します。

```
libname invty 'SAS-library';
/* This DATA step terminates with an error! */
data invty.stock;
  set newinv;
  modify invty.stock key=partno;
  INSTOCK=instock+nwstock;
  RECDATE=today();
run;
```

SAS ログには次のメッセージが表示されます。

```
ERROR: No matching observation was found in MASTER data set.
PARTNO=M4J7 NWSTOCK=26 DESC=sander INSTOCK=166 RECDATE=08DEC10 PRICE=45.88 _ERROR_=1
_IORC_=1230015 _N_=3
NOTE: The SAS System stopped processing this step because of errors.
NOTE: There were 3 observations read from the data set WORK.NEWINV.
NOTE: The data set INVTY.STOCK has been updated. There were 2 observations rewritten, 0
      observations added and 0 observations deleted.
```

MODIFY ステートメントに UNIQUE オプションを追加すると、前述の DATA ステップで発生したエラーを回避することができます。UNIQUE オプションを追加すると、SET データセットで一致する値を検索するたびに、インデックスの先頭に戻ってから検索を実行します。そのため、SET データセットで検出されるそれぞれの **M4J7** に対して、マスターデータセットに含まれる **M4J7** が検出されます。出力される **M4J7** の更新結果では、データセット NEWINV にある変数 NWSTOCK の 2 つの **M4J7** の値が、データセット INVTY.STOCK の変数 INSTOCK の値 **M4J7** に追加されます。次に、合計ステートメントで値を合計します。合計ステートメントを指定しない場合、**M4J7** の最後のインスタンスの値だけがデータセット INVTY.STOCK に反映されます。

```
data invty.stock;
  set newinv;
  modify invty.stock key=partno / unique;
  INSTOCK=instock+nwstock;
  RECDATE=today();
  if _iorc_=0 then replace;
run;
proc print data=invty.stock noobs;
  title 'Results of Using the UNIQUE Option';
run;
```

アウトプット 2.32 UNIQUE オプションを使用した INSTOCK フィールドと RECDATE フィールドの更新結果

Results of Using the UNIQUE Option

PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	89	08DEC10	245.00
M4J7	sander	145	08DEC10	45.88
LK43	filter	164	08DEC10	10.99
MN21	brace	116	08DEC10	27.87
BC85	clamp	137	08DEC10	9.55
NCF3	valve	288	08DEC10	24.50
KJ66	cutter	8	08DEC10	19.77
UYN7	rod	319	08DEC10	11.55
JD03	switch	438	08DEC10	13.99
BV1E	timer	53	08DEC10	34.50

例 7: I/O の制御

この例では、SYSRC 自動呼び出しマクロと自動変数_IORC_を使用して、I/O 条件を制御しています。これにより、検出されない可能性のある予期しない結果を防ぐことができます。この例では、インデックス値を使用したダイレクトアクセスメソッドで、データセット INVTY.STOCK を更新します。データセット NEWSHIP のデータにより、データセット INVTY.STOCK が更新されます。

次の DATA ステップを実行すると、データセット NEWSHIP が作成されます。

```
data newship;
  input PARTNO $ DESC $ NWSTOCK @17
        SHPDATE date7. @25 NWPRICE;
  datalines;
K89R seal 14 14nov96 245.00
M4J7 sander 24 23aug96 47.98
LK43 filter 11 29jan97 14.99
MN21 brace 9 09jan97 27.87
BC85 clamp 12 09dec96 10.00
ME34 cutter 8 14nov96 14.50
;
```

SELECT ステートメントに指定したそれぞれの WHEN 句では、SYSRC 自動呼び出しマクロが返す入出力のリターンコードごとに対応するアクションを指定しています。

- _SOK は、MODIFY ステートメントが正常に実行されたことを示します。
- _DSENO は、データセット INVTY.STOCK の中に一致するオブザベーションが見つからなかったことを示します。OUTPUT ステートメントでは、データセット INVTY.STOCK にオブザベーションを追加するように指定しています。出力結果に表示される最後のオブザベーションを確認してください。

- SYSRC 自動呼び出しマクロで他のコードが返された場合は、DATA ステップを終了し、PUT ステートメントでメッセージを SAS ログに出力します。

```
libname invty 'SAS-library';

data invty.stock;
  set newship;
  modify invty.stock key=partno;
  select (_iorc_);
    when (%sysrc(_sok)) do;
      INSTOCK=instock+nwstock;
      RECDATE=shpdate;
      PRICE=nwprice;
      replace;
    end;
    when (%sysrc(_dsenom)) do;
      INSTOCK=nwstock;
      RECDATE=shpdate;
      PRICE=nwprice;
      output;
      _error_=0;
    end;
    otherwise do;
      put
        'An unexpected I/O error has occurred.'/
        'Check your data and your program';
      _error_=0;
      stop;
    end;
  end;
run;

proc print data=invty.stock noobs;
  title 'INVTY.STOCK Data Set';
run;
```

アウトプット 2.33 更新後の INVTY.STOCK データセット

INVTY.STOCK Data Set

PARTNO	DESC	INSTOCK	RECDATE	PRICE
K89R	seal	48	14NOV96	245.00
M4J7	sander	122	23AUG96	47.98
LK43	filter	132	29JAN97	14.99
MN21	brace	52	09JAN97	27.87
BC85	clamp	92	09DEC96	10.00
NCF3	valve	198	20MAR96	24.50
KJ66	cutter	6	18JUN96	19.77
UYN7	rod	211	09SEP96	11.55
JD03	switch	383	09JAN97	13.99
BV1E	timer	26	03JAN97	34.50
ME34	cutter	8	14NOV96	14.50

例 8: オブザベーションの置換と削除、オブザベーションの他の SAS データセットへの書き込み

この例では、オブザベーションの置き換えや削除、別のデータセットへのオブザベーションの書き込みが実行できることを示しています。この例に示すように、OUTPUT ステートメント、REPLACE ステートメント、REMOVE ステートメントを使用する場合、デフォルトのステートメントが生成されないので、実行するアクションを明示的に指定する必要があります。

1997 年に受け取った部品はデータセット INVTY.STOCK97 に書き込み、データセット INVTY.STOCK から削除します。同じように 1995 年に受け取った部品はデータセット INVTY.STOCK95 に書き込み、データセット INVTY.STOCK から削除します。1996 年に受け取った部品だけはデータセット INVTY.STOCK に残し、データセット INVTY.STOCK の中でだけ PRICE の値を更新します。

```

libname invty 'SAS-library';
data invty.stock invty.stock95 invty.stock97;
  modify invty.stock;
  if recdate > '01jan97'd then do;
    output invty.stock97;
    remove invty.stock;
  end;
  else if recdate < '01jan96'd then do;
    output invty.stock95;
    remove invty.stock;
  end;
  else do;
    price = price * 1.1;
    replace invty.stock;
  end;
run;

```

```
proc print data=invty.stock noobs;  
  title 'New Prices for Stock Received in 1996';  
run;
```

アウトプット 2.34 特定の SAS データセットにオブザベーションを書き込んだ後の出力結果

New Prices for Stock Received in 1996				
PARTNO	DESC	INSTOCK	RECDATE	PRICE
LK43	filter	121	19MAY96	12.089
MN21	brace	43	10AUG96	30.657
BC85	clamp	80	16AUG96	10.505
NCF3	valve	198	20MAR96	26.950
KJ66	cutter	6	18JUN96	21.747
UYN7	rod	211	09SEP96	12.705

関連項目

- [SAS SQL プロシジャユーザーガイド](#)

ステートメント:

- [“MISSING ステートメント” \(261 ページ\)](#)
- [“OUTPUT ステートメント” \(285 ページ\)](#)
- [“REMOVE ステートメント” \(337 ページ\)](#)
- [“REPLACE ステートメント” \(341 ページ\)](#)
- [“UPDATE ステートメント” \(380 ページ\)](#)

null ステートメント

データの終わりを示すか、またはブレースホルダーとして動作します。

注: [null ステートメント](#)は SAS グローバルステートメントに移動しました。

OPTIONS ステートメント

1 つまたは複数の SAS システムオプションの値の指定や変更を行います。

注: [OPTIONS ステートメント](#)は SAS グローバルステートメントに移動しました。

OUTPUT ステートメント

現在のオブザベーションを SAS データセットに書き込みます。

該当要素: DATA ステップ

カテゴリ: アクション

CAS

In-Database 処理

種類: 実行

制限事項: [2025.03](#) 以降、このステートメントは In-database DATA ステップでサポートされています。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

OUTPUT <[data-set-name\(s\)](#)>;

引数なし

引数を指定せずに OUTPUT を使用すると、現在のオブザベーションが DATA ステートメントに指定したすべてのデータセットに書き込まれます。

MODIFY ステートメントが存在する場合、引数を指定せずに OUTPUT ステートメントを使用すると、現在のオブザベーションが MODIFY ステートメントに指定したデータセットの最後に書き込まれます。

引数

data-set-name

オブザベーションを書き込むデータセットの名前を指定します。

制限事項 OUTPUT ステートメントに指定したすべての名前は、DATA ステートメントにも指定する必要があります。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

OUTPUT ステートメントには、DATA ステップの DATA ステートメントに指定したデータセット数と同じ数までデータセットを指定できます。

詳細

OUTPUT ステートメントがオブザベーションを書き込むタイミングと場所

OUTPUT ステートメントは、DATA ステップの最後ではなく、現在のオブザベーションをすぐに SAS データセットに書き込みます。OUTPUT ステートメントにデータセット名が指定されていない場合、DATA ステートメントにリストされている 1 つまたは複数のデータセットにオブザベーションが書き込まれます。

暗示的な OUTPUT と明示的な OUTPUT

デフォルトでは、すべてのデータステップで繰り返しの最後に暗示的な OUTPUT ステートメントが含まれ、作成中の 1 つまたは複数のデータセットにオブザベーションが書き込まれます。OUTPUT ステートメントを DATA ステップに明示的に指定すると、この自動出力より優先されるため、明示的な OUTPUT ステートメントの実行時のみオブザベーションがデータセットに追加されるようになります。OUTPUT ステートメントを使用して 1 つでもデータセットにオブザベーションを書き込むようにすると、DATA ステップの最後に暗示的な OUTPUT ステートメントは存在しなくなります。そのため、DATA ステップでオブザベーションが書き込まれるのは、明示的な OUTPUT ステートメントの実行時のみになります。OUTPUT ステートメントはそれだけでも使用できますが、IF-THEN ステートメント、SELECT ステートメント、DO ループの処理と組み合わせても使用できます。

MODIFY ステートメントを使用する場合

MODIFY ステートメントを OUTPUT ステートメントと組み合わせて使用する場合、REMOVE および REPLACE の各ステートメントは、DATA ステップの繰り返しの最後の暗示的な書き込みアクションより優先されます。詳細については、“[比較](#)” (286 ページ) を参照してください。特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインターの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。

比較

- OUTPUT ステートメントはオブザベーションを SAS データセットに書き込みますが、PUT ステートメントは変数の値またはテキスト文字列を外部ファイルや SAS ログに書き込みます。
- 指定した出力データセットにオブザベーションを書き込むタイミングを制御するには、OUTPUT ステートメントを使用します。指定した出力データセットに書き込む変数を制御するには、DATA ステートメントで KEEP= または DROP= データセットオプションを使用するか、KEEP または DROP ステートメントを使用します。
- MODIFY ステートメントと組み合わせて OUTPUT ステートメントを使用する場合、次のようになります。

- OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)これらのステートメントのいずれかを DATA ステップで使用する場合、データセットに新しいオブザベーションが追加されるように出力を明示的にプログラムする必要があります。
- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。
- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインターの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。

例

例 1: OUTPUT ステートメントの使用例

これらの例では、OUTPUT ステートメントの使用方法を示します。

- 次のコード行では、現在のオブザベーションが SAS データセットに書き込まれます。

```
output;
```

- 次のコード行では、指定した条件が真の場合に現在のオブザベーションが SAS データセットに書き込まれます。

```
if deptcode gt 2000 then output;
```

- 次のコード行では、PHONE の値が存在しない場合にオブザベーションがデータセット MARKUP に書き込まれます。

```
if phone=. then output markup;
```

例 2: 1 つの入力行から複数のオブザベーションを作成する

入力データの 1 つの行から複数のオブザベーションを作成することができます。次の SAS プログラムでは、データセット SULFA 内の 1 つのオブザベーションから、データセット RESPONSE 内に 3 つのオブザベーションが作成されます。

```
data response(drop=time1-time3);
  set sulfa;
  time=time1;
  output;
  time=time2;
  output;
  time=time3;
  output;
run;
```

例 3: 1 つの入力ファイルから複数のデータセットを作成する

1 つの入力ファイルから複数の SAS データセットを作成できます。この例の OUTPUT ステートメントは、2 つのデータセット OZONE と OXIDES にオブザベーションを書き込みます。

```
data ozone oxides;
  infile file-specification;
  input city $ 1-15 date date9.
         chemical $ 26-27 ppm 29-30;
  if chemical='O3' then output ozone;
  else output oxides;
run;
```

例 4: 複数の入力行から 1 つのオブザベーションを作成する

複数の入力オブザベーションを 1 つのオブザベーションにまとめることができます。この例の OUTPUT ステートメントは、入力データセットにある最初の 10 個のオブザベーションの DEFECTS 値を合計し、1 つのオブザベーションを作成します。

```
data discards;
  set gadgets;
  drop defects;
  reps+1;
  if reps=1 then total=0;
  total+defects;
  if reps=10 then do;
    output;
    stop;
  end;
run;
```

関連項目

ステートメント:

- [“DATA ステートメント” \(68 ページ\)](#)
- [“MODIFY ステートメント” \(261 ページ\)](#)
- [“PUT ステートメント” \(289 ページ\)](#)
- [“REMOVE ステートメント” \(337 ページ\)](#)
- [“REPLACE ステートメント” \(341 ページ\)](#)

PAGE ステートメント

SAS ログの新しいページにスキップします。

注: [PAGE ステートメント](#) は SAS グローバルステートメントに移動しました。

PUT ステートメント

SAS ログ、SAS 出力ウィンドウ、または最後の FILE ステートメントに指定した外部の場所に行を書き込みます。

該当要素:	DATA ステップ
カテゴリ:	CAS ファイル操作
種類:	実行
制限事項:	このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“ DATA Step Acceleration ” (<i>SAS In-Database Products: User's Guide</i>)を参照してください。
操作:	FILE ステートメントで DSD オプションを使用し、PUT ステートメントを使用して SAS ログ、PRINT、または外部ファイルに書き込む場合、出力で欠損値が抑制される場合があります。PUT ステートメントに明示的な改行(/)が含まれていれば、欠損値を表示することができます。

構文

PUT <*specification(s)*> <> <@ | @@>;

PUT 'ERROR: *text*' | 'NOTE: *text*' | 'WARNING: *text*';

PUT 'ERROR- *text*' | 'NOTE: *text*' | 'WARNING- *text*';

引数なし

引数を指定しない PUT ステートメントは、*null PUT* ステートメントと呼ばれます。これらの作業を行うには、null PUT ステートメントを使用することができます。

- 現在の出力行がブランクの場合でも、現在の出力行を現在の場所に行書き込みます。
- 前回の PUT ステートメントの後置@で保持された出力行を解除します。

例については、“[例 5: 出力行の保持と解放](#)” (310 ページ)を参照してください。詳細については、“[ラインホールド指定子の使用](#)” (303 ページ)を参照してください。

引数

specification(s)

書き込む内容、書き込み方法、書き込み先を指定します。指定には 1 つ以上のアイテムを記載することができます。

variable

変数の名前を指定します。指定した変数の値が書き込まれます。

(variable-list)

変数のリストを指定します。指定した変数の値が書き込まれます。

要件 (format-list)は(variable-list)の後ろに指定する必要があります。

参照項目: [“PUT ステートメント: フォーマット” \(316 ページ\)](#)

'character-string'

書き込む文字列を引用符で囲んで指定します。

ヒン 16 進数の文字列を EBCDIC または ASCII で書き込むには、終了引用符
ト の後ろに x を追加します。

間にブランクを含まない単一引用符(')または二重引用符("")の記号のみをテキスト文字列として指定すると、単一引用符(')または二重引用符("")がそれぞれ書き出されます。

参照 [“リスト出力” \(300 ページ\)](#)
項目:

例 次のステートメントでは、16 進数の文字列を ASCII 文字に変換してから、HELLO と書き込みます。
`put '68656C6C6F'x;`

n*

後ろに指定する文字列を *n* 回繰り返します。

例 次のステートメントでは、出力行にアンダースコアを 132 回書き込みます。
`put 132*'_';`

例 [“例 4: テキストに下線を引く” \(309 ページ\)](#)

pointer-control

出力バッファ内の指定した行または列に出力ポインターを移動させます。

参照項目: [“引数: 列:ポインターコントロール” \(291 ページ\)](#)

[“引数: 行:ポインターコントロール” \(293 ページ\)](#)

column-specifications

値を書き込む出力行内の列位置を指定します。

参照項目: [“列出力” \(299 ページ\)](#)

例 [“例 2: ポインターをページ内で移動させる” \(306 ページ\)](#)

format.

変数の値を書き込むときに使用する出力形式を指定します。

参照項目: [“フォーマット出力” \(300 ページ\)](#)

例 [“例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する” \(305 ページ\)](#)

(format-list)

出力形式のリストを指定します。このリストは、前に指定した変数リストの値を書き込むときに使用します。

制限事項 (format-list)は(variable-list)の後ろに指定する必要があります。

参照項目: [“PUT ステートメント: フォーマット” \(316 ページ\)](#)

INFILE

現在の入力ファイルから、または DATALINES ステートメントの後ろに指定したデータ行のどちらかから読み込んだ最後の入力データレコードを書き込みます。

ヒ _INFILE_ は、現在の INPUT バッファを参照する自動変数です。他の
ン SAS ステートメントでもこの自動変数を使用できます。
ト

最後に使用した INPUT ステートメントが行ポインターコントロールを使用して複数の入力データレコードを読み込む場合、PUT _INFILE_ では、入力ポインターが置かれたレコードのみを書き込みます。

例 次の PUT ステートメントでは、1 行目の入力データレコードの値をすべて書き込みます。

```
input #3 score #1 name $ 6-23;
put _infile_;
```

例 [“例 6: 現在の入力レコードを SAS ログに書き込む” \(310 ページ\)](#)

ALL

現在の DATA ステップにて定義されるすべての変数の値を、自動変数を含めて書き込みます。名前付き出力を使用します。

参照項目: [“名前付き出力” \(300 ページ\)](#)

@ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、後置@および後置@@と呼ばれます。

制限事項 後置@または後置@@は、PUT ステートメントの最後のアイテムとして指定する必要があります。

ヒント ポインターを現在の位置に保持するには、@または@@を使用します。次の PUT ステートメントは、新しい出力行ではなく同じ出力行に書き込みます。

参照項目: [“ラインホールド指定子の使用” \(303 ページ\)](#)

例 [“例 5: 出力行の保持と解放” \(310 ページ\)](#)

引数: 列:ポインターコントロール**@n**

ポインターを列 *n* に移動させます。

範囲 正の整数

例 正の整数@15 が指定されているため、NAME の値を書き込む前に、ポインターを列 15 に移動させます。

```
put @15 name $10.;
```

例 “例 2: ポインターをページ内で移動させる” (306 ページ)および

“例 4: テキストに下線を引く” (309 ページ)

@numeric-variable

指定した *numeric-variable* の値が示す列にポインターを移動させます。

範囲 正の整数

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*n* が 0 または負の場合、ポインターは列 1 に移動します。

例 変数 A の値に従って、NAME 変数の値を書き込む前に、ポインターを列 15 に移動させます。

```
a=15;
put @a name $10.;
```

例 “例 2: ポインターをページ内で移動させる” (306 ページ)

@(expression)

指定した *expression* の値が示す列にポインターを移動させます。

範囲 正の整数

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。値が 0 になる場合、ポインターは列 1 に移動します。

例 式の結果に従って、NAME 変数の値を書き込む前に、ポインターを列 15 に移動させます。

```
b=5;
put @(b*3) name $10.;
```

+n

n に指定した列数だけポインターを移動させます。

範囲 正の整数または 0

ヒント *n* が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

例 次のステートメントでは、ポインターを列 23 に移動させ、LENGTH の値を列 23 から 26 に書き込みます。次に、ポインターを 5 列だけ右に移動させ、列 32 から 35 に WIDTH の値を書き込みます。

```
put @23 length 4. +5 width 4.;
```

+numeric-variable

指定した *numeric-variable* の値が示す列数だけポインターを移動させます。

範囲 正の整数、負の整数、または 0

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*numeric-variable* の値が負の値の場合、ポインタを後方(左)へ移動させます。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が出力バッファの長さを超える場合、現在の行を書き込んだ後、ポインタは次の行の列 1 に移動します。

+(expression)

指定した *expression* の値が示す列数だけポインタを移動させます。

範囲 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。*expression* の実行結果が負の値の場合、ポインタを左に戻します。ただし、現在の列位置が 1 未満になる場合、ポインタは列 1 に移動します。値が 0 の場合、ポインタは移動しません。この値が出力バッファの長さを超える場合、現在の行を書き込んだ後、ポインタは次の行の列 1 に移動します。

例 “例 2: ポインタをページ内で移動させる” (306 ページ)

引数: 行:ポインタコントロール

#n

行 *n* の列 1 にポインタを移動させます。

範囲 正の整数

例 正の整数#2 と指定されているので、ポインタは ID の値を列 3 と 4 に書き込む前に 2 行目に移動します。

```
put @12 name $10. #2 id 3-4;
```

#numeric-variable

指定した *numeric-variable* の値が示す行の列 1 にポインタを移動させます。

範囲 正の整数

ヒント *numeric-variable* の値が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

#(expression)

指定した *expression* の値が示す行の列 1 にポインタを移動させます。

範囲 *expression* の実行結果は正の整数でなければなりません。

ヒント *expression* の実行結果が整数ではない場合、小数点以下の値を切り捨て、整数の値のみを使用します。

/

ポインタを次の行の列 1 に移動させます。

注 1 つまたは複数の“//”行ポインターコントロールを使用して SAS ログにブランク行を追加しようとしても、ブランク行は生成されません。それ以外の出力形式では、ブランク行が生成されます。

例 変数 NAME と AGE の値を 1 行目に書き込んでから、ポインターを 2 行目に移動させ、列 3 と 4 に変数 ID の値を書き込みます。

```
put name age / id 3-4;
```

例 “例 3: ポインターを新しいページに移動させる” (308 ページ)

OVERPRINT

キーワード OVERPRINT の後ろに指定した値を、直前に書き込んだ出力行に重ねて出力します。

要件 この出力はファイルに送信する必要があります。FILE ステートメントの N=オプションを 1 に設定し、PUT ステートメントの出力先をファイルに設定してください。

ヒント OVERPRINT は、画面に書き込まれる行には影響しません。

テキストを重ねて出力する場合、列ポインターコントロールと行ポインターコントロールと組み合わせて OVERPRINT を使用します。

例 次のステートメントでは列 15 からアンダースコアを重ねて出力し、タイトルに下線を表示します。

```
put @15 'Report Title' overprint
    @15 '_____';
```

例 “例 4: テキストに下線を引く” (309 ページ)

BLANKPAGE

ポインターを新しいページの最初の行に移動させます。ポインターが新しいページの最初の行の最初の列にある場合でも移動を実行します。

ヒント 現在の出力ファイルにキャリッジコントロール文字が含まれている場合、_BLANKPAGE_は適切なキャリッジコントロール文字を含む出力行を生成します。

例 “例 3: ポインターを新しいページに移動させる” (308 ページ)

PAGE

ポインターを新しいページの最初の行に移動させます。行が現在の PAGESIZE= に指定された値を超えると、自動的に新しいページが開始されます。

ヒント 現在の出力ファイルを書き込む場合に、_PAGE_オプションは適切なキャリッジコントロール文字を含む出力行を生成します。_PAGE_オプションは、書き込まれていないファイルには影響しません。

対話型 SAS セッションで FILE PRINT を指定すると、出力ウィンドウではフォームフィードコントロール文字が改ページと解釈されるので、その文字は出力から削除されます。結果として生成されるファイルは、改ページ文字を含まないフラットファイルになります。フォームフィード文字をファイル

に含める必要がある場合、FILE ステートメントに物理ファイルの場所と PRINT オプションを指定する必要があります。

例 “例 3: ポインターを新しいページに移動させる” (308 ページ)

引数: ログ出力コントロール

通常のログ出力を生成する際に SAS が使用するのと同じ色と形式で表示される、カスタム **ERROR**、**NOTE**、および **WARNING** の各ログメッセージを生成できます。これを行うには、キーワード(**ERROR**、**WARNING**、または **NOTE**)をすべて大文字で指定し、その後にコロン(:)またはハイフン(-)を続けます。

```
data L;
  put 'ERROR: This is my error message in red text';
  put 'NOTE: This is my note in blue text';
  put 'WARNING: This is my warning in green text';
run;
```

```
ERROR: This is my error message in red text
NOTE: This is my note in blue text
WARNING: This is my warning in green text
```

キーワードとハイフン-コロンのペアは、PUT ステートメントで直接指定することも、文字変数代入ステートメントで引用符で囲まれた文字列値の一部として指定することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

注: この機能は **PUTLOG** ステートメントと **%PUT** マクロステートメントで使用できます。

'ERROR: <text>'

SAS ログに赤いテキストで表示されるエラーメッセージを指定します。

```
data L;
  put 'ERROR: This is my error message in red text';
run;
```

```
ERROR: This is my error message in red text;
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード **ERROR** は、すべて大文字で、直後にコロン(:)を付ける必要があります。

'NOTE: <text>'

NOTE メッセージを指定するもので、SAS ログに青字で表示されます。

```
data L;
  put 'NOTE: This is my note in blue text';
run;
```

```
NOTE: This is my note message in blue text;
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード NOTE は、すべて大文字で、直後にコロン(:)を付ける必要があります。

'WARNING: <text>'

警告メッセージを指定するもので、SAS ログに緑字で表示されます。

```
data L;
  put 'WARNING: This is my warning in green text';
run;
```

```
WARNING: This is my warning message in green text;
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード WARNING は、すべて大文字で、直後にコロン(:)を付ける必要があります。

'ERROR- <text>'

SAS ログに赤字で表示されるエラーメッセージを指定します。ハイフンを入れると、キーワード ERROR が省略され、テキストがインデントされます。

PUT ステートメントでは、キーワード ERROR をすべて大文字で指定し、その直後にハイフン(-)、そしてメッセージを指定します。

```
data L;
  put 'ERROR: This is my error message in red text';
  put 'ERROR- This is the second line of my error message';
run;
```

```
ERROR:   This is my error message in red text
        This is the second line of my error message
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード ERROR は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

'NOTE- <text>'

NOTE メッセージを指定するもので、SAS ログに青字で表示されます。ハイフンを入れると、キーワード NOTE が省略され、テキストがインデントされます。

単一引用符または二重引用符で囲み、キーワード NOTE をすべて大文字で指定し、その直後にハイフンを続けます(-)そしてメッセージを指定します。

```
data L;
  put 'NOTE: This is my note in blue text';
  put 'NOTE- This is the second line of my note';
run;
```

```
NOTE:   This is my note message in blue text
        This is the second line of my note message
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード NOTE は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

'WARNING- <text>'

警告メッセージを指定するもので、SAS ログに緑字で表示されます。ハイフンを入れると、キーワード WARNING が省略され、テキストがインデントされます。

単一引用符または二重引用符で囲み、キーワード WARNING をすべて大文字で指定し、その直後にハイフン(-)、そしてメッセージを指定します。

```
data L;
  put 'WARNING: This is my warning message in green text';
  put 'WARNING- This is the second line of my warning message';
run;
```

```
WARNING:   This is my warning message in green text
          This is the second line of my warning message
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード WARNING は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

詳細

PUT を使用する必要がある場合

SAS ログ、SAS 出力ウィンドウ、または外部の場所に行を出力する場合は、PUT ステートメントを使用します。DATA ステップのの反復時、PUT ステートメントの前に FILE ステートメントを実行しない場合、行は SAS ログに書き込まれます。FILE ステートメントで PRINT オプションを指定すると、行は SAS 出力ウィンドウに書き込まれます。

PUT ステートメントでは、変数の値、文字列、16 進数の文字定数を含む行を書き込むことができます。PUT ステートメントの指定を使用して、書き込み対象、書き込み先、および書き込み時の出力形式を指定できます。

出力スタイル

出力スタイルの概要

PUT ステートメントを使用して変数の値を書き込む場合、次の 4 つの方法を使用します。

- 列
- リスト(単純および修飾)
- フォーマット
- 名前付き

1 つの PUT ステートメントに、行の出力に合わせて、利用可能な出力スタイルの一部またはすべてを使用できます。

列出力

列出力を使用する場合、PUT ステートメントで、値の位置を示す列番号を変数の後ろに指定します。これらの番号は、次の値 put name 6-15 age 17-19;を書き込む行内の位置を示しています。

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。

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

```
-----1-----2-----+
Peterson  21
Morgan    17
```

PUT ステートメントによって、指定した列に変数 NAME と AGE の値が書き込まれます。詳細については、「[PUT ステートメント: 列](#)」(313 ページ)を参照してください。

リスト出力

*リスト出力*を使用する場合、書き込む順序で PUT ステートメントに変数と文字列を指定します。例えば、この PUT ステートメントによって、NAME と AGE の値は SAS ログに書き込まれます。

```
put name age;
```

次に SAS ログを示します。

```
----+----1-----2----+
Peterson 21
Morgan 17
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

詳細については、“[PUT ステートメント: リスト](#)” (321 ページ)を参照してください。

フォーマット出力

*フォーマット出力*を使用する場合、変数名の後ろに SAS 出力形式またはユーザー定義の出力形式を指定します。この出力形式によって、変数の値を書き込む方法が指示されます。出力形式を指定すると、パック 10 進などの非標準形式のデータや、カンマなどの特殊文字を含む数値を書き込むことができます。-L、-C、-R を使用すると、フォーマット出力のデフォルトの配置より優先されます。

例えば、この PUT ステートメントによって、NAME、AGE、および DATE の値は SAS ログに書き込まれます。

```
put name $char10. age 2. +1 date mmddyy10.;
```

次に SAS ログを示します。

```
----+----1-----2-----+
Peterson 21 07/18/1999
Morgan 17 11/12/1999
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

ポインターコントロールを+1 と指定すると、変数 AGE と DATE の値の間に空白が 1 つ挿入されます。詳細については、“[PUT ステートメント: フォーマット](#)” (316 ページ)を参照してください。

名前付き出力

*名前付き出力*を使用する場合、変数名の前に等号を付けてリストします。例えば、この PUT ステートメントによって、NAME と AGE の値は SAS ログに書き込まれます。

```
put name= age=;
```

次に SAS ログを示します。

```

----+----1-----+----2----+
name=Peterson age=21
name=Morgan age=17

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

詳細については、“[PUT ステートメント: 名前付き](#)” (326 ページ)を参照してください。

1 つの PUT ステートメントで複数の出力スタイルを使用する

1 つの PUT ステートメントに、異なる出力スタイルの一部またはすべてを組み合わせ使用することができます。

```

put name 'on ' date mmdyy8. ' weighs '
startwght +(-1) ' ' idno= 40-45;

```

SAS ログへの行の書き込みについては、“[例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する](#)” (305 ページ)を参照してください。

異なる出力スタイルを組み合わせ使用する場合、それぞれの値を書き込んだ後の出力ポインターの位置を理解することが重要です。ポインターの位置の詳細については、“[値を書き込んだ後のポインターの位置](#)” (303 ページ)を参照してください。

文字定数と変数の書き込み時によく発生するエラーを回避する

PUT ステートメントを使用して文字定数の後ろに変数名を書き込む場合、終了引用符と変数名の間に常に空白を 1 つ追加します。

```

put 'Player:' name1 'Player:' name2 'Player:' name3;

```

空白を追加しないと、次の表に示すように、変数名が後ろに続く文字定数は特殊な SAS 定数と解釈される場合があります。

表 2.4 文字定数の後ろに追加すると正しく解釈されない文字

変数の開始文字	説明	例
b	ビットテスト定数	'00100000'b
d	日付定数	'01jan04'd
dt	日時定数	'18jan2003:9:27:05am'dt
n	名前リテラル	'My Table'n
t	時間定数	'9:25:19pm't
x	16 進法表記	'534153'x

“[例 7: 変数が後ろに続く文字定数の書き込み時によく発生するエラーを回避する](#)” (311 ページ)では、文字定数の後ろに変数を使用する方法について説明しています。

SAS の名前リテラルや式の中の SAS 定数についての詳細は、[SAS プログラマガイド: エssenシャル](#)を参照してください。

ポインターコントロール

PUT ステートメントの値を書き込むときは、ポインターを使用して書き込み位置を制御します。PUT ステートメントには、ポインターの動きを制御する 3 つの方法があります。

列ポインターコントロール

PUT ステートメントで出力行への値の書き込みを開始するときに、ポインターの列の位置をリセットします。

行ポインターコントロール

PUT ステートメントで出力行に書き込むときに、ポインターの行の位置をリセットします。

ラインホールド指定子

出力バッファーに行が保持されるので、他の PUT ステートメントでも同じ行に書き込むことができます。デフォルトでは、PUT ステートメントを実行すると、以前の行を解放して、新しい行に書き込みます。

列ポインターコントロールや行ポインターコントロールを使用すると、ポインターを移動させる行または列の絶対位置を指定したり、現在のポインターの位置の相対位置として移動先の列または行を指定できます。PUT ステートメントで指定できるすべてのポインターコントロールをこの表に示します。

表 2.5 PUT ステートメントで指定できるポインターコントロール

ポインターコントロール	相対指定	絶対指定
列ポインターコントロール	$+n$	$@n$
	$+numeric-variable$	$@numeric-variable$
	$+(expression)$	$@(expression)$
行ポインターコントロール	$/$, $_PAGE_$, $_BLANKPAGE_$	$\#n$ $\#numeric-variable$ $\#(expression)$
	OVERPRINT	なし
ラインホールド指定子	$@$	(なし)
	$@@$	(なし)

注: ポインターコントロールは、常に適用する変数の前に指定するようにしてください。

ポインター位置を指定する方法の詳細については、“[値を書き込んだ後のポインターの位置](#)” (303 ページ)を参照してください。

ラインホールド指定子の使用

ラインホールド指定子を使用して、次のいずれかの状況が発生した場合にポインターを現在の出力行に保持します。

- 複数の PUT ステートメントを使用して値を同じ出力行に書き込む場合
- 1 つの PUT ステートメントを使用して複数のオブザベーションの値を同じ出力行に書き込む場合

ラインホールド指定子を指定しない場合、DATA ステップの PUT ステートメントごとに値が新しい出力行に書き込まれます。

PUT ステートメントでは、後置@と後置@@は同じ働きをします。INPUT ステートメントとは異なり、PUT ステートメントでは、DATA ステップの新しい繰り返しの開始時に、後置@で保持されている行は自動的に解放されません。後置@または後置@@で保持されている現在の出力行は、次のいずれかの状況が発生した場合に解放されます。

- PUT ステートメントに後置@が指定されていない場合
- PUT ステートメントに `_BLANKPAGE_` または `_PAGE_` が指定されている場合
- 現在の行の行端に達した場合(FILE ステートメントの `LRECL=` オプションまたは `LINESIZE=` オプションの現在の値、または指定されている場合には `LINESIZE=` システムオプションの現在の値により決定)
- DATA ステップで最後の繰り返しが終了した場合

後置@または後置@@を使用すると、次の PUT ステートメントの実行時にポインターの値が変更されないで、現在の行の長さを超えても値を書き込もうとする場合があります。参照: “[ポインターが行の終わりを超える場合](#)” (304 ページ)。

値を書き込んだ後のポインターの位置

特に 1 つの PUT ステートメントで複数の出力スタイルを組み合わせる場合は、値を書き込んだ後の出力ポインターの位置を理解しておくことが重要になります。値を書き込んだ後のポインターの位置は、指定した出力スタイルや、書き込んだ値が文字列か変数かによって異なります。列出力またはフォーマット出力を使用すると、ポインターの位置は PUT ステートメントに指定したフィールドの最後から 1 番目の列になります。この 2 つのスタイルでは、変数の値のみが書き込まれます。

リスト出力と名前付き出力を使用すると、PUT ステートメントでは、それぞれの値を書き込んだ後に列を 1 つ自動的にスキップするので、ポインターの位置は変数の値の最後から 2 番目の列になります。ただし、PUT ステートメントでリスト出力を使用して文字列を書き込む場合、ポインターの位置は文字列の最後から 1 番目の列になります。文字列を書き込んだ後に行ポインターコントロールや列出力を使用しない場合、文字列と次の値を区切るため、文字列の最後に空白を 1 つ追加してください。

`_INFILE_` の指定時、ポインターは現在の入力ファイルのレコードを書き込んだ後に列 1 に配置されます。

出力ポインターがページの左上にあるとき、次のいずれかのステートメントを使ってポインターを制御することができます。

- `PUT _BLANKPAGE_` がブランクページを書き込み、ポインターを次のページの先頭に移動させた場合
- `PUT _PAGE_` が同じ位置にポインターを保持する場合

`FILE` ステートメントの `COLUMN=` オプションまたは `LINE=` オプションに指定した変数の値を調べて、現在のポインターの位置を特定できます。

ポインターが行の終わりを越える場合

現在指定されている出力行の長さを超える場合、出力行は書き込まれません。現在の出力ファイルの行の長さは次のオプションの値で決定されます。

- 現在の `FILE` ステートメントの `LINESIZE=` オプションの値
- `LINESIZE=` システムオプションの値(SAS 出力ウィンドウの場合)。
- 現在の `FILE` ステートメントの `LRECL=` オプションの値(外部ファイルの場合)

次に示す指定を 1 つまたは複数使用すると、ポインターの位置が誤って現在の行の長さを超えてしまう場合があります。

- 現在の行の長さを超えた列にポインターを移動させるように `+ポインターコントロール` の値が指定されている場合
- 列範囲が現在の行の長さを超えている場合(たとえば、現在の行の長さが 80 の場合に `PUT X 90 - 100` と指定する場合)
- 変数の値または文字列が現在の出力行のスペースよりも長い場合

デフォルトでは、`PUT` ステートメントが現在の行の終わりを越えて書き込む必要がある場合、現在の行に収まらないアイテムの書き込みは行わずに、現在の行の書き込みを終了します。次に、現在の行に書き込めなかったアイテムを新しい行の列 1 から書き込みます。[“FILE ステートメント”\(104 ページ\)](#)の `FLOWOVER`、`DROPOVER`、`STOPOVER` の各オプションを参照してください。

配列

`PUT` ステートメントを使用して配列要素を書き込むことができます。`subscript` には、`PUT` ステートメントの実行時に結果が整数になる SAS 式を指定できます。次に示すように、配列参照を丸かっこで囲むと、ポインターコントロール付きの *numeric-variable* の構文に配列参照を使用できます。

- `@(array-name{i})`
- `+(array-name{i})`
- `#{array-name{i})`

事前に定義した配列のすべての要素を外部の場所書き込むには、配列の `subscript` にアスタリスク(*)を使用します。1 次元配列または多次元配列は使用できますが、`_TEMPORARY_` 配列は使用できません。`subscript` を中かっこ、大かっこ、丸かっこで囲みます。次に、リスト出力、フォーマット出力、列出力、名前付き出力を使用

して配列を書き込みます。リスト出力を使用する場合、ステートメントの形式は次のようになります。PUT *array-name*{*};

フォーマット出力を使用する場合、ステートメントの形式は次のようになります。PUT *array-name*{*}{*format*|*format.list*}

丸かっこで囲んだ出力形式は、配列参照の後ろに続けて指定します。

16 進表記の数値定数

16 進表記の数値定数は、数字(通常は 0)で始まります。さらに 16 進文字を続けることができ、文字 X で終わります。この定数には、最大 16 個の有効な 16 進文字 (0 から F) を含めることができます。

比較

- PUT ステートメントでは変数の値や文字列を SAS ログや外部の場所へ書き込みますが、INPUT ステートメントでは外部ファイルの生データやインストリームで入力されるデータ行を読み込みます。
- INPUT ステートメントと PUT ステートメントのどちらも、後置@や後置@@ラインホールド指定子を使用すると、入力バッファや出力バッファ内に現在の行が保持されます。INPUT ステートメントでは、後置@@を使用して入力バッファに行を保持し、DATA ステップの繰り返し時にそのまま使用します。PUT ステートメントでは、後置@は後置@@と同じ動作になります。どちらを使用しても、DATA ステップの反復間を通して行を保持します。
- PUT ステートメントと OUTPUT ステートメントはどちらも DATA ステップで出力を生成します。PUT ステートメントは出力バッファを使用し、外部の場所、SAS ログ、またはモニターに出力を書き込みます。OUTPUT ステートメントは、プログラムデータベクトルを使用し、SAS データセットにオブザベーションを書き込みます。

例

例 1: 1 つの PUT ステートメントで複数の出力スタイルを使用する

この例では、1 つの PUT ステートメントで複数の出力スタイルを使用します。

```
data club1;
  input idno name $ startwght date : date7.;
  put name 'on ' date mmdyy8. ' weighs '
      startwght +(-1) ' ' idno= 32-40;
  datalines;
032 David 180 25nov99
049 Amelia 145 25nov99
219 Alan 210 12nov99
;
```

この例の各変数に指定されている出力スタイルを次の表に示します。

変数	出力スタイル
NAME, STARTWGHT	リスト出力
DATE	フォーマット出力
IDNO	名前付き出力

この PUT ステートメントではポインターコントロールも使用しています。また、文字列と変数名も指定しています。

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。

```

-----1-----2-----3-----4
David on 11/25/99 weighs 180. idno=1032
Amelia on 11/25/99 weighs 145. idno=1049
Alan on 11/12/99 weighs 210. idno=1219

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

文字列の先頭と末尾に空白が挿入されているので、ポインターの位置が移動します。この空白によって、変数の値と文字列が区切られています。ポインターコントロールを+(-1)と指定しているので、ポインターが後方(左)に移動し、STARTWGHT の値とピリオドの間の不要な空白が削除されます。ポインター位置を指定する方法の詳細については、“[値を書き込んだ後のポインターの位置](#)” (303 ページ)を参照してください。

例 2: ポインターをページ内で移動させる

次の PUT ステートメントは、列ポインターコントロールと行ポインターコントロールを使用して出力ポインターの位置を指定する方法を示しています。

- ポインターを特定の列に移動させるには、@の後ろに列番号か、列番号の値を示す変数や式を指定します。たとえば、次のステートメントではポインターを列 15 に移動してから、リスト出力を使用して変数 TOTAL SALES の値を書き込みます。

```
put @15 totalsales;
```

次の PUT ステートメントでは、ポインターを変数 COLUMN に指定した値に移動した後、COMMA6 出力形式を使用して変数 TOTAL SALES の値を書き込みます。

```

data _null_;
  set carsales;
  column=15;
  put @column totalsales comma6.;
run;

```

- このプログラムでは、ポインターを戻す 2 つの方法を示しています。

```

data carsales;
  input item $10. jan : comma5.
        feb : comma5. mar : comma5.;
  saleqtr1=sum(jan,feb,mar);

```

```

/* an expression moves pointer backward */
put '1st qtr sales for ' item
   'is ' saleqtr1 : comma6. +(-1) ' ';
/* a numeric variable with a negative
   value moves pointer backward.    */
x=-1;
put '1st qtr sales for ' item
   'is ' saleqtr1 : comma5. +x ' ';
datalines;
trucks    1,382  2,789  3,556
vans      1,265  2,543  3,987
sedans    2,391  3,011  3,658
;

```

SALEQTR1 の値は修飾リスト出力を使用して書き込まれるので、ポインターは自動的に 2 列先に移動します。詳細については、“[修飾リスト出力とフォーマット出力の違い](#)” (324 ページ) を参照してください。値とピリオドの間の不要なブランクを削除するために、ポインターを 1 列だけ戻します。

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。

```

-----1-----2-----3-----4
st qtr sales for trucks is 7,727.
st qtr sales for trucks is 7,727.
st qtr sales for vans is 7,795.
st qtr sales for vans is 7,795.
st qtr sales for sedans is 9,060.
st qtr sales for sedans is 9,060.

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

- 次のプログラムでは、PUT ステートメントと行ポインターコントロール(/)を使用して、次行にポインターを移動させます。

```

data _null_;
  set carsales end=lastrec;
  totalsales+saleqtr1;
  if lastrec then
    put @2 'Total Sales for 1st Qtr'
      / totalsales 10-15;
run;

```

この DATA ステップでは、CARSALES データセット内のすべてのオブザベーションを使用して TOTALSALES の値を算出した後、PUT ステートメントを実行します。列 2 から文字列を書き込み、次の行に移動してから列 10 から 15 に TOTALSALES の値を書き込みます。

```

-----1-----2-----3
Total Sales for 1st Qtr
      24582

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

例 3: ポインターを新しいページに移動させる

この例では STATEPOP というデータセットを作成します。このデータセットには、1990 年の米国国勢調査の都市圏と非都市圏の人口に関する情報が含まれています。ここでは、FORMAT プロシジャを実行し、50 州とコロンビア特別区を 4 つの地域にグループ化しています。次に、IF ステートメントと PUT ステートメントを使用して出力を制御しています。

```

title1;
data statepop;
  input state $ cityp90 ncityp90 region @@;
  label cityp90= '1990 metropolitan population
            (million)'
        ncityp90='1990 nonmetropolitan population
            (million)'
        region= 'Geographic region';
  datalines;
ME .443 .785 1 NH .659 .450 1
VT .152 .411 1 MA 5.788 .229 1
RI .938 .065 1 CT 3.148 .140 1
NY 16.515 1.475 1 NJ 7.730 .A 1
PA 10.083 1.799 1 DE .553 .113 2
MD 4.439 .343 2 DC .607 . 2
VA 4.773 1.414 2 WV .748 1.045 2
NC 4.376 2.253 2 SC 2.423 1.064 2
GA 4.352 2.127 2 FL 12.023 .915 2
KY 1.780 1.906 2 TN 3.298 1.579 2
AL 2.710 1.331 2 MS .776 1.798 2
AR 1.040 1.311 2 LA 3.160 1.060 2
OK 1.870 1.276 2 TX 14.166 2.821 2
OH 8.826 2.021 3 IN 3.962 1.582 3
IL 9.574 1.857 3 MI 7.698 1.598 3
WI 3.331 1.561 3 MN 3.011 1.364 3
IA 1.200 1.577 3 MO 3.491 1.626 3
ND .257 .381 3 SD .221 .475 3
NE .787 .791 3 KS 1.333 1.145 3
MT .191 .608 4 ID .296 .711 4
WY .134 .319 4 CO 2.686 .608 4
NM .842 .673 4 AZ 3.106 .559 4
UT 1.336 .387 4 NV 1.014 .183 4
WA 4.036 .830 4 OR 1.985 .858 4
CA 28.799 .961 4 AK .226 .324 4
HI .836 .272 4
;
proc format;
  value regfmt 1='Northeast'
              2='South'
              3='Midwest'
              4='West';
run;
data _null_;
  set statepop;
  by region;
  pop90=sum(cityp90,ncityp90);
  file print;
  put state 1-2 @5 pop90 7.3 ' million';
  if first.region then

```

```

        regioncitypop=0; /* new region */
regioncitypop+cityp90;
if last.region then
do;
    put // '1990 US CENSUS for ' region regfmt.
        / 'Total Urban Population: '
        regioncitypop' million' _page_;
end;
run;

```

アウトプット 2.35 PUT ステートメントを使用した北東地域の出力

```

ME      1.228 million
NH      1.109 million
VT      0.563 million
MA      6.017 million
RI      1.003 million
CT      3.288 million
NY      17.990 million
NJ      7.730 million
PA      11.882 million
1990 US CENSUS for Northeast
Total Urban Population: 45.456 million
1

```

LAST.REGION の値が 1 の場合、PUT _PAGE_ はポインターを新しいページの 1 行に移動させます。例では、ページから移動の前にフッターメッセージが表示されます。

例 4: テキストに下線を引く

この例では、OVERPRINT を使用し、直前の PUT ステートメントで書き込んだ値の下に下線を表示します。

```

data _null_;
input idno name $ startwght;
file file-specification print;
put name 1-10 @15 startwght 3.;
if startwght > 200 then
    put overprint @15 '___';
datalines;
032 David 180
049 Amelia 145
219 Alan 210
;

```

2 番目の PUT ステートメントでは、最初の PUT ステートメントで書き込んだ体重の値が 200 ポンドを超えていた場合に下線を引きます。

次の PUT ステートメントでは、列ポインターコントロールと行ポインターコントロールとともに OVERPRINT を使用しています。

```

put @5 name $8. overprint @5 8*'_ '
/ @20 address;

```

この PUT ステートメントでは、変数 NAME の値を書き込んでから、8 個のアンダースコアを重ねて下線を引きます。その後、出力ポインターを次の行に移動させ、変数 ADDRESS の値を書き込みます。

例 5: 出力行の保持と解放

次の DATA ステップでは、PUT ステートメントで出力行の保持と解放を行う方法を示します。

```
data _null_;
  input idno name $ startwght 3.;
  put name @;
  if startwght ne . then
    put @15 startwght;
  else put;
  datalines;
032 David 180
049 Amelia 145
126 Monica
219 Alan 210
;
```

この例の PUT ステートメントは、これらの作業を行います。

- 最初の PUT ステートメントに後置@を指定し、変数 NAME の値を書き込んだ後、現在の出力行を保持します。
- IF-THEN ステートメントの条件が真になる場合、2 番目の PUT ステートメントで STARTWGHT の値を書き込んでから、現在の出力行を解放します。
- 条件が偽になる場合、2 番目の PUT ステートメントは実行されません。かわりに、ELSE PUT ステートメントが実行されます。ELSE PUT ステートメントで出力行を解放した後、出力ポインターを出力バッファの列 1 に移動します。

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。

```
----+----1-----2
David      180
Amelia     145
Monica
Alan       210
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

例 6: 現在の入力レコードを SAS ログに書き込む

ID の値が 1000 未満の場合、PUT _INFILE_を実行し、現在の入力レコードを SAS ログに書き込みます。DELETE ステートメントを実行することによって、DATA ステップで TEAM データセットにオブザベーションが書き込まれないようにします。

```
data team;
  input id team $ score1 score2;
  if id le 1000 then
  do;
    put _infile_;
    delete;
  end;
  datalines;
032 red 180 165
049 yellow 145 124
```



```
219 red 210 192
;
```

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。

```
----+----1----+----2
219 red 210 192
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

例 7: 変数が後ろに続く文字定数の書き込み時によく発生するエラーを回避する

この例では、SAS 名前リテラルとして誤って解釈されないように、PUT ステートメントを使用して文字リテラルと変数の値を書き込む方法を示します。SAS 名前リテラルとは、引用符で囲んだ文字列として表される名前トークンで、この文字列の後ろに文字 `n` が追加されます。SAS 名前リテラルの詳細については、*SAS プログラマガイド: エッセンシャル*を参照してください。

次のプログラムでは、PUT ステートメントは変数 `NVAR1` の値が後ろに続く定数 `'n'` を書き込んでから、別の定数 `'n'` を書き込みます。

```
data _null_;
  n=5;
  nvar1=1;
  var1=7;
  put @1 'n' nvar1 'n';
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。

```
----+----1----+----2
n1 n
```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

前述の PUT ステートメントから定数と変数の間にある空白をすべて削除すると、定数 `'n'` は定数として読み込まれず、名前リテラルと解釈されます。次の変数は、`NVAR1` ではなく `VAR1` として読み込まれます。最後の定数 `'n'` は正しく解釈されます。

```
put @1 'n'nvar1'n';
```

PUT ステートメントにより、この行が SAS ログに書き込まれます。

```
----+----1----+----2
5 7 n
```

文字定数と変数の値の間に空白を追加した後、必要に応じてポインターコントロールを使用すると、誤った解釈を潜在的に発生させることなく、文字定数と変数の値の間に空白を挿入することなく書き込まれます。たとえば、次の PUT ステートメントではポインターコントロールを使用して正しい文字定数と変数の値を書

き込みますが、空白は挿入されません。+(-1)と指定すると、PUT ステートメントでポインターが 1 列だけ戻ります。

```
put @1 'n' nvar1 +(-1) 'n';
```

PUT ステートメントにより、この行が SAS ログに書き込まれます。

```
----+----1-----2
n1n
```

例 8: 複数の列で構成される出力を作成する

この例では、#n 行ポインターコントロールと@n 列ポインターコントロールを使用して複数の列で構成される出力を作成します。

```
/*
 * Use #i and @j to position name and weight information into
 * four columns in column-major order. That is print down column 1
 * first, then print down column 2, etc.
 * This example highlights the need to specify # before @ because
 * # sets the column pointer to 1.
 */
data _null_;
  file print n=ps notitles header=hd;

  do i = 1 to 80 by 20;
    do j = 1 to ceil(num_students/4);
      set sashelp.class nobs=num_students;
      put #(j+3) @i name $8. ' ' +1 weight 5.1;
    end;
  end;
  stop;

  hd:
  put @26 'Student Weight in Pounds' / @26 24*' ';
  return;
run;
```

このプログラムを実行すると、次の出力が作成されます。

Student Weight in Pounds			

Alfred - 112.5	James - 83.0	Joyce - 50.5	Robert - 128.0
Alice - 84.0	Jane - 84.5	Judy - 90.0	Ronald - 133.0
Barbara - 98.0	Janet - 112.5	Louise - 77.0	Thomas - 85.0
Carol - 102.5	Jeffrey - 84.0	Mary - 112.0	William - 112.0
Henry - 102.5	John - 99.5	Philip - 150.0	

関連項目

ステートメント:

- [“FILE ステートメント”\(104 ページ\)](#)

- [“PUT ステートメント: 列” \(313 ページ\)](#)
- [“PUT ステートメント: フォーマット” \(316 ページ\)](#)
- [“PUT ステートメント: リスト” \(321 ページ\)](#)
- [“PUT ステートメント: 名前付き” \(326 ページ\)](#)

システムオプション:

- [“LINESIZE= システムオプション” \(SAS システムオプション: リファレンス\)](#)
- [“PAGESIZE= システムオプション” \(SAS システムオプション: リファレンス\)](#)

PUT ステートメント: 列

変数の値を出力行の指定した列に書き込みます。

該当要素:	DATA ステップ
カテゴリ:	CAS ファイル操作
種類:	実行
制限事項:	このステートメントは、In-database DATA ステップではサポートされていません。 詳細については、 “DATA Step Acceleration” (SAS In-Database Products: User's Guide) を参照してください。

構文

```
PUT variable start-column <- end-column>
<.decimal-places> <@ | @@>;
```

引数

variable

変数の名前を指定します。指定した変数の値が書き込まれます。

start-column

出力行での、値を書き込むフィールドの開始列を指定します。

-end-column

この値のフィールドの最終列を指定します。

ヒント 出力行に必要な列数が 1 つの場合、*end-column* の指定を省略することができます。

例 *end-column* が省略されているため、文字変数 GENDER の値は列 16 にのみ存在します。

```
put name 1-10 gender 16;
```

.decimal-places

数値で使用する小数点以下の桁数を指定します。

範囲 正の整数

ヒント *d* の値に 0 を指定したり、*d* の値を省略する場合は、小数点なしの値が書き込まれます。

例 [“例 2: PUT ステートメントで列出力の使用” \(315 ページ\)](#)

@ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、後置@および後置@@と呼ばれます。

要件 後置@または後置@@は、PUT ステートメントの最後のアイテムとして指定する必要があります。

参照項目: [“ラインホールド指定子の使用” \(303 ページ\)](#)

詳細

列出力では、列番号は出力行内で各変数値が占める位置を示します。指定した列数より少ない列数で値を出力できる場合、文字変数は左寄せで、数値変数は右寄せで表示されます。

1 つの PUT ステートメントでの列の指定数に制限はありません。出力行のどの位置でも値を書き込むことができます。また、同じステートメントですでに値を書き込んだ列の上に値を上書きすることもできます。1 つの PUT ステートメントで、列の出力を他の出力スタイルと組み合わせられます。詳細については、[“1 つの PUT ステートメントで複数の出力スタイルを使用する” \(301 ページ\)](#)を参照してください。

例

例 1: PUT ステートメントを使用した CAS DATA ステップでの列出力の使用

この例では、最初の DATA ステップが CAS で実行され、PUT ステートメントを使用して変数 name と sex の値を SAS ログに書き込みます。2 番目の DATA ステップは CAS で実行され、@引数を使用して同じ出力をログに書き込みますが、sex の値は両端揃えで変数 name から 15 スペースあけてあります。

```
cas casauto;
libname casuser cas;

data casuser.class;
  set sashelp.class;
  put name sex;
run;
```

```
data _null_;
  set casuser.class;
  put name 1-4 @15 sex;
run;
```

```
Alfred M
Alice F
Barbara F
Carol F
Henry M
James M
Jane F
Janet F
Jeffrey M
John M
Joyce F
Judy F
Louise F
Mary F
Philip M
Robert M
Ronald M
Thomas M
William M
```

```
Alfr      M
Alic      F
Barb      F
Caro      F
Henr      M
Jame      M
Jane      F
Jeff      M
John      M
Joyc      F
Judy      F
Loui      F
Mary      F
Phil      M
Robe      M
Rona      M
Thom      M
Will      M
```

例 2: PUT ステートメントで列出力の使用

PUT ステートメントで列出力を使用する例を次に示します。

- この PUT ステートメントは列出力を使用します。

```
data _null_;
  input name $ 1-18 score1 score2 score3;
  put name 1-20 score1 23-25 score2 28-30
    score3 33-35;
  datalines;
Joseph      11  32  76
Mitchel     13  29  82
Sue Ellen   14  27  74
;
```

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。

```

-----1-----2-----3-----4
Joseph      11  32  76
Mitchel     13  29  82
Sue Ellen   14  27  74

```

注: 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

文字変数 NAME の値は列 1 から出力されます。また、指定したフィールド(列 1 から 20)に左寄せで表示されます。数値変数 SCORE1 から SCORE3 の値は、各フィールドに右寄せで表示されます。

- 次のステートメントでも同じ出力行が生成されますが、SCORE1 の値が最初に書き込まれ、NAME の値が最後に書き込まれます。

```

put score1 23-25 score2 28-30
   score3 33-35 name $ 1-20;

```

- この DATA ステップでは、列出力に小数点以下の桁数を指定します。

```

data _null_;
  x=11;
  y=15;
  put x 10-18.1 y 20-28.1;
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。

```

-----1-----2-----3-----4
          11.0   15.0

```

関連項目

ステートメント:

- [“PUT ステートメント” \(289 ページ\)](#)

PUT ステートメント: フォーマット

指定した出力形式で変数の値を出力行に書き込みます。

該当要素: DATA ステップ

カテゴリ: CAS
ファイル操作

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

PUT <*pointer-control*> *variable format*. <@ | @@>;

PUT <*pointer-control*>(*variable-list*) (*format-list*) <@ | @@>;

引数

pointer-control

指定した行または列に出力ポインターを移動させます。

参照項目: [“引数: 列:ポインターコントロール” \(291 ページ\)](#)

[“引数: 行:ポインターコントロール” \(293 ページ\)](#)

例 [“例 1: フォーマット値の間に文字を書き込む” \(320 ページ\)](#)

variable

変数の名前を指定します。指定した変数の値が書き込まれます。

(*variable-list*)

変数のリストを指定します。指定した変数の値が書き込まれます。

要件 (*format-list*)は(*variable-list*)の後ろに指定する必要があります。

参照項目: [“変数と出力形式をグループ化する方法” \(319 ページ\)](#)

例 [“例 1: フォーマット値の間に文字を書き込む” \(320 ページ\)](#)

format.

変数の値を書き込むときに使用する出力形式を指定します。デフォルトの配置を無効にするには、配置の指定を出力形式に追加します。

-L 値を左揃えにします。

-C 値を中央揃えにします。

-R 値を右揃えにします。

ヒント 出力形式の長さには、値に加えて出力形式の指定に使用するカンマ、ドル記号、小数点、その他の特殊文字を表示するのに十分なスペースを指定してください。

例 次の PUT ステートメントは、出力形式 `dollar7.2` を使用して `X` の値を書き込みます。

```
put x dollar7.2;
```

`X` が 100 の場合、フォーマット値は 7 列を使用します。
\$100.00

例 [“例 2: フォーマット値のデフォルトの配置を無効にする” \(320 ページ\)](#)

(*format-list*)

出力形式のリストを指定します。このリストは、前に指定した変数リストの値を書き込むときに使用します。PUT ステートメントでは、*format-list* に次を指定できます。

format.

変数の値を書き込むときに使用する出力形式を指定します。

ヒント SAS 出力形式またはユーザー定義の出力形式のどちらかを指定できます。

参照項目: [SAS 出力形式と入力形式 リファレンス](#)

pointer-control

値の読み込み位置を指示するポインターコントロール(@、#、/、+、OVERPRINT)のうちの 1 つを指定します。

例 [“例 1: フォーマット値の間に文字を書き込む” \(320 ページ\)](#)

character-string

フォーマット値の間に追加する 1 つまたは複数の文字を指定します。

例 次のステートメントは、フォーマット値 CODE1、CODE2、CODE3 の間にハイフンを追加します。

```
put bldg $ (code1 code2 code3) (3. '-');
```

例 [“例 1: フォーマット値の間に文字を書き込む” \(320 ページ\)](#)

n*

出力形式リストで n*に続けて指定した出力形式を n 回繰り返します。

制限事項 (format-list)は(variable-list)の後ろに指定する必要があります。

参照項目: [“変数と出力形式をグループ化する方法” \(319 ページ\)](#)

例 次のステートメントは、7.2 出力形式を使用して GRADES1、GRADES2、GRADES3 を書き込み、5.2 出力形式を使用して GRADES4、GRADES5 を書き込みます。

```
put (grades1-grades5) (3*7.2, 2*5.2);
```

@ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、後置@および後置@@と呼ばれます。

制限事項 後置@または後置@@は、PUT ステートメントの最後のアイテムとして指定する必要があります。

参照項目: [“ラインホールド指定子の使用” \(303 ページ\)](#)

詳細

フォーマット出力の使用

フォーマット出力では、出力行を指定するために、変数名とその変数の値を書き込むときに使用する出力形式をリストします。SAS 出力形式またはユーザー定義の出力形式を使用して、変数の値の出力形式を制御することができます。SAS 出力形式の詳細については、“[出力形式について](#)” ([SAS 出力形式と入力形式 リファレンス](#))を参照してください。

フォーマット出力では、PUT ステートメントでは、変数の値を書き込むときに使用する出力形式を変数名の後ろに指定します。値の間にブランクは自動的に追加されません。指定した列数より少ない列数で値を出力できる場合、出力形式の幅で指定されたフィールド内に文字の値は左寄せ、数値は右寄せで表示されます。

フォーマット出力とポインターコントロールを組み合わせると、変数の値を書き込む行と列の位置を正確に指定できます。次の PUT ステートメントは、dollar7.2 出力形式を使用して、列 12 の位置から中央揃えで X の値を表示します。

```
put @12 x dollar7.2-c;
```

変数と出力形式をグループ化する方法

特定のパターンで出力行に値を書き込む場合、出力形式リストを使用するとコードを書く時間を短縮することができます。出力形式リストは、出力形式をブランクまたはカンマで区切り、丸かっこで囲みます。出力形式リストは、丸かっこで囲んだ変数名のリストの後に指定する必要があります。

たとえば、次のステートメントは、出力形式リストを使用して SCORE1 から SCORE5 までの 5 つの変数を順に書き込みます。各変数の値の出力に列を 4 つ使用します。また、値の間にブランクは追加しません。

```
put (score1-score5) (4. 4. 4. 4. 4.);
```

前述のステートメントは次のように簡略化できます。

```
put (score1-score5) (4.);
```

出力形式リストには、任意のポインターコントロール(@、#、/、+、OVERPRINT)、n*、および文字列を挿入できます。PUT ステートメントには必要な数の出力形式リストを使用できますが、出力形式リストはネストさせないでください。変数リストにあるすべての値が書き込まれると、PUT ステートメントは出力形式リストに残っている指示を無視します。例については、“[例 3: 必要な数以上の出力形式が指定されている場合](#)” ([320 ページ](#))を参照してください。

配列に含まれるすべての要素への参照も指定できます。出力形式リストの前に、(array-name {*})を指定します。ただし、この場合、_TEMPORARY_配列内の要素を指定することはできません。この PUT ステートメントは、配列名と出力形式リストを指定します。

```
put (array1{*}) (4.);
```

配列の参照方法の詳細については、“[配列](#)” ([304 ページ](#))を参照してください。

例

例 1: フォーマット値の間に文字を書き込む

次の例では、一部の値をフォーマットし、変数 BLDG と ROOM の間に-(ハイフン)を1つ書き込みます。

```
data _null_;
  input name & $15. bldg $ room;
  put name @20 (bldg room) ($1. "-" 3.);
  datalines;
Bill Perkins J 126
Sydney Riley C 219
;
```

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。

```
Bill Perkins    J-126
Sydney Riley    C-219
```

例 2: フォーマット値のデフォルトの配置を無効にする

この例では出力形式に配置方法を指定します。

```
data _null_;
  input name $ 1-12 score1 score2 score3;
  put name $12.-r +3 score1 3. score2 3.
    score3 4.;
  datalines;
Joseph      11 32 76
Mitchel     13 29 82
Sue Ellen   14 27 74
;
```

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。¹

```
----+----1----+----2----+----3----+----4
      Joseph 11 32 76
      Mitchel 13 29 82
      Sue Ellen 14 27 74
```

文字変数 NAME の値はフォーマットフィールドに右寄せで表示されています。(文字変数の場合、デフォルト設定は左寄せです。)

例 3: 必要な数以上の出力形式が指定されている場合

次の出力形式リストには、PUT ステートメントの実行時に必要となる数よりも多い出力形式が指定されています。

```
data _null_;
  input x y z;
  put (x y z) (2.,+1);
  datalines;
2 24 36
0 20 30
```

1. 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

;

この PUT ステートメントでは、出力形式 2.を使用して X の値を書き込みます。列ポインターコントロールは+1 と指定されているため、ポインターを 1 列だけ前方(右)に移動します。次に、出力形式 2.を使用して Y の値を書き込みます。列ポインターコントロールは+1 と指定されているため、再度ポインターを 1 列だけ前方(右)に移動します。最後に、出力形式 2.を使用して Z の値を書き込みます。3 回目の繰り返しでは、PUT ステートメントはポインターコントロールの移動を指示する+1 を無視します。

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。¹

```
----+----1----+
2 24 36
0 20 30
```

関連項目

ステートメント:

- [“PUT ステートメント” \(289 ページ\)](#)

PUT ステートメント: リスト

変数値と指定した文字列を出力行に書き込みます。

該当要素: DATA ステップ

カテゴリ: CAS

ファイル操作

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“[DATA Step Acceleration](#)” (*SAS In-Database Products: User's Guide*)を参照してください。

構文

PUT <pointer-control> variable <@ | @@>;

PUT <pointer-control> <n> 'character-string' <@ | @@>;

PUT <pointer-control> variable <: | ~> format.<@ | @@>;

1. 理解しやすいようにルーラー付きの行が表示されていますが、PUT ステートメントではこの行は生成されません。

引数

pointer-control

指定した行または列に出力ポインターを移動させます。

参照項目: [“引数: 列:ポインターコントロール” \(291 ページ\)](#)

[“引数: 行:ポインターコントロール” \(293 ページ\)](#)

例 [“例 2: 文字列と変数値の書き込み” \(325 ページ\)](#)

variable

変数の名前を指定します。指定した変数の値が書き込まれます。

例 [“例 1: リスト出力の値の書き込み” \(324 ページ\)](#)

*n**

後ろに指定する文字列を n 回繰り返します。

例 次のステートメントでは、出力行にアンダースコアを 132 回書き込みます。

```
put 132*'_';
```

'character-string'

書き込む文字列を引用符で囲んで指定します。

操作 指定した文字列全体を書き込むスペースが現在の行に残されていない場合、指定した文字列を保留して、現在の行を書き込みます。次に、新しい行の列 1 からこの保留した文字列を書き込みます。詳細については、[“ポインターが行の終わりを超える場合” \(304 ページ\)](#)を参照してください。

ヒント 誤って解釈されるのを防ぐため、PUT ステートメントで使用する終了引用符の後ろには常にブランクを追加してください。

引用符の後ろに X がある場合、指定した文字列が 16 進定数であると判断されます。

間にブランクを含まない連続した 2 つの単一引用符(')または二重引用符(“)をテキスト文字列として指定すると、単一引用符(')または二重引用符(“)がそれぞれ書き出されます。

参照 [“リスト出力のスペース設定方法” \(324 ページ\)](#)

項目:

例 [“例 2: 文字列と変数値の書き込み” \(325 ページ\)](#)

:

PUT ステートメントで変数値を書き込むときに使用する出力形式を指定できます。値の先頭または末尾にあるブランクは削除されます。また、値の後ろにブランクが 1 つ追加されます。

参照項目: [“修飾リスト出力とフォーマット出力の違い” \(324 ページ\)](#)

例 [“例 3: 修飾リスト出力\(:\)を用いた値の書き込み” \(325 ページ\)](#)

~

PUT ステートメントで変数値を書き込むときに使用する出力形式を指定できます。フォーマット値に区切り文字が含まれていない場合でも、フォーマット値を引用符で囲んで表示します。値の先頭または末尾にあるブランクは削除されます。また、値の後ろにブランクが1つ追加されます。デフォルトでは、文字変数の欠損値はブランク(" ")として書き込まれ、数値の欠損値はピリオド(".")として書き込まれます。

要件 FILE ステートメントに DSD オプションを指定する必要があります。

例 [“例 4: 修飾リスト出力\(~\)を用いた値の書き込み” \(326 ページ\)](#)

format.

データの値を書き込むときに使用する出力形式を指定します。

ヒント SAS 出力形式またはユーザー定義の出力形式のどちらかを指定できます。次を参照してください。 [SAS 出力形式と入力形式: リファレンス](#)

例 [“例 3: 修飾リスト出力\(:\)を用いた値の書き込み” \(325 ページ\)](#)

@ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、`後置@`および`後置@@`と呼ばれます。

制限事項 `後置@`または`後置@@`は、PUT ステートメントの最後のアイテムとして指定する必要があります。

参照項目: [“ラインホールド指定子の使用” \(303 ページ\)](#)

詳細

リスト出力の使用

リスト出力を使用する場合、値を書き込む変数名をリストするか、文字列を引用符で囲んで指定します。PUT ステートメントは変数値を書き込んだ後、ブランクを1つ挿入してから次の値を書き込みます。数値の欠損値は1つのピリオドとして書き込まれます。文字列はフィールドに左寄せで表示されます。また文字列の先頭と末尾にあるブランクは削除されます。それぞれの値の後ろに追加されるブランク以外に、ブランクを挿入するには、リスト出力ではなく、フォーマット出力または列出力を使用します。

リスト入力には、次の2種類があります。

- 単純リスト出力
- 修飾リスト出力

修飾リスト出力では、出力形式を指定して変数値を書き込む方法を制御できるため、PUT ステートメントの用途が広がります。参照: [“例 3: 修飾リスト出力\(:\)を用いた値の書き込み” \(325 ページ\)](#)。

リスト出力のスペース 設定方法

リスト出力は、変数値や文字列を書き込む場合に、各種のスペース 設定を使用します。リスト出力を使用して変数値を書き込む場合、ブランクが自動的に挿入されます。そのため、出力ポインターは変数値の後の 2 列目に移動します。ただし、文字列を書き込む場合はブランクは自動的に挿入されません。この場合、出力ポインターは文字列の最終文字のすぐ後の列に移動します。

文字列や変数値の書き込み時にスペース 設定の問題が発生しないように、文字列の最終文字としてブランクを 1 つ追加することをお勧めします。変数値の後ろに句読点を提供する文字列を続ける場合、出力ポインターを戻す必要があります。出力ポインターを戻すことで、出力行に不要なブランクが挿入されることを防げます。参照: “例 2: 文字列と変数値の書き込み” (325 ページ)。

修飾リスト出力とフォーマット出力の違い

リスト出力とフォーマット出力は、異なる方式を使用して、変数値の書き込み後にポインターをどれだけ移動するかを決定します。そのため、出力形式を使用する修飾リスト出力とフォーマット出力で、出力行の結果が異なります。修飾リスト出力では、値を書き込みブランクを 1 つ挿入してから、ポインターを次の列に移動します。フォーマット出力では、値が指定した出力形式の長さに満たない場合でも、出力形式の長さの分だけポインターを移動します。ポインターは、ブランクを挿入せずに、次の列に移動します。

この DATA ステップは、修飾リスト出力を使用して各出力行を書き込みます。

```
data _null;
  input x y;
  put x : comma10.2 y : 7.2;
  datalines;
2353.20 7.10
6231 121
;
```

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。

```
----+----1----+----2
2,353.20 7.10
6,231.00 121.00
```

比較するため、この例ではフォーマット出力を使用します。

```
put x comma10.2 y 7.2;
```

これらの行が、列内の値の位置が揃えられて SAS ログに書き込まれます。

```
----+----1----+----2
2,353.20 7.10
6,231.00 121.00
```

例

例 1: リスト出力の値の書き込み

次の DATA ステップは、リスト出力を伴う PUT ステートメントを使用して、変数値を SAS ログに書き込みます。

```
data _null;
  input name $ 1-10 sex $ 12 age 15-16;
  put name sex age;
  datalines;
Joseph   M 13
Mitchel  M 14
Sue Ellen F 11
;
```

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。

```
----+-----1----+-----2----+-----3----+-----4
Joseph M 13
Mitchel M 14
Sue Ellen F 11
```

デフォルトでは、文字変数 NAME の値はフィールド内に左寄せで表示されます。

例 2: 文字列と変数値の書き込み

次の PUT ステートメントは、文字列の末尾に 1 つの空白を追加し、出力ポインターを逆報告に移動することにより、変数 STARTWGHT の後の出力行内に不要な空白が表示されないようにします。

```
data _null;
  input idno name $ startwght;
  put name 'weighs ' startwght +(-1) ' ';
  datalines;
032 David 180
049 Amelia 145
219 Alan 210
;
```

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。

```
David weighs 180.
Amelia weighs 145.
Alan weighs 210.
```

文字列の最後にブランクが追加されるため、ポインターの位置が変わります。このブランクによって、文字列と後続の変数値が区切られます。ポインターコントロールを+(-1)と指定しているので、ポインターが後方(左)に移動し、STARTWGHT の値とピリオドの間の不要なブランクが削除されます。

例 3: 修飾リスト出力(:)を用いた値の書き込み

次の DATA ステップは、: 引数を用いた修飾リスト出力を使用して、複数の変数値を 1 つの出力行に書き込みます。

```
data _null;
  input salesrep : $10. tot : comma6. date : date9.;
  put 'Week of ' date : worddate15.
    salesrep : $12. 'sales were '
    tot : dollar9. +(-1) ' ';
  datalines;
Wong 15,300 12OCT2004
```

```
Hoffman 9,600 12OCT2004
;
```

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。

```
Week of Oct 12, 2004 Wong sales were $15,300.
Week of Oct 12, 2004 Hoffman sales were $9,600.
```

例 4: 修飾リスト出力(~)を用いた値の書き込み

次の DATA ステップは、~引数を用いた修飾リスト出力を使用して、複数の変数値を 1 つの出力行に書き込みます。

```
data _null_;
  input salesrep : $10. tot : comma6. date : date9.;
  file log delimiter=" " dsd;
  put 'Week of ' date ~ worddate15.
      salesrep ~ $12. 'sales were '
      tot ~ dollar9. + (-1) '.';
  datalines;
Wong 15,300 12OCT2004
Hoffman 9,600 12OCT2004
;
```

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。

```
Week of "Oct 12, 2004" "Wong" sales were "$15,300".
Week of "Oct 12, 2004" "Hoffman" sales were "$9,600".
```

関連項目

ステートメント:

- [“PUT ステートメント” \(289 ページ\)](#)
- [“PUT ステートメント: フォーマット” \(316 ページ\)](#)

PUT ステートメント: 名前付き

変数名と等号の後に変数値を書き込みます。

該当要素: DATA ステップ

カテゴリ: CAS
ファイル操作

種類: 実行

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

PUT *<pointer-control>* *variable*= *<format.>* *<@ | @@>*;

PUT *variable*= *start-column* *<-end-column>* *<.decimal-places>* *<@ | @@>*;

引数

pointer-control

出力バッファ内の指定した行または列に出力ポインターを移動させます。

参照項目: [“引数: 列:ポインターコントロール” \(291 ページ\)](#)

[“引数: 行:ポインターコントロール” \(293 ページ\)](#)

variable=

変数を次の形式で指定します。指定した変数値が PUT ステートメントで書き込まれます。

variable=*value*

format.

変数の値を書き込むときに使用する出力形式を指定します。

ヒント 出力形式の長さには、値に加えて出力形式の指定に使用するカンマ、ドル記号、小数点、その他の特殊文字を表示するのに十分なスペースを指定してください。

参照項目: [“名前付き出力のフォーマット” \(328 ページ\)](#)
目:

例 次の PUT ステートメントは、出力形式 DOLLAR7.2 を使用して X の値を書き込みます。

```
put x= dollar7.2;
```

X=100 の場合、フォーマット値は 7 列を使用します。

X=\$100.00

start-column

変数名、等号、値を書き込む出力行のフィールドの開始列位置を指定します。

- end-column

値のフィールドの最終列位置を指定します。

ヒント 変数名、等号、値の表示に必要な列数が指定した列数より多い場合、値を切り捨てずに列の最終位置を越えて書き込みます。そのため、次の値の開始位置との間に十分なスペースを確保する必要があります。

.decimal-places

数値で使用する小数点以下の桁数を指定します。*d* に 0 を指定したり、*d* の値を省略する場合は、小数点なしの値が書き込まれます。

範囲 正の整数

@ | @@

次の PUT ステートメントの実行時に使用できるように出力行を保持します。出力行は、DATA ステップの反復間でも保持されます。このラインホールド指定子は、後置@および後置@@と呼ばれます。

制限事項 後置@または後置@@は、PUT ステートメントの最後のアイテムとして指定する必要があります。

参照項目: [“ラインホールド指定子の使用” \(303 ページ\)](#)

詳細

名前付き出力の使用

名前付き出力を使用するには、PUT ステートメントに変数名の後に等号を指定します。変数名と値の表示位置を指定するには、リスト出力、列出力、フォーマット出力のいずれかを使用できます。各変数の間に自動的にブランクを挿入するには、リスト出力を使用します。列の出力をそろえるには、ポインターコントロールと列を指定します。

名前付き出力のフォーマット

SAS 出力形式またはユーザー定義の出力形式のいずれかを指定して、変数値の出力を制御することができます。出力形式の幅には、変数名と等号の表示に必要な列は含まれません。フォーマット値を揃えるため、変数値は先頭にあるブランクは削除され、等号のすぐ後ろに書き込まれます。名前付きでないフォーマット出力のように、指定した幅の右側に揃えることはありません。

SAS 出力形式の詳細については、[“出力形式について” \(SAS 出力形式と入力形式: リファレンス\)](#)を参照してください。

例: PUT ステートメントで名前付き出力を使用する

PUT ステートメントで名前付き出力を使用する例を次に示します。

- この PUT ステートメントでは、名前付き出力と列ポインターコントロールを組み合わせて出力を配置します。

```
data _null_;
  input name $ 1-18 score1 score2 score3;
  put name = @20 score1= score3=;
  datalines;
Joseph      11  32  76
Mitchel     13  29  82
Sue Ellen   14  27  74
;
```

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。

```
-----1-----2-----3-----4
NAME=Joseph    SCORE1=11 SCORE3=76
```

```
NAME=Mitchel   SCORE1=13 SCORE3=82
NAME=Sue Ellen SCORE1=14 SCORE3=74
```

- この例では、変数 AMOUNT の出力形式を指定します。

```
put item= @25 amount= dollar12.2;
```

ITEM の値が binders で AMOUNT の値が 153.25 の場合、出力行は次のように表示されます。

```
-----1-----2-----3-----4
ITEM=binders      AMOUNT=$153.25
```

関連項目

ステートメント:

- [“PUT ステートメント” \(289 ページ\)](#)

PUTLOG ステートメント

SAS ログにメッセージを書き込みます。

該当要素: DATA ステップ

カテゴリ: アクション

CAS

In-Database 処理

種類: 実行

ヒント: メッセージテキストの前に、キーワード [ERROR](#)、[WARNING](#)、または [NOTE](#) を追加すると、SAS ログの出力内容を判別しやすくなります。この機能は [PUT ステートメント](#)と[%PUT マクロステートメント](#)両方で使用できます。

構文

PUTLOG <'text'>;

PUTLOG <'ERROR: text'> <'NOTE: text'> <'WARNING: text'>

PUTLOG <'ERROR- text'> <'NOTE- text'> <'WARNING- text'>;

オプション引数

text

SAS ログに書き込むカスタムメッセージを指定します。*Text* には、文字リテラル (単一引用符または二重引用符で囲む)、変数名、出力形式、ポインターコントロールを使用できます。

要件 テキストは単一引用符または二重引用符で囲む必要があります。

ERROR: text

キーワード ERROR の後に、カスタムログメッセージを赤字で SAS ログに表示します。

```
data L;
  putlog 'ERROR: This is my error message in red text';
run;
```

```
ERROR: This is my error message in red text;
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード ERROR は、すべて大文字で、直後にコロン(:)を付ける必要があります。

ERROR- text

SAS ログに赤字で、ERROR というキーワードを含まないメッセージを表示します。

```
data L;
  putlog 'ERROR: This is my error message in red text';
  putlog 'ERROR- This is the second line of my error message';
run;
```

```
ERROR: This is my error message in red text
      This is the second line of my error message
```

```
This is my error message in red text
This is my note in blue text
This is my warning in green text
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード ERROR は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

NOTE: text

キーワード NOTE の後に、カスタムログメッセージを青字で SAS ログに表示します。

```
data L;
  putlog 'NOTE: This is my note in blue text';
run;
```

```
NOTE: This is my note message in blue text;
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード NOTE は、すべて大文字で、直後にコロン(:)を付ける必要があります。

NOTE- text

SAS ログに青字で、NOTE というキーワードを含まないメッセージを表示します。

```
data L;
  putlog 'NOTE: This is my note in blue text';
  putlog 'NOTE- This is the second line of my note message';
run;
```

```
NOTE: This is my note message in blue text
      This is the second line of my note message
```

```
This is my error message in red text
This is my note in blue text
This is my warning in green text
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード NOTE は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

WARNING: *text*

キーワード WARNING の後に、カスタムログメッセージを緑字で SAS ログに表示します。

```
data L;
  putlog 'WARNING: This is my warning in green text';
run;
```

```
WARNING: This is my warning message in green text;
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR: Red text';
  y = 'NOTE: Blue text';
  z = 'WARNING: Green text';
  put x; put y; put z;
run;
```

```
ERROR: Red text
NOTE: Blue text
WARNING: Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード WARNING は、すべて大文字で、直後にコロン(:)を付ける必要があります。

WARNING- *text*

SAS ログに緑字で、WARNING というキーワードを含まないメッセージを表示します。

```
data L;
  putlog 'WARNING: This is my warning in green text';
  putlog 'WARNING- This is the second line of my warning message';
```

```
run;
```

```
WARNING:  This is my warning message in green text
          This is the second line of my warning message
```

キーワードは、文字変数の代入文で文字列値の一部として使用することもできます。

```
data L;
  x = 'ERROR- Red text';
  y = 'NOTE- Blue text';
  z = 'WARNING- Green text';
  put x; put y; put z;
run;
```

```
Red text
Blue text
Green text
```

要件 キーワードとテキストは単一引用符または二重引用符で囲む必要があります。

キーワード WARNING は、すべて大文字で、直後にハイフン(-)を付ける必要があります。

詳細

PUTLOG ステートメントでは、指定したメッセージを SAS ログに書き込みます。現在のファイル出力先に影響せずに SAS ログに出力するため、PUTLOG ステートメントはマクロを生成するコードの使用時にも便利です。

比較

PUTLOG ステートメントは ERROR ステートメントに似ていますが、PUTLOG ステートメントを実行しても自動変数_ERROR_の値は 1 に設定されません。

例: PUTLOG ステートメントを BY 使用してメッセージを SAS ログに書き込む

次のプログラムでは、computeAverage92 マクロを作成します。このマクロは、平均点の算出、入力データの検証、PUTLOG ステートメントを使用した SAS ログへのエラーメッセージの書き込みを行います。DATA ステップでは、PUTLOG ステートメントを使用して警告メッセージを SAS ログに書き込みます。

```
data ExamScores;
  input Name $ 1-16 Score1 Score2 Score3;
  datalines;
```

```

Sullivan, James 86 92 88
Martinez, Maria 95 91 92
Guzik, Eugene 99 98 .
Schultz, John 90 87 93
van Dyke, Sylvia 98 . 91
Tan, Carol 93 85 85
;

filename outfile 'path-to-your-output-file';
/* Create a macro that computes the average score, validates */
/* input data, and uses PUTLOG to write error messages to the */
/* SAS log. */
%macro computeAverage92(s1, s2, s3, avg);
  if &s1 < 0 or &s2 < 0 or &s3 < 0 then
    do;
      putlog 'ERROR: Invalid score data ' &s1= &s2= &s3=;
      &avg = .;
    end;
  else
    &avg = mean(&s1, &s2, &s3);
%mend;
data _null_;
set ExamScores;
file outfile;
%computeAverage92(Score1, Score2, Score3, AverageScore);
put name Score1 Score2 Score3 AverageScore;
/* Use PUTLOG to write a warning message to the SAS log. */
if AverageScore < 92 then
  putlog 'WARNING: Score below the minimum ' name= AverageScore= 5.2;
run;

proc print;
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。

```

WARNING: Score below the minimum Name=Sullivan, James AverageScore=88.67
ERROR: Invalid score data Score1=99 Score2=98 Score3=.
WARNING: Score below the minimum Name=Guzik, Eugene AverageScore=.
WARNING: Score below the minimum Name=Schultz, John AverageScore=90.00
ERROR: Invalid score data Score1=98 Score2=.Score3=91
WARNING: Score below the minimum Name=van Dyke, Sylvia AverageScore=.
WARNING: Score below the minimum Name=Tan, Carol AverageScore=87.67

```

この出力ファイルが作成されます。

```

Sullivan, James 86 92 88 88.666666667
Martinez, Maria 95 91 92 92.666666667
Guzik, Eugene 99 98 . .
Schultz, John 90 87 93 90
van Dyke, Sylvia 98 . 91 .
Tan, Carol 93 85 85 87.666666667

```


関連項目

ステートメント:

- [“ERROR ステートメント” \(102 ページ\)](#)
- [“PUT ステートメント” \(289 ページ\)](#)

REDIRECT ステートメント

ストアドプログラムを実行するときに、別の入力 SAS データセットまたは出力 SAS データセットを参照します。

該当要素: DATA ステップ

カテゴリ: アクション

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。
このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User’s Guide\)](#)を参照してください。

要件: DATA ステートメントに PGM=オプションを指定する必要があります。

構文

```
REDIRECT INPUT | OUTPUT old-name-1=new-name-1 <...old-name-n=new-name-n>;
```

引数

INPUT | OUTPUT

入力データセットまたは出力データセットのどちらをリダイレクトするか指定します。INPUT を指定すると、プログラム内の入力データセット名が別の SAS データセット名に割り当てられます。OUTPUT を指定すると、出力データセット名が別の SAS データセット名に割り当てられます。

old-name

プログラム内の入力データセットまたは出力データセット名を指定します。

new-name

現在の実行で処理する入力データセットまたは出力データセット名を指定します。

詳細

REDIRECT ステートメントはストアドプログラムを実行する場合にのみ使用できます。

注意

入力データセットをリダイレクトするときは注意が必要です。 REDIRECT ステートメントで読み込む入力データセット内の変数の数や属性は、ソースコード内の MERGE、SET、MODIFY、または UPDATE の各ステートメントの入力データセットで指定した変数の数や属性と一致している必要があります。変数の種類の属性が異なる場合、ストアドプログラムの処理が中止し、該当するエラーメッセージが SAS ログに書き込まれます。変数の長さ属性が異なる場合、リダイレクトされるデータセットに含まれる変数の長さは、ソースコードのデータセットに含まれる変数の長さによって決定します。リダイレクト先のデータセットのほうが変数の数が多い場合は、ストアドプログラムの処理が中止し、該当するエラーメッセージが SAS ログに書き込まれます。

ヒント REDIRECT ステートメントで読み込む入力データセット内の変数の数が、プログラムをコンパイルするために使用するデータセット内の変数の数よりも多い場合、ストアドプログラムに DROP または KEEP データセットオプションを追加できます。

比較

REDIRECT ステートメントは、SAS データセットに対してのみ適用されます。外部ファイルに指定されている入力データセットまたは出力データセットにリダイレクトするには、FILENAME ステートメントを使用してソースプログラムのファイル参照名(fileref)と外部ファイルを関連付けます。

例: ストアドプログラムの実行

この例では、STORED.SAMPLE というストアドプログラムを実行します。REDIRECT ステートメントでは、入力データソース BASE.SAMPLE を指定しています。このプログラムの実行で使用する出力データセットは、SUMS.SAMPLE というデータセットにリダイレクトした後、格納されます。

```
libname stored 'SAS-library';
libname base 'SAS-library';
libname sums 'SAS-library';
data pgm=stored.sample;
  redirect input in.sample=base.sample;
  redirect output out.sample=sums.sample;
run;
```

関連項目

ステートメント:

- [“DATA ステートメント” \(68 ページ\)](#)

REMOVE ステートメント

SAS データセットからオブザベーションを削除します。

該当要素: DATA ステップ

カテゴリ: アクション

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。
このステートメントは、In-database DATA ステップではサポートされていません。
詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。
必ず MODIFY ステートメントと組み合わせて使用します。

構文

REMOVE <[data-set-name\(s\)](#)>;

引数なし

引数を指定しない場合、REMOVE ステートメントは DATA ステートメントに指定されているすべてのデータセットから現在のオブザベーションを削除します。

引数

data-set-name

オブザベーションを削除するデータセットを指定します。

制限事項 このデータセット名は、DATA ステートメントおよび 1 つ以上の MODIFY ステートメントにも指定する必要があります。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

詳細

オブザベーションは、データセットを管理するエンジンによって、物理的または論理的に削除されます。REMOVE ステートメントの使用は、オブザベーションのデフォルトの置き換えより優先されます。DATA ステップで REMOVE ステートメントを使用する場合、DATA ステップのすべての出力を明示的にプログラムする必要があります。

比較

- OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)DATA ステップでこれらのステートメントのいずれかを使用する場合、新しいオブザベーションのすべての出力を明示的にプログラムする必要があります。
- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。
- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインターの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。
- REMOVE ステートメントは物理的または論理的な削除が実行できるため、すべての SAS データセットエンジンにおいて、REMOVE と MODIFY ステートメントの併用が可能です。DELETE ステートメントおよびサブセット化 IF ステートメントは、どちらも物理的な削除のみを実行します。そのため、エンジンによっては、MODIFY ステートメントとこれらのステートメントを併用できない場合があります。

例: データセットからオブザベーションを削除する

この例では、SAS データセットからオブザベーションを 1 つ削除します。

```
libname perm 'SAS-library';

data perm.accounts;
  input AcctNumber Credit;
  datalines;
1001 1500
1002 4900
1003 3000
;
data perm.accounts;
  modify perm.accounts;
  if AcctNumber=1002 then remove;
run;
proc print data=perm.accounts;
```

```
title 'Edited Data Set';
run;
```

PROC PRINT ステートメントの実行結果を次に示します。

アウトプット 2.36 編集したデータセット

Edited Data Set		
Obs	AcctNumber	Credit
1	1001	1500
3	1003	3000

関連項目

ステートメント:

- [“DELETE ステートメント” \(85 ページ\)](#)
- [“IF ステートメント: サブセット化” \(137 ページ\)](#)
- [“MODIFY ステートメント” \(261 ページ\)](#)
- [“OUTPUT ステートメント” \(285 ページ\)](#)
- [“REPLACE ステートメント” \(341 ページ\)](#)

RENAME ステートメント

出力 SAS データセットの変数に新しい名前を指定します。

該当要素: DATA ステップ

カテゴリ: CAS

情報

種類: 宣言

制限事項: このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

構文

```
RENAME old-name-1=new-name-1 <...old-name-n=new-name-n>;
```

引数

old-name

入力データセットに含まれる変数名または変数リスト、または現在の DATA ステップで新しく作成する変数の名前を指定します。

new-name

出力データセットで使用する変数名またはリストを指定します。

詳細

RENAME ステートメントでは、1 つまたは複数の変数、1 つのリスト内の変数、または変数と変数リストの組み合わせに対して名前を変更できます。新しい変数名は、出力データセットのみに書き込まれます。現在の DATA ステップのプログラムステートメントには変更前の変数名を使用してください。RENAME ステートメントはすべての出力データセットに適用されます。

注: RENAME ステートメントは、出力モードで開かれるデータセットのみに影響します。

比較

- PROC ステップでは RENAME ステートメントを使用することはできません、RENAME=データセットオプションを使用することができます。
- RENAME=データセットオプションを使用すると、名前を変更する変数を入力データセットまたは出力データセットごとに指定できます。処理を実行する前に変数の名前を変更するには、入力データセットにこのオプションを使用します。
- 出力データセットに RENAME=データセットオプションを使用する場合、現在の DATA ステップのプログラムステートメントでは変更前の変数名を使用する必要があります。出力データセットが作成されると、新しい変数名を使用できるようになります。
- SET ステートメントに RENAME=データセットオプションを指定すると、入力データセットの変数名を変更できます。現在の DATA ステップのプログラムステートメントでも新しい名前を使用できるようになります。
- ファイル管理として変数名を変更する場合は、DATASETS プロシジャを使用するか、SAS のウィンドウ環境インターフェイスから変数にアクセスします。この方法は簡単に実行できます。また、DATA ステップを処理する必要はありません。

例: データセット変数の名前を変更する

- これらの例では、RENAME ステートメントを使用して変数名を変更する場合の正しい構文を示します。

```
rename street=address;
rename time1=temp1 time2=temp2 time3=temp3;
rename name=Firstname score1-score3=Newscore1-Newscore3;
```

- この例では、変更する前の変数名がプログラムステートメントで使用されています。出力データセットで、変数 Olddept が Newdept に、そして、変数 Oldaccount が ewaccount に名前が変更されます。

```
rename Olddept=Newdept Oldaccount=Newaccount;
if Oldaccount>5000;
keep Olddept Oldaccount items volume;
```

- この例では、変更する前の変数名 OLDACCNT がプログラムステートメントで使用されています。ただし、DATA ステートメントでは新しい変数名 NEWACCNT が使用されています。これは、KEEP=データセットオプションが適用される前に RENAME ステートメントが適用されるからです。

```
data market(keep=newdept newaccnt items volume);
  rename olddept=newdept oldaccnt=newaccnt;
  set sales;
  if oldaccnt>5000;
run;
```

- この例では、変数と変数リストの両方を使用して変数の名前を変更します。新しい変数名は出力データセットに表示されます。

```
data temp;
  input (score1-score3) (2.,+1) name $;
  rename name=Firstname
         score1-score3=Newscore1-Newscore3;
  datalines;
12 24 36 Lisa
22 44 66 Fran
;
```

関連項目

データセットオプション:

- ["RENAME= データセットオプション" \(SAS データセットオプション: リファレンス\)](#)

REPLACE ステートメント

同じ場所にあるオブザベーションを置換します。

該当要素: DATA ステップ

カテゴリ: アクション

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。

このステートメントは、In-database DATA ステップではサポートされていません。詳細については、“[DATA Step Acceleration](#)” ([SAS In-Database Products: User's Guide](#))を参照してください。

必ず MODIFY ステートメントと組み合わせて使用します。

構文

REPLACE <*data-set-name-1*> <...*data-set-name-n*>;

引数なし

引数を指定しない場合、REPLACE ステートメントは、DATA ステートメントに指定したすべてのデータセットに対して、オブザベーションの読み込み元の物理的な場所に現在のオブザベーションを書き込みます。

引数

data-set-name

オブザベーションを書き込むデータセットを指定します。

要件 このデータセット名は、DATA ステートメントおよび 1 つ以上の MODIFY ステートメントにも指定する必要があります。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

詳細

明示的な REPLACE ステートメントの使用は、オブザベーションのデフォルトの置き換えより優先されます。DATA ステップで REPLACE ステートメントを使用する場合、DATA ステップのすべての出力をプログラムする必要があります。

比較

- OUTPUT、REPLACE、REMOVE の各ステートメントの使用は、DATA ステップの最後に実行されるデフォルトの書き込みアクションより優先されます。(OUTPUT がデフォルトのアクションです。ただし、MODIFY ステートメントを使用すると REPLACE がデフォルトのアクションになります。)DATA ステップでこれらのステートメントのいずれかを使用する場合、このステップの新しいオブザベーションの出力を明示的にプログラムする必要があります。

- OUTPUT、REPLACE、REMOVE の各ステートメントは互いに独立して動作します。順序が論理的に正しければ、同じオブザベーションに複数のステートメントを適用できます。
- 特定のオブザベーションに対して OUTPUT ステートメントと REPLACE または REMOVE ステートメントの両方を実行する場合、オブザベーションのポインターの位置を正しく保つため、OUTPUT ステートメントを最後に実行します。
- REPLACE はオブザベーションを元の物理的な場所を書き込みます。OUTPUT は新しいオブザベーションをデータセットの最後に書き込みます。
- REPLACE ステートメントは、MODIFY ステートメントを使用した DATA ステップにのみ使用できます。OUTPUT ステートメントは、MODIFY ステートメントの有無にかかわらず指定できます。

例: オブザベーションの置き換え

この例では、データセット MASTER にある電話番号をデータセット TRANS にある値で置き換えます。また、データセット MASTER の最後に新しいオブザベーションが 1 つ追加されます。SYSRC 自動呼び出しマクロは、MASTER からデータを取得するたびに `_IORC_` の値を確認します。(SYSRC は、SAS 自動呼び出しマクロライブラリのメンバーです。)コードの後に、生成される SAS データセットが表示されます。

```
data master;
  input FirstName $ id $ PhoneNumber;
  datalines;
Kevin ABCjkh 904
Sandi defsns 905
Terry ghitDP 951
Jason jklJWM 962
;
data trans;
  input FirstName $ id $ PhoneNumber;
  datalines;
. ABCjkh 2904
. defsns 2905
Madeline mnombt 2983
;
data master;
  modify master trans;
  by id;
  /* obs found in master */
  /* change info, replace */
  if _iorc_ = %sysrc(_sok) then replace;
  /* obs not in master */
  else if _iorc_ = %sysrc(_dsenmr) then
    do;
      /* reset _error_ */
      _error_ = 0;
      /* reset _iorc_ */
      _iorc_ = 0;
      /* output obs to master */
    output;
  end;
run;
```

```
proc print data=master;  
  title 'MASTER with New Phone Numbers';  
run;
```

アウトプット 2.37 オブザベーションが置き換えられたデータセット

MASTER with New Phone Numbers			
Obs	FirstName	id	PhoneNumber
1	Kevin	ABCjkh	2904
2	Sandi	defsns	2905
3	Terry	ghitDP	951
4	Jason	jklJWM	962
5	Madeline	mnombt	2983

関連項目

ステートメント:

- [“MODIFY ステートメント” \(261 ページ\)](#)
- [“OUTPUT ステートメント” \(285 ページ\)](#)
- [“REMOVE ステートメント” \(337 ページ\)](#)

RESETLINE ステートメント

SAS ログ内のプログラムの行番号を 1 にリセットします。

注: [RESETLINE ステートメント](#)は SAS グローバルステートメントに移動しました。

RETAIN ステートメント

DATA ステップを繰り返す際に、INPUT ステートメントまたは割り当てステートメントで作成される変数の値を保持します。

該当要素: DATA ステップ
カテゴリ: CAS
In-Database 処理
情報

種類: 宣言

制限事項: In-database DATA ステップの場合、RETAIN ステートメントは次の形式の変数リスト構文をサポートしていません。詳細については、“[DATA Step Acceleration](#)” ([SAS In-Database Products: User's Guide](#))を参照してください。

- x1-x10
- _NUMERIC_
- a-NUMERIC-b

構文

```
RETAIN <element-list(s) <initial-value(s) | (initial-value-1) | (initial-value-list-1)>
<... element-list-n <initial-value-n | (initial-value-n) | (initial-value-list-n)>>>;
```

引数なし

引数を指定しない場合、DATA ステップを繰り返す際に、RETAIN ステートメントは INPUT ステートメントまたは割り当てステートメントで作成されるすべての変数の値を保持します。

引数

element-list

値を保持する変数名、変数リスト、配列名を指定します。

ヒン ト *_ALL_*、*_CHAR_*、*_NUMERIC_*を指定する場合、RETAIN ステートメントの前に定義した変数のみに影響します。

変数名が RETAIN ステートメントにのみ指定されている場合に、その初期値を指定しないと、この変数はデータセットに書き込まれず、変数が初期化されていないことを示す注釈が SAS ログに書き込まれます。初期値を設定すると、この変数はデータセットに書き込まれます。

initial-value

直前に指定した 1 つまたは複数の要素に対し、初期値(数値または文字)を指定します。

ヒン ト *initial-value* を省略すると、初期値は欠損値になります。*Initial-value* は、その前にあるリストのすべての要素に割り当てられます。このため、変数リストのすべてのメンバーに同じ初期値が割り当てられます。

参照項目: (*initial-value*)および(*initial-value-list*)

(initial-value)

直前に指定した 1 つの要素、または要素リストにある最初の要素に対し、初期値(数値または文字)を指定します。

(initial-value-list)

直前に指定したリストにある各要素に対し、初期値(数値または文字)を指定します。このリストの最初の値を要素リストの最初の変数に、2 番目の値を要素リストの 2 番目の変数に割り当てます。

要素の文字値は引用符で囲みます。1 つまたは複数の初期値を直接指定するには、次の形式を使用します。

(initial-value(s))

繰り返しの回数とネスト形式のサブリストを初期値に指定するには、次の形式を使用します。

`<constant-iter-value*> <(>constant value | constant-sublist<)>`

制限事項 *initial-value-list* と *element-list* の両方を指定する場合、RETAIN ステートメントでは *element-list* を *initial-value-list* より前にリストする必要があります。

ヒント ブランクまたはカンマで複数の初期値を区切ることができます。

連続する整数値の範囲を指定する場合は、簡略化した表記を使用できます。増分は常に+1 になります。

変数と一時データ要素の両方に初期値を割り当てることができます。

指定した初期値の数よりも変数の数が多い場合、初期値が割り当てられなかった変数には欠損値が割り当てられ、警告メッセージが発行されます。

詳細

DATA ステップのデフォルトの動作

RETAIN ステートメントを使用しない場合、DATA ステップを繰り返す前に、INPUT ステートメントまたは割り当てステートメントを使用して割り当てられる変数は欠損値に設定されます。

初期値の割り当て

個別の変数、変数リスト、配列のメンバーに初期値を指定するには、RETAIN ステートメントを使用します。RETAIN ステートメントに値が表示される場合、リスト内でこの値より前に表示される変数の初期値は表示された値に設定されます。(RETAIN ステートメントで同じ変数に異なる初期値を複数回指定した場合、最後の値が初期値として使用されます。)RETAIN ステートメントを使用すると、合計ステートメントで割り当てられた変数の値にデフォルト値 0 以外の初期値を設定することもできます。

指定の重複

RETAIN ステートメントで次のアイテムを指定すると、指定が重複することになります。次に示すアイテムの値は自動的に保持され、DATA ステップの繰り返しの際に使用されます。

- SET、MERGE、MODIFY、UPDATE の各ステートメントで読み込まれる変数
- 合計ステートメントで値が割り当てられた変数
- 自動変数_N_、_ERROR_、_I_、_CMD_、および_MSG_
- SET、MERGE、MODIFY、UPDATE の各ステートメントで END=または IN=オプションを指定するか、FILE および INFILE の各ステートメントで変数を作成するオプションを指定して作成された変数
- 一時配列で指定されたデータ要素
- ARRAY ステートメントで初期化された配列要素
- ARRAY ステートメントで任意の要素またはすべての要素に初期値が割り当てられた配列要素

ただし、RETAIN ステートメントを使用して前述のアイテムに初期値を割り当てることができます。自動変数_N_と_ERROR_には、初期値を設定できません。

比較

RETAIN ステートメントでは、DATA ステップの各繰り返しの初めに、その値が欠損値に設定されない変数を指定します。KEEP ステートメントでは、作成されるデータセットに書き込む変数を指定します。

例

例 1: 基本的な使用

- 次の RETAIN ステートメントでは、変数 MONTH1 から MONTH5 の値を保持し、DATA ステップの繰り返しで使用します。

```
retain month1-month5;
```

- 次の RETAIN ステートメントでは、9 個の変数の値を保持し、その初期値を設定します。

```
retain month1-month5 1 year 0 a b c 'XYZ';
```

変数 MONTH1 から MONTH5 の初期値は 1、YEAR の初期値は 0、変数 A、B、C の初期値は文字列値 XYZ にそれぞれ設定されます。

- 次の RETAIN ステートメントは、変数 MONTH1 にのみ初期値 1 を割り当てます。

```
retain month1-month5 (1);
```

変数 MONTH2 から MONTH5 の初期値は欠損値に設定されます。

- 次の RETAIN ステートメントでは、DATA ステップの上部で定義されたすべての変数の値が保持されますが、その後定義された値は保持されません。

```
retain _all_;
```

- 次の場合、どのステートメントを実行しても変数 VAR1 から VAR4 に初期値 1 から 4 が割り当てられます。

```
□ retain var1-var4 (1 2 3 4);
```

```
□ retain var1-var4 (1,2,3,4);
```

```
□ retain var1-var4(1:4);
```

例 2: RETAIN ステートメントの操作概要

この例は、RETAIN ステートメントの要素として変数名と配列名を使用する方法を示しています。割り当てる初期値はかっこで囲まれているものと、かっこで囲まれていないものがあります。

```
data _null_;
  array City{3} $ City1-City3;
  array cp{3} Citypop1-Citypop3;
  retain Year Taxyear 1999 City ''
         cp (10000,50000,100000);
  file file-specification print;
  put 'Values at beginning of DATA step.'
    / @3 _all_ /;
  input Gain;
  do i=1 to 3;
    cp{i}=cp{i}+Gain;
  end;
  put 'Values after adding Gain to city populations.'
    / @3 _all_ /;
  datalines;
5000
10000
;
```

RETAIN ステートメントで割り当てられた初期値は次のようになります。

- Year と Taxyear の初期値は 1999 になります。
- City1、City2、City3 には欠損値が割り当てられます。
- Citypop1 には 10000 が割り当てられます。
- Citypop2 には 50000 が割り当てられます。
- Citypop3 には 100000 が割り当てられます。

PUT ステートメントで書き込まれた行を次に示します。

```

Values at beginning of DATA step:
  City1= City2= City3= Citypop1=10000
  Citypop2=50000 Citypop3=100000
Year=1999 Taxyear=1999 Gain=, i=,
_ERROR_=0 _N_=1
Values after adding GAIN to city populations:
  City1= City2= City3= Citypop1=15000
  Citypop2=55000 Citypop3=105000
Year=1999 Taxyear=1999 Gain=5000 i=4
_ERROR_=0 _N_=1
Values at beginning of DATA step:
  City1= City2= City3= Citypop1=15000
  Citypop2=55000 Citypop3=105000
Year=1999 Taxyear=1999 Gain=, i=,
_ERROR_=0 _N_=2
Values after adding GAIN to city populations:
  City1= City2= City3= Citypop1=25000
  Citypop2=65000 Citypop3=115000
Year=1999 Taxyear=1999 Gain=10000 i=4
_ERROR_=0 _N_=2
Values at beginning of DATA step:
  City1= City2= City3= Citypop1=25000
  Citypop2=65000 Citypop3=115000
Year=1999 Taxyear=1999 Gain=, i=,
_ERROR_=0 _N_=3

```

最初の PUT ステートメントは 3 回実行されていますが、2 番目の PUT ステートメントは 2 回しか実行されていません。3 回目の INPUT ステートメントを実行して END ステートメントに到達すると、DATA ステップは終了します。

例 3: 複数のオブザベーションから 1 つの値を選択する

この例のデータセット ALLSCORESID には、個々の ID 番号および変数 ID に対して複数のオブザベーションが含まれています。ある 1 つの ID 値を持つ複数のオブザベーションでは、変数 GRADE の値が異なる場合があります。ここでは、新しいデータセット CLASS.BESTSCORES を作成します。作成される新しいデータセットには、ID の値ごとに 1 つのオブザベーションが含まれます。BESTSCORES データセットのオブザベーションは、ID が同じすべてのオブザベーションのうちで変数 GRADE の値が最も大きいオブザベーションになります。

```

libname class 'SAS-library';
proc sort data=class.allscores;
  by id;
run;
data class.bestscores;
  drop grade;
  set class.allscores;
  by id;
  /* Prevents HIGHEST from being reset*/
  /* to missing for each iteration. */
  retain highest;
  /* Sets HIGHEST to missing for each */
  /* different ID value. */
  if first.id then highest=.;
  /* Compares HIGHEST to GRADE in */
  /* current iteration and resets */
  /* value if GRADE is higher. */
  highest=max(highest,grade);
  if last.id then output;

```

```
run;
```

関連項目

ステートメント:

- [“割り当て ステートメント” \(51 ページ\)](#)
- [“BY ステートメント” \(57 ページ\)](#)
- [“INPUT ステートメント” \(186 ページ\)](#)

その他のドキュメント

- [“行間依存関係とマルチスレッド処理” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)
- [“テーブル全体での変数の合計” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)

RETURN ステートメント

DATA ステップ内の現在の位置でステートメントの実行を中止し、ステップで事前に定義されている位置に戻ります。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

RETURN;

引数なし

RETURN ステートメントは、DATA ステップ内の現在の位置でステートメントの実行を中止し、前の DATA ステップステートメントに制御を戻します。

詳細

戻る位置は、DATA ステップでのステートメントの実行順序によって異なります。

RETURN ステートメントは、次のステートメントと組み合わせて使用します。

- GOTO ステートメント
- FILE ステートメントの HEADER=オプション
- LINK ステートメント

DATA ステップに明示的な OUTPUT ステートメントを使用していない場合や、REMOVE ステートメントや REPLACE ステートメントが MODIFY ステートメントと組み合わせて使用されていない場合、RETURN ステートメントを使用して DATA ステップの先頭に戻ると、暗示的な OUTPUT ステートメントによって現在のオブザベーションが新しいデータセットに書き込まれます。すべての DATA ステップには、最後の実行ステートメントとして暗示的な RETURN ステートメントが含まれています。

例: 基本的な使用

この例では、X と Y の値が等しい場合、RETURN ステートメントを実行してオブザベーションをデータセットに追加します。X と Y の値が等しくない場合は、残りのステートメントを実行してからオブザベーションをデータセットに追加します。

```
data survey;
  input x y;
  if x=y then return;
  put x= y=;
  datalines;
21 25
20 20
7 17
;
```

関連項目

ステートメント:

- [“FILE ステートメント” \(104 ページ\)](#)
- [“GOTO ステートメント” \(135 ページ\)](#)
- [“LINK ステートメント” \(242 ページ\)](#)

RUN ステートメント

すでに入力されている SAS ステートメントを実行します。

注: [RUN ステートメント](#)は SAS グローバルステートメントに移動しました。

%RUN ステートメント

%INCLUDE *ステートメントの後にあるソースステートメントを終了します。

注: [%RUN ステートメント](#)は SAS グローバルステートメントに移動しました。

SASFILE ステートメント

SAS データセットを開き、ファイル全体をメモリに保持するために必要なバッファを割り当てます。

注: [SASFILE ステートメント](#)は SAS グローバルステートメントに移動しました。

SELECT ステートメント

複数のステートメントまたはステートメントグループからその 1 つを実行します。

該当要素:	DATA ステップ
カテゴリ:	CAS 制御 In-Database 処理
種類:	実行

構文

```
SELECT <(select-expression)> ;  
    WHEN-1 (when-expression-1 <..., when-expression-n> ) statement;  
    <... WHEN-n (when-expression-1 <...,when-expression-n> ) statement;>  
    <OTHERWISE statement;>  
  
END;
```

引数

(select-expression)

1 つの値に評価される任意の SAS 式を指定します。

制限事項 *select-expression* で変数が指定されている場合、*when-expression* で演算子を使用することはできません。詳細については、[“例 6: 変数の Select ステートメントでの使用” \(356 ページ\)](#)を参照してください。

参照項目: [“select-expression を指定する場合の when-expression の評価” \(353 ページ\)](#)

(when-expression)

複合式などの任意の SAS 式を指定します。SELECT では、*when-expression* を少なくとも 1 つ指定する必要があります。

制限事項 演算子は、変数が *select-expression* に表示されない限り、*when-expression* で使用できます。詳細については、[“例 6: 変数の Select ステートメントでの使用” \(356 ページ\)](#)を参照してください。

ヒント 複数の *when-expressions* をカンマで区切ることは、論理演算子 OR で区切ることに相当します。

when-expression の使われ方は、*select-expression* を指定するかどうかによって異なります。

参照項目: [“select-expression を指定しない場合の when-expression の評価” \(354 ページ\)](#)

statement

DO ステートメント、SELECT ステートメント、null ステートメントなどの SAS 実行ステートメントを指定します。*statement* 引数は必ず指定する必要があります。

詳細

SELECT グループ内の WHEN ステートメントの使用

SELECT ステートメントは、SELECT グループを開始します。SELECT グループ内に WHEN ステートメントを置き、条件が真の場合に実行する SAS ステートメントを指定します。WHEN ステートメントを SELECT グループ内に少なくとも 1 つ使用する必要があります。オプションの OTHERWISE ステートメントは、WHEN の条件が偽の場合に実行するステートメントを指定します。END ステートメントは、SELECT グループを終了します。

WHEN ステートメントに null ステートメントを使用すると、真の場合に追加アクションを伴わない条件として認識されます。OTHERWISE ステートメントに null ステートメントを使用すると、WHEN の条件がすべて偽の場合にエラーメッセージの表示を避けることができます。

select-expression を指定する場合の *when-expression* の評価

select-expression を指定する場合、*select-expression* と *when-expression* が評価されます。この 2 つの値が等しいかどうかを比較した後、真または偽の値を返します。比較結果が真ならば、*statement* を実行します。比較結果が偽ならば、現在の WHEN ステートメント内の次の *when-expression* に移動するか、他に式が指定されていない場合は次の WHEN ステートメントに移動します。WHEN ステートメントが他に存在せず、OTHERWISE ステートメントが指定されている場合は OTHERWISE ステートメントに移動します。SELECT-WHEN の比較結果がすべて偽であり、

OTHERWISE ステートメントが指定されていない場合は、エラーメッセージを表示して DATA ステップの実行を中止します。

select-expression を指定しない場合の *when-expression* の評価

select-expression を指定しない場合、*when-expression* が評価されて真または偽の結果が生成されます。結果が真ならば、*statement* を実行します。結果が偽ならば、SAS は現在の WHEN ステートメント内の次の *when-expression* に移動するか、他に式が指定されていない場合は次の WHEN ステートメントに移動します。(つまり、最初の真の WHEN ステートメントに示されたアクションを実行します。) *when-expression* が真に評価されない場合、SAS は OTHERWISE ステートメントが存在する場合は、OTHERWISE ステートメントに移動します。*when-expressions* の結果がすべて偽になり、OTHERWISE ステートメントが指定されていない場合は、エラーメッセージを表示します。複数の WHEN ステートメントで *when-expression* が真になる場合、最初の WHEN ステートメントが使用されます。*when-expression* が真になった後は、それ以降の *when-expressions* は評価されません。

%INCLUDE ファイルを使用した大量のデータの処理

大量のデータを処理する方法の 1 つとして、DATA ステップ内で %INCLUDE ステートメントを使用することが挙げられます。%INCLUDE ステートメントを使用すると、メインプログラムを管理しやすい状態に保ったまま複雑な処理を実行できます。メインプログラムで使用する %INCLUDE ファイルは、データを処理する WHEN ステートメントとその他の SAS ステートメントから構成されます。例については、“[例 5: 大量のデータの処理](#)” (355 ページ) を参照してください。

比較

ステートメントが少ないプログラムの場合には、IF-THEN/ELSE ステートメントを使用します。IF 句に指定した条件が真になるオブザベーションやレコードのみに対して処理を継続するには、サブセット化 IF ステートメントを THEN 句を指定せずに使用します。

SELECT ステートメントは SQL プロシジャの CASE ステートメントのように動作します。

例

例 1: ステートメントの使用

```
select (a);
  when (1) x=x*10;
  when (2);
  when (3,4,5) x=x*100;
  otherwise;
end;
```

例 2: DO グループの使用

```

select (payclass);
  when ('monthly') amt=salary;
  when ('hourly')
    do;
      amt=hrlywage*min(hrs,40);
      if hrs>40 then put 'CHECK TIMECARD';
    end;    /* end of do */
  otherwise put 'PROBLEM OBSERVATION';
end;      /* end of select */

```

例 3: 複合式の使用

```

select;
  when (mon in ('JUN', 'JUL', 'AUG')
    and temp>70) put 'SUMMER ' mon=;
  when (mon in ('MAR', 'APR', 'MAY'))
    put 'SPRING ' mon=;
  otherwise put 'FALL OR WINTER ' mon=;
end;

```

例 4: 等しいかを比較

```

/* INCORRECT usage to select value of 2 */
select (x);
/* evaluates T/F and compares for      */
/* equality with x                      */
  when (x=2) put 'two';
end;
/* correct usage */
select(x);
/* compares 2 to x for equality */
  when (2) put 'two';
end;
/* correct usage */
select;
/* compares 2 to x for equality      */
  when (x=2) put 'two';
end;

```

例 5: 大量のデータの処理

この例では、%INCLUDE ステートメントには、インベントリ内の新しいアイテムや古いアイテムを処理する WHEN ステートメントからなるコードが含まれます。メインプログラムには、DATA ステップの全体的なロジックが示されます。

```

data test (keep=ItemNumber);
  set ItemList;
  select;
    %include NewItems;
    %include OldItems;
    otherwise put 'Item ' ItemNumber ' is not in the inventory.';
  end;
run;

```

例 6: 変数の Select ステートメントでの使用

変数は、SELECT ステートメントで変数が指定されていなくても、*when-expression* の演算で使用できます。次に例を示します。

```
data _null_;
  h1=hour(datetime());
  select;
    when (h1<12) greetp1 = "Morning ";
  end;
  put greetp1;
run;
```

この例では、*when-expression* で指定された less than (<)演算子が、*select-expression* で指定された h1 変数を演算しようとしているため、エラーが発生します。

```
data _null_;
  h1=hour(datetime());
  select (h1);
    when (<12) greetp1 = "Morning ";
  end;
  put greetp1;
run;
```

less than (<)演算子の時点で生成される SAS ログは次のとおりです。

Syntax error, expecting one of the following: a name, a quoted string, a numeric constant, a datetime constant, a missing value, INPUT, PUT

関連項目

ステートメント:

- [“DO ステートメント” \(88 ページ\)](#)
- [“IF ステートメント: サブセット化” \(137 ページ\)](#)
- [“IF-THEN/ELSE ステートメント” \(140 ページ\)](#)

SET ステートメント

1 つ以上の SAS データセットからオブザベーションを読み込みます。

該当要素: DATA ステップ

カテゴリ: CAS

In-Database 処理

ファイル操作

種類: 実行

制限事項: In-database DATA ステップでは、複数の SET ステートメントはサポートされません。入力テーブルのデータセットオプションを含め、SET ステートメントオプショ

ンはサポートされていません。詳細については、“[DATA Step Acceleration](#)” ([SAS In-Database Products: User's Guide](#))を参照してください。

注:

CAS で実行される DATA ステップは、入力テーブルに十分な行が含まれており、データが複数のスレッドに分散されている場合は、シングルスレッドで実行する必要があります。DATA ステートメントで SINGLE=YES オプションを使用すると、DATA ステップをシングルスレッドで実行できます。

SET ステートメントを使用して読み込まれた変数は PDV で保持されます。詳細については、“[DATA ステップの定義](#)” ([SAS プログラマガイド: エッセンシャル](#))および“[RETAIN ステートメント](#)” (344 ページ)を参照してください。

構文

```
SET<SAS-data-set\(s\) <(data-set-options(s))>>
    <options>;
```

引数なし

引数を指定しない場合、SET ステートメントは最近作成されたデータセットからオブザベーションを読み込みます。

引数

[SAS-data-set \(s\)](#)

1 レベル名、2 レベル名、特殊 SAS データセット名の 1 つを指定します。

ヒント データセットのリストを指定できます。詳細については、“[SET ステートメントでデータセットリストを使用する](#)” (364 ページ)を参照してください。

データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

参照項目: SAS データセット名のレベルと各レベルの用途については、“[SAS データセットの命名規則](#)” ([SAS プログラマガイド: エッセンシャル](#))を参照してください。

例 “[例 13: データセットリストを使用する](#)” (369 ページ)

[\(data-set-options\)](#)

処理対象の変数またはオブザベーションをプログラムデータベクトルに読み込むときに実行するアクションを指定します。

ヒント データセットリストに適用するデータセットオプションは、データセットリストに存在するすべてのデータセットに適用されます。

参照項目: 入力データセットに使用できるデータセットオプションの一覧については、“[データセットオプションの定義](#)” ([SAS データセットオプション: リファレンス](#))を参照してください。

SET オプション

CUROBS=*variable*

データセットから読み込んだオブザベーション数を含む変数を作成および名前を付与

例 “例 14: 現在のオブザベーション番号を調べる” (371 ページ)

END=*variable*

作成する一時変数の名前を指定します。この変数の値にはファイル終端指示子が格納されます。この変数の値は 0 に初期化されますが、SET ステートメントに指定されている最後のデータセットから最後のオブザベーションを読み込むときに 1 に設定されます。この変数は、新しいデータセットには追加されません。

制限 END=オプションは POINT=オプションと併用できません。ランダムアクセスを使用する場合、END=に指定した変数の値は 1 に設定されません。

操作 BY ステートメントを使用する場合、インタリーブされたデータセットから最後のオブザベーションを SET ステートメントが読み込むときに END=の値が 1 に設定されます。詳細については、“[SET を使用した BY グループ処理](#)” (365 ページ)を参照してください。

例 “例 11: すべてのオブザベーション読み込み後にオブザベーションを書き込む” (368 ページ)

INDSNAME=*variable*

作成する変数の名前を指定します。この変数には、現在のオブザベーションの読み込み先となる SAS データセット名が格納されます。格納される名前はデータセット名または物理名になります。物理名には動作環境でファイルを判別できる名前を指定します。

ヒ データセット名の場合、変数の値にライブラリ名を追加した後
ン (例:WORK.PRICE)、この 2 レベル名を大文字に変換します。
ト

定義されていない場合、変数の長さは 41 バイトに設定されます。ファイル名が 41 バイトより長い場合、LENGTH ステートメントを使用すると、物理ファイル名の値を十分格納できるまで変数の長さを拡張できます。

この変数が特定の長さを指定した文字変数として定義されている場合、変数の長さは変更されません。INDSNAME 変数の値が定義した長さを超えている場合、値は切り捨てられます。

この変数が数値変数として定義されている場合はエラーが発生します。

変数は DATA ステップのどの位置にでも指定できますが、変数は出力には追加されません。

例 “例 12: 現在のオブザベーションを読み込むデータセットの名前を取得する” (368 ページ)

KEY=*index*</UNIQUE>

インデックス変数またはキーの値に基づいて、SAS データセット内のオブザベーションに非順次アクセスします。

範囲	読み込むデータセットの単一インデックス名または複合インデックス名を指定します。
制限事項	KEY=オプションは POINT=オプションと併用できません。
ヒント	SYSRC 自動呼び出しマクロと _IORC_ 自動変数を組み合わせて使用すると、エラー処理に活用できる情報をこれまでよりも多く得ることができます。SET ステートメントに KEY=オプションを使用すると、自動変数 _IORC_ が作成されます。この自動変数の値は、SAS データセットのオブザベーションに対して実行した、最新の入出力処理のステータスを示すリターンコードに設定されます。KEY=オプションの値が見つからない場合、_IORC_ 変数は SYSRC 自動呼び出しマクロのニーマニック _DSENOM_ に対応する値を返します。また、自動変数 _ERROR_ の値を 1 に設定します。 SET ステートメントを KEY=オプションと重複するインデックスで使った場合、SET ステートメントはキー値に一致する最初のオブザベーションから読み込み開始するようにした方が良い結果が得られます。KEYRESET=オプションを使って、読み込むデータセットのインデックスの先頭から KEY=オプションに指定された値の検索を開始するかどうかをコントロールします。
参照項目:	詳細については、SAS マクロ言語: リファレンスの SYSRC 自動呼出しマクロの説明を参照してください。 "KEYRESET=variable" (359 ページ) UNIQUE オプション (362 ページ)
例	"例 7: テーブルルックアップの実行" (367 ページ) "例 8: マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する" (367 ページ)
注意	KEY=オプションを使用するときは、無限ループが発生する可能性があります。 プライマリデータセットを指定せずに KEY=オプションを使用する場合、DATA ステップの処理を終了させる STOP ステートメント、または _IORC_ 自動変数と SYSRC 自動呼び出しマクロを組み合わせて使用し、_IORC_ 変数の無効な値をチェックするプログラムロジックのどちらかまたは両方を使用する必要があります。

KEYRESET=variable

読み込むデータセットのインデックスの先頭から KEY=オプションに指定された値の検索を開始するかどうかをコントロールします。KEYRESET 変数の値が 1 の場合、インデックスの一番上から検索します。KEYRESET 変数の値が 0 の場合、インデックスの検索はリセットされないで、最後の検索のつづきから検索します。

操作 KEYRESET=オプションは UNIQUE オプションに似ていますが、KEYRESET=オプションでは KEY =検索をインデックスの最初からにするかどうかを決められます。

参照項目: ["KEY=index</UNIQUE>" \(358 ページ\)](#)

["UNIQUE" \(362 ページ\)](#)

例 ["例 15: KEYRESET オプションの使用" \(371 ページ\)](#)

NOBS=variable

作成する一時変数の名前を指定します。この一時変数の値は、1 つまたは複数の入力データセットの中のオブザベーションの合計数になります。SET ステートメントに複数のデータセットを指定する場合、NOBS=オプションの変数の値は、指定した複数のデータセットに含まれるオブザベーションの合計数に等しくなります。オブザベーション数には、削除対象としてマークされていても、まだ削除されていないオブザベーションも含まれます。

制限事項 特定の SAS ビューと TAPE や XML エンジンのような順次エンジンに対しては、SAS がオブザベーション数を検知することができません。この場合、NOBS=オプションに指定した変数の値には、動作環境で使用できる最大の正の整数値が設定されます。

操作 NOBS=オプションと POINT=オプションは互いに独立しています。

ヒント コンパイル時に、各データセットのディスクリプター情報を読み込んでから、NOBS=オプションの変数の値を自動的に割り当てます。そのため、SET ステートメントの実行前にも NOBS=オプションの変数を参照できます。この変数は DATA ステップで使用できますが、出力データセットには追加されません。

例 ["例 10: 最終オブザベーションに到達するまで関数を実行する" \(368 ページ\)](#)

OPEN=(| DEFER)

処理の準備ができるまで、連結された SAS データセットを開くことを保留できません。

IMMEDIATE

コンパイル中に、SET ステートメントに指定されているすべてのデータセットを開きます。

制限事項 IMMEDIATE オプションを使用する場合、KEY=、POINT=、BY ステートメントの処理を使用できますが、使用できるのはそのうち 1 つだけです。

ヒント 後から開いたデータセットの変数の種類が最初に開いたデータセットにある同じ名前の変数の種類と異なる場合(例:最初の変数の種類が文字で後の変数の種類が数値など)、DATA ステップの処理は中止され、エラーメッセージが出力されます。

DEFER

コンパイル中に最初のデータセットを開きます。後続のデータセットは実行中に開きます。1 つの DATA ステップ内のすべてのオブザベーションの読み込みと処理が終了すると、そのオブザベーションを閉じてから、指定されている次のデータセットを開きます。

制限事項 DEFER オプションを指定する場合、KEY=ステートメントオプション、POINT=ステートメントオプション、BY ステートメントは使用できません。この 3 つによって、データセットのオブザベーションをランダムに処理するかインタリーブするかを暗示的に指定します。ただし、すべてのデータセットが開いていない場合、この処理を実行できません。

要件 変数の処理に DROP=、KEEP=、RENAME=データセットオプションを使用できますが、各データセットで処理する変数は同一の変数でなければなりません。通常、後続のデータセットで定義された変数が最初のデータセットで定義された変数と異なる場合、警告メッセージが出力されますが、処理は中止せずに行われます。

- 後から開いたデータセットの変数の種類が最初に開いたデータセットにある同じ名前の変数の種類と異なる場合(例:最初の変数の種類が文字で後の変数の種類が数値など)、DATA ステップの処理は中止され、エラーメッセージが出力されます。
- 後から開いたデータセットの変数が SET ステートメントで最初に開いたデータセットには定義されていないが、DATA ステッププログラムにあらかじめ定義されている場合、DATA ステップの処理は中止され、エラーメッセージが出力されます。この場合、これまでの反復処理でのこの変数の値が不適切である可能性があります。SET ステートメントのセマンティック動作では、最初のデータセットから最初のオブザベーションを処理する際にこの変数を欠損値に設定する必要があります。

デフォルト IMMEDIATE

POINT=variable

一時変数の名前を指定します。この変数の数値によって、読み込むオブザベーションが特定されます。POINT=オプションを指定すると、SET ステートメントはランダム(ダイレクト)アクセスを使用して SAS データセットを読み込みます。

制限事項 POINT=オプションは、BY ステートメント、WHERE ステートメント、WHERE=データセットオプションと併用できません。また、移送用形式のデータセット、テープまたはディスク上の順次形式のデータセット、SAS/ACCESS ビュー、外部ファイルからデータを読み込む SQL プロシジャビューでは、POINT=を使用することはできません。

POINT=オプションと KEY=オプションを併用することはできません。

CAS では POINT=はサポートされていません。

要 件 STOP ステートメント

注 `_N_` は繰り返しの回数で、最後に読み込まれたオブザベーションのオブザベーション番号ではありません。

ヒント `POINT=` 変数の値を入力しなければなりません。たとえば、`DO` ステートメントの一部の形式では、`POINT=` 変数をインデックス変数として使用できません。

`POINT=` オプションに指定した変数は `DATA` ステップのどの位置にでも指定できますが、新しい SAS データセットには追加されません。

例 “例 6: 1 オブザベーションと複数オブザベーションを結合する” (367 ページ)

“例 9: ダイレクトアクセスを用いたサブセットの読み込み” (368 ページ)

注意 **`POINT=` オプションを使用するときは、無限ループが発生する可能性があります。** `POINT=` オプションを使用する場合、`DATA` ステップの処理を終了させる `STOP` ステートメント、または `POINT=` オプションに指定した変数の無効な値をチェックするプログラムロジックのどちらかまたは両方を使用する必要があります。 `POINT=` オプションでは `DO` ステートメントに指定したオブザベーションだけを読み込むので、順次アクセスでファイルを読み込む場合とは違い、ファイル終端指示子を読み取ることができません。ファイル終端指示子を読み取ることによって `DATA` ステップを自動的に終了させるため、`POINT=` オプションの使用時に `DATA` ステップを終了させる別の方法を準備しないと、`DATA` ステップで無限ループが発生する場合があります。 `POINT=` オプションに指定した変数の値が無効な場合、自動変数 `_ERROR_` の値が 1 に設定されます。この情報を使用すると、`DO` ループが無限に発生する、または `STOP` ステートメントが `DATA` ステップの最後に含まれているなどの状態をチェックできます。

UNIQUE

読み込むデータセットのインデックスの先頭から `KEY=` オプションに指定された値の検索を開始します。

制限事項 `UNIQUE` は `KEY=` オプションを指定した場合にのみ指定できます。また、スラッシュを前に追加する必要があります。

注 デフォルトでは、`SET` ステートメントは、`KEY=` オプションの値が変更された場合にのみインデックスの先頭から検索を開始します。

`SET` ステートメントを繰り返し実行するときに `KEY=` の値が変更されない場合、最近取得したオブザベーションの直後から検索を開始します。 `KEY=` オプションに指定した変数の値の重複が連続して存在する場合、`SET` ステートメントは読み込み中のデータセットにある重複したインデックス付きの値に対して 1 対 1 で対応させようとしています。 `KEY=` オプションに指定した変数の値の重複が読み込み中のデータセットに存在する数よりも多い場合、それ以上の重複は検出されなかったと判断されます。

KEY=オプションに指定した変数の値が重複しない場合、最初にそのキーの値を使用してオブザベーションを読み込む場合は成功しますが、2 回目以降に同じ値を使用してオブザベーションを読み込もうとすると失敗します。この場合、_IORC_変数は、SYSRC 自動呼び出しマクロのニーマニック _DSENUM に対応する値を返します。/UNIQUE オプションを追加すると、初回以降でも KEY=オプションに指定した変数の重複しない値を使用してオブザベーションを正常に読み込むことができます。この場合、_IORC_変数は 0 を返します。

参照項目: “KEYRESET=variable” (359 ページ)

例 “例 8: マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する” (367 ページ)

詳細

SET の処理

SET ステートメントを実行するたびに、1 つのオブザベーションをプログラムデータベクトルに読み込みます。他の処理を指示しない限り、SET ステートメントでは、すべての変数とすべてのオブザベーションを入力データセットから読み込みます。SET ステートメントでは複数のデータセットを使用できます。また、1 つの DATA ステップには複数の SET ステートメントを指定できます。

注: DATA ステップがファイル終端指示子またはすべての開かれているデータセットの終わりに到達すると、シャットダウンが行われます。たとえば、SET ステートメントを FIRSTOBS と併用すると、複数のファイルの中でヘッダーレコードのみを含むファイルによって、DATA ステップの通常のシャットダウンがトリガーされます。このシャットダウンは、ファイル終端指示子を超えて読み込みを行うと DATA ステップが終了するために発生します。END=オプションを使用すると、シャットダウンを回避することができます。

使用法

SET ステートメントはプログラミング時にさまざまな方法で柔軟に使用できます。SET ステートメントをオプションやステートメントと組み合わせることにより、次のことが行えます。

- DATA ステップでさらに処理を行うために、既存の SAS データセットからオブザベーションと変数の値を読み込みます。
- データセットを連結します。データセットをインタリーブします。また、データセットの 1 対 1 の読み込みを実行します。
- ダイレクトアクセスメソッドを使用して SAS データセットを読み込みます。

SET ステートメントでデータセットリストを使用する

SET ステートメントでは、データセットリストを使用できます。データセットリストを使用すると、現在存在するデータセットのグループを簡単に示せます。このデータセットリストには、名前接頭辞リストまたは番号付き範囲リストを指定する必要があります。

名前接頭辞リストは、指定した文字列で始まるすべてのデータセットを示します。たとえば、`set SALES1;`と指定すると、"SALES1"で始まる SALES1、SALES10、SALES11、SALES12 などのデータセットがすべて読み込まれます。

番号付き範囲リストでは、連番である最後の文字を除き、同じ名前のデータセットが存在する必要があります。番号付き範囲リストでは、開始値と終了値に任意の数値を使用してもかまいません。たとえば、次のリストは同じデータセットを参照しています。sales1 sales2 sales3 sales4 sales1-sales4

注: 最初のデータセット名にある数値接尾辞の先頭に 0 が置かれる場合、最後のデータセット名の数値接尾辞の桁数は最初のデータセット名の桁数と一致するか、または上回る必要があります。このように指定されていない場合はエラーが発生します。たとえば、データセットリストを sales001-sales99 または sales01-sales9 と指定するとエラーが発生します。ただし、このデータセットリストの有効な指定は、sales001-sales999 です。最初のデータセット名にある数値接尾辞の先頭に 0 が置かれない場合、最初と最後のデータセット名に使用する数値接尾辞の桁数を一致させる必要はありません。たとえば、データセットリストの指定として sales1-sales999 は有効です。

番号付きデータセットリストを使用する場合に考慮すべきその他のルールは次のとおりです。

- 範囲のグループを複数指定できます。

```
set cost1-cost4 cost11-cost14 cost21-cost24;
```
- 番号付き範囲リストと名前接頭辞リストを組み合わせで指定できます。

```
set cost1-cost4 cost2: cost33-37;
```
- 個々のデータセットとデータセットリストを組み合わせで指定できます。

```
set cost1 cost10-cost20 cost30;
```
- データセットリストを囲んだ引用符は無視されます。

```
/* these two lines are the same */
set sales1 - sales4;
set 'sales1'n - 'sales4'n;
```
- データセット名のブランクは無効です。引用符を使用する場合、末尾にあるブランクは無視されます。

```
/* blanks in these statements will cause errors */
set sales 1 - sales 4;
set 'sales 1'n - 'sales 4'n;
/* trailing blanks in this statement will be ignored */
set 'sales1 'n - 'sales4 'n;
```
- 数値接尾辞に使用できる最大数は、2147483647 です。

```
/* this suffix will cause an error */
set prod2000000000-prod2934850239;
```


SET を使用した BY グループ処理

DATA ステップでは、1 つの BY ステートメントのみ、SET ステートメントに指定できます。BY ステートメントは適用する SET ステートメントの直後に指定する必要があります。SET ステートメントにリストされるデータセットは、BY ステートメントにリストされる変数の値に基づいて並べ替えられているか、適切なインデックスが含まれている必要があります。SET ステートメントは BY ステートメントと使用すると、複数のデータセットをインタリーブします。新しいデータセットのオブザベーションは 1 つまたは複数の BY 変数の値に従って並べられます。BY グループ内は、元のデータセットの順序に従って並べられます。SET ステートメントでの BY グループの処理の例については、“[例 2: SAS データセットのインタリーブ](#)” (366 ページ) を参照してください。

SAS データセットの結合

1 つの SET ステートメントに複数のデータセットを指定すると、指定したデータセットが連結されます。新しいデータセットのオブザベーション数は、指定した複数のデータセットにあるオブザベーション数の合計になります。オブザベーションは、最初のデータセットのすべてのオブザベーションの後ろに、2 番目のデータセットのすべてのオブザベーションが続くという順序で配置されます。データセットの連結の例については、“[例 1: SAS データセットの連結](#)” (366 ページ) を参照してください。

1 つの SET ステートメントを BY ステートメントとともに使用すると、指定したデータセットがインタリーブされます。新しいデータセットのオブザベーションは 1 つまたは複数の BY 変数の値に従って並べられます。BY グループ内は、元のデータセットの順序に従って並べられます。データセットのインタリーブの例については、“[例 2: SAS データセットのインタリーブ](#)” (366 ページ) を参照してください。

複数の SET ステートメントを使用すると、指定したデータセットに対して 1 対 1 の読み込み(1 対 1 マッチともいう)を行います。新しいデータセットには、すべての入力データセットの変数が格納されます。新しいデータセットに含まれるオブザベーションの数は、元のデータセットのうち最も小さなデータセットに含まれるオブザベーションの数になります。複数のデータセットに共通する変数が含まれている場合、前のデータセットから読み込んだ値を最後のデータセットから読み込んだ値で置き換えます。データセットの 1 対 1 の読み込みの例は、次を参照してください。

- “[例 6: 1 オブザベーションと複数オブザベーションを結合する](#)” (367 ページ)
- “[例 7: テーブルルックアップの実行](#)” (367 ページ)
- “[例 8: マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する](#)” (367 ページ)

データセットを準備する方法の詳細については、“[Combining SAS Data Sets: Basic Concepts](#)” ([SAS Language Reference: Concepts](#)) を参照してください。

比較

- SET ステートメントでは、既存の SAS データセットからオブザベーションを読み込みます。INPUT ステートメントでは、SAS 変数とオブザベーションを作成するために、外部ファイルまたはインストリームデータ行から生データを読み込みます。

- SET ステートメントで KEY=オプションを使用すると、インデックスの値に基づいて、SAS データセットのオブザベーションに非順次アクセスできます。SET ステートメントで POINT=オプションを使用すると、オブザベーションの番号に基づいて、SAS データセットのオブザベーションに非順次アクセスできます。

例

例 1: SAS データセットの連結

SET ステートメントに複数のデータセット名を指定すると、ステートメント実行後には、指定したすべてのデータセットを連結した出力データセットが作成されます。最初のデータセットからすべてのオブザベーションを読み込んだ後、2 番目のデータセットからすべてのオブザベーションを読み込むというように、すべてのデータセットからすべてのオブザベーションが読み込まれます。この例では、3 つのデータセットを連結して、FITNESS という 1 つの出力データセットを作成します。

```
data fitness;
  set health exercise well;
run;
```

例 2: SAS データセットのインタリーブ

2 つ以上の SAS データセットをインタリーブするには、SET ステートメントの後ろに BY ステートメントを使用します。

```
data april;
  set payable recvable;
  by account;
run;
```

例 3: SAS データセットの読み込み

次の DATA ステップでは、データセット NC.MEMBERS にある各オブザベーションをプログラムデータベクトルに読み込みます。次に、CITY の値が Raleigh であるオブザベーションのみを新しいデータセット RALEIGH.MEMBERS に出力します。

```
data raleigh.members;
  set nc.members;
  if city='Raleigh';
run;
```

例 4: 1 つのオブザベーションを SAS データセットのすべてのオブザベーションとマージする

既存のデータセットにマージできるオブザベーションは、SAS プロシジャまたは他の DATA ステップで作成できます。この例では、データセット AVGSALES にはオブザベーションが 1 つだけ含まれます。

```
data national;
  if _n_=1 then set avgsales;
  set totsales;
run;
```


例 5: 同一データセットを複数回読み込む

この例の SET ステートメントは、個別に処理されます。つまり、1 つのデータセットからの読み込みを、あたかも 2 つのデータセットからの読み込みのように処理されます。

```
data drugxyz;
  set trial5(keep=sample);
  if sample>2;
  set trial5;
run;
```

DATA ステップの反復時、最初の SET ステートメントからオブザベーションが 1 つ読み込みます。最初の SET ステートメントの実行後、次のオブザベーションが読み込まれます。DATA ステップの同じ反復内で、それぞれの SET ステートメントから別のオブザベーションが読み込まれます。

例 6: 1 オブザベーションと複数オブザベーションを結合する

ダイレクトアクセスメソッドを使用すると、1 つのデータセットから読み込んだ複数のオブザベーションをサブセットし、それを他のデータセットから読み込んだオブザベーションと結合できます。

```
data south;
  set revenue;
  if region=4;
  set expense point=_n_;
run;
```

例 7: テーブルルックアップの実行

この例では、KEY=オプションを使用して、テーブルルックアップを実行します。この DATA ステップでは、INVTORY という名前のプライマリデータセットと、PARTCODE という名前のルックアップデータセットを読み込みます。DATA ステップはインデックス PARTNO を使用し、それぞれのデータセットで PARTNO 値間で一致する値を探しながら、PARTCODE を非順次で読み込みます。これは、プライマリデータセット INVTORY に登録されているすべてのパーツについて、ルックアップデータセット PARTCODE の変数 DESC のみに含まれる適切な説明を読み込むのが目的です。

```
data combine;
  set invtory(keep=partno instock price);
  set partcode(keep=partno desc) key=partno;
run;
```

例 8: マスターファイルに重複するオブザベーションが含まれる場合、テーブルルックアップを実行する

この例でも、KEY=オプションを使用して、テーブルルックアップを実行します。この DATA ステップでは、インデックス PARTNO が格納された INVTORY という名前のプライマリデータセットと、PARTCODE という名前のルックアップデータセットを読み込みます。PARTCODE の変数 NEW_STK には、新しい在庫量が含まれます。UNIQUE オプションを使用することにより、INVTORY に重複する値を持つ変数 PARTNO が存在する場合、重複する値を持つグループの最初のオブザベーションにのみ NEW_STK の値を追加します。

```

data combine;
  set partcode(keep=partno new_stk);
  set invtory(keep=partno instock price)
  key=partno/unique;
  instock=instock+new_stk;
run;

```

例 9: ダイレクトアクセスを用いたサブセットの読み込み

次のステートメントでは、オブザベーション番号を指定してダイレクトアクセスを行う POINT=オプションを使用し、データセット DRUGTEST から 50 オブザベーションのサブセットを選択します。

```

data sample;
  do obsnum=1 to 100 by 2;
    set drugtest point=obsnum;
    if _error_ then abort;
    output;
  end;
  stop;
run;

```

例 10: 最終オブザベーションに到達するまで関数を実行する

次のステートメントでは、NOBS=オプションを使用し、DO ループ処理を終了させる値を設定します。一時変数 LAST の値は、SURVEY1 および SURVEY2 のオブザベーション数の合計になります。

```

do obsnum=1 to last by 100;
  set survey1 survey2 point=obsnum nobs=last;
  output;
end;
stop;

```

例 11: すべてのオブザベーション読み込み後にオブザベーションを書き込む

この例では、END=変数 LAST を使用して変数 REVENUE に値を割り当てるように指示します。また、RENTAL の最後のオブザベーションの読み込みを終えてから、オブザベーションを出力します。

```

set rental end=last;
totdays + days;
if last then
  do;
    revenue=totdays*65.78;
    output;
  end;

```

例 12: 現在のオブザベーションを読み込むデータセットの名前を取得する

この例では、3 つのデータセットを作成し、dsn という名前の変数にデータセット名を格納します。名前は 3 つの部分に分割され、次の結果が出力されます。

```

/* Create some data sets to read */
data gas_price_option; value=395; run;
data gas_rbid_option; value=840; run;
data gas_price_forward; value=275; run;
/* Create a data set D */
data d;
  set gas_price_option gas_rbid_option gas_price_forward indsname=dsn;
  /* split the data set names into 3 parts */
  commodity = scan (dsn, 2, ".");
  type = scan (dsn, 3, ".");
  instrument = scan (dsn, 4, ".");
run;
proc print data=d;
run;

```

アウトプット 2.38 3つの部分に分割されたデータセット名

Obs	value	commodity	type	instrument
1	395	GAS	PRICE	OPTION
2	840	GAS	RBID	OPTION
3	275	GAS	PRICE	FORWARD

例 13: データセットリストを使用する

この例では、番号付き範囲リストを使用してデータセットを読み込みます。

```

data dept008; emp=13; run;
data dept009; emp=9; run;
data dept010; emp=4; run;
data dept011; emp=33; run;
data _null_;
  set dept008-dept010;
  put _all_;
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。

例のコード 2.2 SET ステートメントでのデータセットリストの使用

```

1 data dept008; emp=13; run;
NOTE: The data set WORK.DEPT008 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.06 seconds
      cpu time     0.03 seconds

2 data dept009; emp=9; run;
NOTE: The data set WORK.DEPT009 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

3 data dept010; emp=4; run;
NOTE: The data set WORK.DEPT010 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

4 data dept011; emp=33; run;
NOTE: The data set WORK.DEPT011 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

5
6 data _null_;
7   set dept008-dept010;
8   put _all_;
9 run;
emp=13 _ERROR_=0 _N_=1
emp=9 _ERROR_=0 _N_=2
emp=4 _ERROR_=0 _N_=3
NOTE: There were 1 observations read from the data set WORK.DEPT008.
NOTE: There were 1 observations read from the data set WORK.DEPT009.
NOTE: There were 1 observations read from the data set WORK.DEPT010.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

```

加えて、欠損しているデータセットを探すのにデータセットリストを使用することもできます。この例では、番号付き範囲リストを使用して欠損しているデータセットを探します。存在しないデータセットごとにエラーが発生します。どのデータセットが欠損しているかがわかったら、実際に存在するデータセットを反映するように SET ステートメントを修正できます。

```

data dept008; emp=13; run;
data dept009; emp=9; run;
data dept011; emp=4; run;
data dept014; emp=33; run;
data _null_;
  set dept008-dept014;
  put _all_;
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。

例のコード 2.3 SET ステートメントを使用した欠損データセットの検索

```

1  data dept008; emp=13; run;
NOTE: The data set WORK.DEPT008 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.04 seconds
      cpu time     0.04 seconds

2  data dept009; emp=9; run;
NOTE: The data set WORK.DEPT009 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

3  data dept011; emp=4; run;
NOTE: The data set WORK.DEPT011 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.03 seconds
      cpu time     0.01 seconds

4  data dept014; emp=33; run;
NOTE: The data set WORK.DEPT014 has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

5  data _null_;
6  set dept008-dept014;
ERROR: File WORK.DEPT010.DATA does not exist.
ERROR: File WORK.DEPT012.DATA does not exist.
ERROR: File WORK.DEPT013.DATA does not exist.
7  put _all_;
8  run;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: DATA statement used (Total process time):
      real time    0.00 seconds
      cpu time     0.00 seconds

```

例 14: 現在のオブザベーション番号を調べる

次の例では、CUROBS オプションを使って現在のオブザベーションの番号を返すようにしています。

```

data women;
  set sashelp.class curobs=cobs;
  where sex = 'F';
  orig_obs = cobs;
run;

```

例 15: KEYRESET オプションの使用

この例では、KEYRESET=オプションを使って、I=3 が 2 回の時にすべての値を検索するようにしています。

```

data a(index=(i));
  do i = 1,2,3,3,3,4,5;
    j=ranuni(4);
    output;
  end;
run;
data _null_;
  input i;

```

```

reset = 1;
do while (_iorc_ = 0);
  set a key=i keyreset=reset;
  put _all_;
end;
_error_ = 0; _iorc_ = 0;
datalines;
3
3
;

```

関連項目

- [“データセットオプションの定義” \(SAS データセットオプション: リファレンス\)](#)
- [“Reading, Combining, and Modifying SAS Data Sets” \(SAS Language Reference: Concepts\)](#)
- [“Rules for Words and Names in the SAS Language” \(SAS Language Reference: Concepts\)](#)
- [SAS マクロ言語: リファレンス](#)

ステートメント:

- [“BY ステートメント” \(57 ページ\)](#)
- [“DO ステートメント” \(88 ページ\)](#)
- [“INPUT ステートメント” \(186 ページ\)](#)
- [“MERGE ステートメント” \(252 ページ\)](#)
- [“STOP ステートメント” \(372 ページ\)](#)
- [“UPDATE ステートメント” \(380 ページ\)](#)

SKIP ステートメント

SAS ログ内にブランク行を 1 行作成します。

注: [SKIP ステートメント](#)は SAS グローバルステートメントに移動しました。

STOP ステートメント

現在の DATA ステップの実行を中止します。

該当要素: DATA ステップ

カテゴリ: アクション

種類:

CAS
In-Database 処理
実行

構文

STOP;

詳細

STOP ステートメントでは、現在の DATA ステップの処理を中止し、現在の DATA ステップ以降のステートメントから処理を再開します。

現在の DATA ステップのデータセットが書き出されます。しかし、STOP の実行時に処理されていたオブザベーションは追加されません。STOP ステートメントはそれだけでも、IF-THEN ステートメントまたは SELECT グループと組み合わせても使用できます。

STOP ステートメントは、SET ステートメント内の POINT=オプションなど、ランダムアクセスを使用して SAS データセットを読み込む機能と一緒に使用します。ランダムアクセスでは、EOF を検出できません。そのため、DATA ステップの無限ループを回避するプログラムステートメントを指定する必要があります。

比較

- ウィンドウ環境または他の対話型モードの操作を使用している場合、ABORT ステートメントおよび STOP ステートメントの両方で処理を中止することができます。ABORT ステートメントでは自動変数 _ERROR_ の値を 1 に設定しますが、STOP ステートメントでは設定しません。
- バッチモードまたは非対話型モードでは、この 2 つのステートメントの結果は異なります。バッチモードまたは非対話型モードでは STOP ステートメントを使用して、処理を次の DATA ステップまたは PROC ステップから再開します。

例

例 1: 基本的な使用

- stop;
- if idcode=9999 then stop;
- select (a);
 when (0) output;
 otherwise stop;

```
end;
```

例 2: 無限ループの回避

この例は、ランダムアクセスを使用している場合に、STOP を使用して DATA ステップでの無限ループを回避する方法を示しています。

```
data sample;
  do sampleobs=1 to totalobs by 10;
    set master.research point=sampleobs nobs=totalobs;
    output;
  end;
  stop;
run;
```

関連項目

ステートメント:

- [“ABORT ステートメント” \(31 ページ\)](#)
- [SET ステートメントの POINT=オプション \(361 ページ\)](#)

合計 ステートメント

式の計算結果を合計変数に加算します。

該当要素:	DATA ステップ
カテゴリ:	アクション
	CAS
	In-Database 処理
種類:	実行
例:	<pre>data Table2; set Table1; total + x; run;</pre>

構文

variable+*expression*;

引数

variable

数値が含まれる合計変数の名前を指定します。

この変数の値は、最初のオブザベーションを読み込む前に自動的に 0 に設定されます。RETAIN ステートメントと同じように、合計変数の値は繰り返しの際に保持されます。

合計変数を 0 以外の値に初期化するには、この変数を RETAIN ステートメントで使用し、初期値を割り当てます。

expression

SAS 式を指定します。

ヒント 式を計算し、結果を合計変数に加算します。

式の計算結果が欠損値になる場合、0 として扱われます。

比較

合計ステートメントは、SUM 関数と RETAIN ステートメントを次のように使用することと同じです。

```
retain variable 0;
variable=sum(variable,expression);
```

例

さまざまな式を使用する合計ステートメントの例を次に示します。各例では、アキュムレータ変数が強調表示されています。

- data Table2;
set Table;
Total + x;
run;
- data AccountBal;
set Account;
retain Balance 1000;
Balance + (-Debit);
run;
- data Table2;
set Table;
SumXsq + x * x;
run;
- data Table2;
set Table;
nx + (x ne .);
run;
- data AccountBal2;
set AccountBal;
if Balance <=500 then Alert + 1;
run;

関連項目

関数:

- [“SUM 関数” \(SAS 関数と CALL ルーチン: リファレンス\)](#)

ステートメント:

- [“RETAIN ステートメント” \(344 ページ\)](#)

その他のドキュメント

- [“行間依存関係とマルチスレッド処理” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)
- [“テーブル全体での変数の合計” \(SAS Cloud Analytic Services: DATA ステッププログラミング\)](#)

SYSECHO ステートメント

グローバルステートメントの完了イベントを送信し、テキスト文字列を IOM クライアントに戻します。

注: [SYSECHO ステートメント](#)は SAS グローバルステートメントに移動しました。

SYSTASK ステートメント

SAS セッションから、非同期タスクを実行、リスト、または終了する操作環境コマンドを発行します。

該当要素: 任意の場所

制限事項: SYSTASK ステートメントでは、XCMD システムオプションが有効になっている必要があります。デフォルトでは、XCMD システムオプションは無効になっています。システム管理者は、XCMD システムオプションを有効にするようにサーバーを構成する必要があります。

このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

注: SAS Viya では、SYSTASK コマンドは Kubernetes コンテナ環境内で実行され、基礎となるオペレーティングシステムへの直接アクセスを抽象化します。したがって、SYSTASK コマンドは、そのコマンドが実行されるコンピュータサーバーコンテナ内でしかタスクを生成できず、そのコンテナで利用可能なリソースによって制限されます。ユーザーは、この動作が SAS 9 または Viya 3.5 とは異なることを認識し、それに応じてワークロードを計画する必要があります。

構文

```
SYSTASK COMMAND "operating environment command"<WAIT |  
NOWAIT><TASKNAME=taskname><MNAME=name-variable><STATUS=status-  
variable><SHELL="shell-command"><CLEANUP>;
```

```
SYSTASK LIST<_ALL_|taskname><STATE><STATVAR>;
```

```
SYSTASK KILL taskname<taskname>;
```

必須引数

COMMAND

operating-environment-command を実行します。

LIST

特定のアクティブタスクまたはシステム内のすべてのアクティブタスクをリストします。タスクが実行中であるか、完了し、WAITFOR ステートメントを使用して待機していない場合、タスクはアクティブです。

KILL

指定されたタスクを強制終了します。

operating-environment-command

Linux コマンドの名前(コマンド固有のオプションを含む)または X ウィンドウシステムアプリケーションの名前を指定します。コマンドを単一引用符または二重引用符で囲みます。コマンド固有のオプションで引用符が必要な場合は、オプションごとに引用符を繰り返します。たとえば、次のようになります。

```
SYSTASK COMMAND "mv ~usr/file.txt /tmp/file.txt" shell;
```

この例では、SHELL オプションを使用することで、~usr パスが実行時に展開され、直接実行されません。

注: *operating-environment-command* は、キーボードからの入力を要求できません。

ヒント SHELL オプションを使用している場合でも SHELL 別名を使用するとエラーが発生する場合、シェルはシェル初期化ファイルを処理していません。SHELL 別名でなく実際の SHELL コマンドを使用します。

オプション引数

ALL

システム内のすべてのアクティブなタスクを指定します。

CLEANUP

タスクが完了したときにタスクを LISTTASK 出力から削除するように指定します。タスクが削除されると、WAITFOR ステートメントを発行せずにタスク名を再利用できます。

SYSTASK コマンドを複数回使用して長時間実行されるジョブがある場合、WAITFOR ステートメントまたは、SYSTASK コマンドの CLEANUP オプションを

使用して、メモリをクリアします。WAITFOR ステートメントは、SYSTASK コマンドによって開始され、完了したすべてのプロセスの情報を削除することによりメモリを解放します。CLEANUP オプションでは、特定のジョブの完了時にメモリをクリアし、今後の使用のためにメモリを解放します。WAITFOR ステートメントをジョブの完了後に使用する場合、ジョブはステートメントオプションによってすでにクリーンアップされているため、無効になります。

MNAME=name-variable

SYSTASK でタスク用に自動生成されたタスク名を格納するマクロ変数を指定します。TASKNAME オプションと MNAME オプションの両方を指定すると、SYSTASK は、TASKNAME で指定した名前を、MNAME で指定した変数にコピーします。

SHELL="shell-command"

operating-environment-command をオペレーティングシステムのシェルコマンドで実行するように指定します。シェルは、operating-environment-command に含まれるシェル特殊文字を展開します。shell-command を指定すると、SYSTASK は指定したシェルコマンドを使用してシェルを呼び出します。それ以外の場合、SYSTASK はデフォルトのシェルを使用します。シェルコマンドを引用符で囲みます。

注: SHELL オプションは、指定した SHELL コマンドが `-i` オプションを使用してステートメントを渡すことを前提としています。通常、シェルコマンドは `sh`、`csh`、`ksh`、または `bash` です。

STATE

タスクのステータスを表示します。開始失敗、実行中、または完了のいずれかになります。

STATUS=status-variable

SYSTASK でタスクのステータスを格納するマクロ変数を指定します。ステータス変数名は、すべてのアクティブなタスクの中で一意である必要があります。

STATVAR

タスクに関連付けられているステータス変数を表示します。ステータス変数は、SYSTASK COMMAND ステートメントの STATUS オプションで割り当てた変数です。

TASKNAME=taskname

タスクを識別する名前を指定します。タスク名は、すべてのアクティブなタスクの中で一意である必要があります。タスクが実行中であるか、完了し、WAITFOR ステートメントを使用して待機していない場合、タスクはアクティブです。タスク名が重複すると、SAS ログにエラーが生成されます。タスク名を指定しなかった場合、SYSTASK が自動的に名前を生成します。タスク名に空白文字が含まれている場合は、引用符で囲みます。

タスク名は、タスクに対して WAITFOR ステートメントを発行するか、CLEANUP オプションを指定しない限り、タスクが完了していても再利用できません。

WAIT | NOWAIT

タスクが完了するまで SYSTASK COMMAND が現在の SAS セッションの実行を中断するかどうかを指定します。NOWAIT がデフォルトです。NOWAIT オプションで開始するタスクの場合、必要に応じて WAITFOR ステートメントを使用することで、タスクが完了するまで SAS セッションの実行を中断できます。

詳細

SYSTASK ステートメントを使用すると、SAS セッションまたはアプリケーション内からホスト固有のコマンドを実行できます。X ステートメントとは異なり、SYSTASK ステートメントはこれらのコマンドを非同期タスクとして実行します。つまり、これらのタスクは現在実行中の他のすべてのタスクとは独立して実行されます。非同期タスクはバックグラウンドで実行されるため、非同期タスクの実行中に追加のタスクを実行できます。たとえば、新しいシェルを開始し、そのシェルで Linux `cp` コマンドを実行するには、次のステートメントを使用します。

```
systask command "cp /tmp/sas* ~/archive/" taskname="copyjob1"
               status=copysts1 shell;
```

`cp` コマンドからのリターンコードは、マクロ変数 `COPYSTS1` に保存されます。

このコマンドからの出力は、SAS ログに表示されます。

UNIX と PC の間の構文は異なる可能性があるため、PC の SAS ジョブを Linux で実行するように変換すると、変換プロセスでエラーが発生することがあります。たとえば、次のコマンドを入力すると、エラーが発生します。

```
systask command "md directory-name" taskname="mytask";
```

`md` は、PC では `"make directory"` コマンドですが、UNIX では意味を持たないため、エラーが発生します。変換処理では、`md` は `mkdir` になります。`mkdir` は Linux シェルに組み込まれており、PC 上の場合と同様に別個のコマンドではないため、SYSTASK ステートメントで `SHELL` オプションを使用する必要があります。

SAS はエラーメッセージをログに書き込みます。

注: SAS アプリケーションでの SYSTASK ステートメントに続くプログラムステップは、通常、SYSTASK ステートメントが正常に実行されるかどうかによって異なります。そのため、一部の SYSTASK ステートメントの構文エラーにより、SAS アプリケーションが異常終了する可能性があります。

SAS から開始できる非同期プロセスには次の 2 種類があります。

タスク

タスク SYSTASK COMMAND で始まる TaskAll は、タスクの種類です。これらのタスクでは、`STATVAR` または `STATE` を指定しないと、SYSTASK LIST には、タスク名、種類、状態、およびステータスマクロ変数の名前が表示されます。SYSTASK KILL を使用して、種類が Task のタスクのみを強制終了できます。

SAS/CONNECT プロセス

`SIGNON` ステートメントまたはコマンドと `RSUBMIT` ステートメントを使用して SAS/CONNECT から開始されたタスクは、SAS/CONNECT プロセスの種類です。SAS/CONNECT プロセスを表示するには、`LISTTASK` ステートメントを使用して、タスク名、種類、および状態を表示します。SAS/CONNECT プロセスを終了するには、`KILLTASK` ステートメントを使用します。SAS/CONNECT プロセスの詳細については、SAS/CONNECT ユーザーガイドを参照してください。

注: タスク(SAS/CONNECT プロセスだけでなく)を表示する場合、SYSTASK LIST ではなく `LISTTASK` ステートメントを使用する方法が推奨されます。タスクを終了する

場合、SYSTASK KILL のかわりに KILLTASK ステートメントを使用する方法が推奨されます。

SYSRC マクロ変数には、SYSTASK ステートメントのリターンコードが含まれます。STATUS オプションで指定するステータス変数には、SYSTASK COMMAND で開始されたプロセスのリターンコードが含まれます。タスクが正常に実行されることを確認するには、SYSTASK ステートメントのステータスと、SYSTASK ステートメントによって開始されるプロセスのステータスの両方をモニターする必要があります。

SYSTASK ステートメントを正常に実行できない場合、SYSRC マクロ変数にゼロ以外の値が格納されます。たとえば、タスクを完了するのに十分なリソースがない場合や、SYSTASK ステートメントに構文エラーが含まれている場合があります。SYSTASK KILL ステートメントでは、1 つ以上のプロセスを強制終了できない場合、SYSRC にはゼロ以外の値が設定されます。

タスクが開始されると、そのステータス変数は NULL に設定されます。各タスクのステータス変数を使用して、どのタスクが完了に失敗したかを判断できます。ステータス変数が NULL のタスクは実行を完了しませんでした。タスクが異常終了した場合、そのステータス変数は-1 に設定されます。

SYSTASK ステートメントを使用して新しい対話型セッションを開始することはできません。

TITLE ステートメント

SAS 出力に使用するタイトルを指定します。

注: [TITLE ステートメント](#)は SAS グローバルステートメントに移動しました。

UPDATE ステートメント

トランザクションを適用してファイルを更新します。

該当要素: DATA ステップ

カテゴリ: ファイル操作

種類: 実行

制限事項: このステートメントは CAS で実行する DATA ステップではサポートされていません。

このステートメントは、In-database DATA ステップではサポートされていません。詳細については、[“DATA Step Acceleration” \(SAS In-Database Products: User's Guide\)](#)を参照してください。

注: UPDATE ステートメントを使用して読み込まれた値は PDV で保持されます。詳細については、[“DATA ステップ” \(SAS プログラマガイド: エッセンシャル\)](#)および [“RETAIN ステートメント” \(344 ページ\)](#)を参照してください。

注意: UPDATE ステートメントを使用する際に OUTPUT ステートメントを追加すると、望ましくない結果が生成されます。

構文

UPDATE *master-data-set*<(*data-set-options*)> *transaction-data-set*<(*data-set-options*)>

<END=*variable*>

<UPDATEMODE=MISSINGCHECK | NOMISSINGCHECK>;

BY *by-variable*;

引数

master-data-set

マスターファイルとして使用する SAS データセットを指定します。

範囲 この名前は 1 レベル名(例:FITNESS)、2 レベル名(例:IN.FITNESS)、および特殊 SAS データセット名のいずれでもかまいません。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

参照 [“Rules for Words and Names in the SAS Language” \(SAS Language Reference: Concepts\)](#)

(data-set-options)

DATA ステップに処理する変数を読み込むときに実行するアクションを指定します。

要件 *Data-set-options* は SAS データセット名の後に丸かっこで囲んで指定する必要があります。

ヒント データセットを更新する場合、変数の除外、変数の指定、変数名の変更を実行すると便利な場合があります。同じ名前の変数名を変更すると、同じ名前を持つ最初の変数の値が、同じ名前を持つ 2 番目の変数の値で上書きされるのを回避することができます。このように変数名の 1 つを変更すると、比較などの処理で両方の変数の値を使用できるようになります。

参照 入力データセットで使用するデータセットオプションのリストについては、次のドキュメントを参照してください。SAS データセットオプション: リファレンス

例 [“例 2: 変数名を変更する更新” \(386 ページ\)](#)

transaction-data-set

マスターデータセットに適用する変更を含む SAS データセットを指定します。

範囲 この名前は 1 レベル名(例:HEALTH)、2 レベル名(例:IN.HEALTH)、および特殊 SAS データセット名のいずれでもかまいません。

ヒント データセット名を使用するかわりに、オペレーティングシステムでサポートされている構文を使用してファイルの物理パス名を指定することができます。物理パス名は単一引用符または二重引用符で囲む必要があります。

END=variable

作成する一時変数の名前を指定します。この変数の値にはファイル終端指示子が格納されます。この変数の値は 0 に初期化され、UPDATE ステートメントで最後のオブザベーションを処理するときに 1 に設定されます。この変数はどのデータセットにも追加されません。

UPDATERODE=MISSINGCHECK**UPDATERODE=NOMISSINGCHECK**

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある既存の変数の値を置き換えるかどうかを指定します。

MISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えないように指示します。これはデフォルト設定です。

NOMISSINGCHECK

トランザクションデータセットにある変数の欠損値を使用して、マスターデータセットにある変数の値を置き換えるように指示します。

例 data newpay;
 update payroll increase updatemode=nomissingcheck;
 by id;
 run;

デフォルト MISSINGCHECK

ヒント ただし、特殊欠損値は例外で、MISSINGCHECK(デフォルト)が有効な場合でも、マスターデータセットの値と値と置き換えられます。

詳細

要件

- UPDATE ステートメントは、必ず BY ステートメントとともに指定します。BY ステートメントには、オブザベーションを識別するための変数を指定します。
- BY ステートメントは適用する UPDATE ステートメントの直後に指定する必要があります。
- UPDATE ステートメントにリストされているデータセットは、BY ステートメントにリストされている変数の値でソートされているか、適切なインデックスを含んでいる必要があります。
- マスターデータセットの各オブザベーションには、重複しない BY 変数の値が含まれている必要があります。複数の BY 変数の値がある場合、その値の最初のオブザベーションのみが更新されます。トランザクションデータセットには、同一の BY 変数の値を持つ複数のオブザベーションが含まれていてもかまいません。(複数のトランザクションオブザベーションは、出力ファイルに書き込まれる前に、マスターオブザベーションにすべて適用されます。)

詳細については、[“How to Prepare Your Data Sets” \(SAS Language Reference: Concepts\)](#)を参照してください。

トランザクションデータセット

通常、マスターデータセットとトランザクションデータセットには同じ変数が含まれています。ただし、処理時間を短縮するために、更新対象の変数のみを含むトランザクションデータセットを作成することができます。また、トランザクションデータセットには出力データセットに追加する新しい変数を含めることもできます。

出力データセットには、マスターデータセット内のオブザベーションごとに 1 つのオブザベーションが含まれます。トランザクションデータセットにマスターデータセットと一致しないオブザベーションが存在する場合、一致しないオブザベーションは新しいオブザベーションとして出力データセットに書き込まれます。更新対象ではないオブザベーションはトランザクションデータセットから省くことができます。詳細については、“[BY-Group Processing in the DATA Step](#)” (*SAS Language Reference: Concepts*)を参照してください。

欠損値

デフォルトでは、UPDATEMODE=MISSINGCHECK オプションが有効です。そのため、トランザクションデータセットの欠損値によって、マスターデータセットの既存の値が置き換えられることはありません。トランザクションデータに含まれる欠損値でマスターデータセットに含まれる既存の値を置き換える場合は、UPDATE ステートメントで UPDATEMODE=NOMISSINGCHECK オプションを指定します。

```
data newpay;
  update payroll increase updatemode=nomissingcheck;
  by id;
run;
```

欠損値を含む一部の変数のみをマスターデータセットの値と置き換える場合は、トランザクションデータセットに [MISSING ステートメント](#)を指定します。MISSING ステートメントを使用すると、特殊な欠損値を表す文字を割り当てることができます。

- トランザクションデータセットで MISSING ステートメントを指定して特殊文字の値を定義し、欠落している値を MISSING ステートメントで定義された特殊文字に置き換えます。
- MISSING ステートメントの有効な特殊文字値は、任意の単一文字 A~Z、a~z、またはアンダースコア文字(_)です。[UPDATEMODE=MISSINGCHECK](#) が有効な場合でも、トランザクションデータセットで特別な欠損値として指定された欠損値は、マスターデータセットの元の値を置き換えます。
- 出力データセット内の文字 a~z は大文字に変換されます。
- 文字 A-Z または a-z で指定された特別な欠損値は、出力データセットにその文字で表示されます。
- アンダースコア(_)で指定された特別な欠損値は、出力データセットで通常の欠損値として表示されます。つまり、ピリオド(.)で指定されます。

表 2.6 DATA ステップで MISSING ステートメントを使用した特殊欠損値の定義

欠損文字の指定	例	結果生じる更新出力																
A-Z	<pre>data master; input Item \$ Num; datalines; boot 12 shoe 10 sock 11 ; data trans; input Item \$ Num; missing Z; datalines; boot 9 shoe Z sock 11 ; data new; update master trans; by Item; run;</pre>	トランザクションデータセットの任意の文字 A-Z として定義されている欠損値はマスターデータセットの値を置き換えます。 <table><tr><th>Obs</th><th>Item</th><th>Num</th></tr><tr><td>1</td><td>boot</td><td>9</td></tr><tr><td>2</td><td>shoe</td><td>Z</td></tr><tr><td>3</td><td>sock</td><td>11</td></tr></table>	Obs	Item	Num	1	boot	9	2	shoe	Z	3	sock	11				
Obs	Item	Num																
1	boot	9																
2	shoe	Z																
3	sock	11																
アンダースコア コア()	<pre>data master; input Item \$ Num Grp \$; datalines; boot 12 wnt shoe 10 snk sock 11 fab ; data trans; input Item \$ Num Grp \$; missing _; datalines; boot 9 wnt shoe 10 _ sock _ fab ; data new; update master trans; by Item; run;</pre>	トランザクションデータセットのアンダースコア()文字として定義されている欠損値は、マスターデータセットの値を数値変数の場合はドット(.)、文字変数の場合は空白で置き換えます。 <table><tr><th>Obs</th><th>Item</th><th>Num</th><th>Grp</th></tr><tr><td>1</td><td>boot</td><td>9</td><td>wnt</td></tr><tr><td>2</td><td>shoe</td><td>10</td><td></td></tr><tr><td>3</td><td>sock</td><td>.</td><td>fab</td></tr></table>	Obs	Item	Num	Grp	1	boot	9	wnt	2	shoe	10		3	sock	.	fab
Obs	Item	Num	Grp															
1	boot	9	wnt															
2	shoe	10																
3	sock	.	fab															

特殊欠損値の使用方法を示す別の例については、“[例 3: 欠損値による更新](#)” (387 ページ)を参照してください。

比較

- UPDATE ステートメントと MERGE ステートメントは、どちらも SAS データセットのオブザベーションを更新します。
- MERGE ステートメントは、最初のデータセットにある既存の値を 2 番目のデータセットにある欠損値に自動的に置き換えます。ただし、UPDATE ステートメントは、デフォルトではこのように動作しません。UPDATE ステートメントを使用してマスターデータセットの既存の値をトランザクションデータセットの欠損値で上書きするには、UPDATERMODE=NOMISSINGCHECK を使用する必要があります。
- UPDATE ステートメントでは、トランザクションを適用することによって、マスターファイル内の選択したオブザベーションの値のみを変更したり、更新することができます。UPDATE ステートメントでは、新しいオブザベーションを追加することもできます。

例

例 1: 基本的な更新

マスターデータセット(work.jan)にトランザクションを適用することにより、これらのプログラムステートメントは新しいデータセット(work.qtr1)を作成します。BY 変数 store は、work.jan と work.week4 の両方に表示される必要があり、マスターデータセット内のその値は一意である必要があります。

```
data work.jan;
input store $ num;
datalines;
amer 4
cali 5
;

data work.week4;
input store $ num;
datalines;
amer 6
cali 10
sizzle 100
;
run;
data work.qtr1;
  update work.jan work.week4;
  by store;
run;
proc print data=work.qtr1; run;
```

Obs	store	num
1	amer	6
2	cali	10
3	sizzle	100

例 2: 変数名を変更する更新

この例は、データセットの変数名を変更したため、プログラムデータベクトル内の同じ変数が上書きされないことを示しています。また、各データセットで WEIGHT 変数の名前が変更され、新しい WEIGHT 変数の値が算出されます。マスターデータセットとトランザクションデータセットは更新を実行するコードの前にリストされています。

Master Data Set

HEALTH

OBS	ID	NAME	TEAM	WEIGHT
1	1114	sally	blue	125
2	1441	sue	green	145
3	1750	joey	red	189
4	1994	mark	yellow	165
5	2304	joe	red	170

Transaction Data Set

FITNESS

OBS	ID	NAME	TEAM	WEIGHT
1	1114	sally	blue	119
2	1994	mark	yellow	174
3	2304	joe	red	170

/*******/

```
data health;
  input ID NAME $ TEAM $ WEIGHT;
  length team $ 6;
  cards;
1114 sally blue 125
1441 sue green 145
1750 joey red 189
1994 mark yellow 165
2304 joe red 170
;
data fitness;
  input ID NAME $ TEAM $ WEIGHT;
  length team $ 6;
  cards;
1114 sally blue 119
1994 mark yellow 174
2304 joe red 170
;
```

```
/* Sort both data sets by ID */
proc sort data=health;
  by id;
run;
proc sort data=fitness;
  by id;
```

```

run;
/* Update Master with Transaction */
data health2;
length STATUS $11;
update health(rename=(weight=ORIG) in=a)
  fitness(drop=name team in=b);
by id ;
if a and b then
do;
  CHANGE=abs(orig - weight);
  if weight<orig then status='loss';
  else if weight>orig then status='gain';
  else status='same';
end;
else status='no weigh in';
run;

proc print data=health2;
  title 'Weekly Weigh-in Report';
run;

```

アウトプット 2.39 変数名を変更する更新

Weekly Weigh-in Report							
Obs	STATUS	ID	NAME	TEAM	ORIG	WEIGHT	CHANGE
1	loss	1114	sally	blue	125	119	6
2	no weigh in	1441	sue	green	145	.	.
3	no weigh in	1750	joey	red	189	.	.
4	gain	1994	mark	yellow	165	174	9
5	same	2304	joe	red	170	170	0

例 3: 欠損値による更新

この例は、マスターデータセット PAYROLL と、通常の欠損値と特殊欠損値を含むデータセット INCREASE を作成する DATA ステップを示しています。更新が行われた後、次の情報に注意してください。

- ID 1026 の給与の値は変更されません。
- ID 1034 の給与の値は特殊欠損値になります。
- ID 1057 の給与の値は通常の欠損値になります。

```

/* Create the Master Data Set */
data payroll;
input ID SALARY;
datalines;
1011 245
1026 269
1028 374
1034 333

```

```

1057 582
;
/* Create the Transaction Data Set */
data increase;
input ID SALARY;
missing A _; /* 1 */
datalines;
1011 376
1026 . /* 2 */
1028 374
1034 A /* 3 */
1057 _ /* 4 */
;
/* Update Master with Transaction */
data newpay;
update payroll increase;
by id;
run;
proc print data=newpay;
title 'Updating with Missing Values';
run;

```

- 1 トランザクションデータセットで **MISSING ステートメント**を指定し、特殊な欠損値 "A" とアンダースコア(_)を定義します。
 - MISSING ステートメントで A から Z のいずれかの文字を使用して特別な欠損値を定義し、更新を実行すると、更新されたデータセット内の欠損値が、定義した特別な欠損値と一致するように変更されます。
 - MISSING ステートメントでアンダースコア文字(_)を使用して特別な欠損値を定義してから更新を実行すると、SAS は更新されたデータセットの欠損値をピリオド(.)に変更します。
- 2 UPDATE ステートメントは、値がトランザクションデータセット内の通常の欠損値である場合、値を更新しません。通常の欠損値は、特別な欠損値として定義されていない値です。これは、欠落している通常の数値変数(.)の通常の表記であるため、UPDATE ステートメントは ID 1026 の元の給与値 269 を保持します。トランザクションデータセットの欠損値を使用して出力データセットを更新する場合は、UPDATE ステートメントで **UPDATEMODE=NOMISSINGCHECK** を指定する必要があります。または、欠損値の代わりにアンダースコア文字を挿入し、MISSING ステートメントを使用してアンダースコア文字を特別な欠損値として定義することもできます。
- 3 ID 1034 の給与は、トランザクションの DATA ステップの MISSING ステートメントで定義されているため、文字 "A" で指定された特別な欠損値です。
- 4 トランザクションデータセットの MISSING ステートメントでアンダースコア(_)を指定し、トランザクションデータセットの値としてアンダースコアを使用してデータを更新すると、SAS はアンダースコアを通常の欠損値として読み取り、出力データセットの値をドットに置き換えます。ドット表記は、SAS で欠損している数字を表す伝統的な記号です。

アウトプット 2.40 欠損値による更新

Updating with Missing Values

Obs	ID	SALARY
1	1011	376
2	1026	269
3	1028	374
4	1034	A
5	1057	.

注: 更新されたデータセットの行 5 の SALARY の値は、元の給与を欠損値に置き換えます。MISSING ステートメントで欠損値を定義するためにアンダースコアが使用されたため、SAS はアンダースコアを通常の欠損値記号であるピリオドに置き換えます。詳細については、[表 2.13 \(384 ページ\)](#)を参照してください。

関連項目

- [“データセットオプションの定義” \(SAS データセットオプション: リファレンス\)](#)
- [“Reading, Combining, and Modifying SAS Data Sets” \(SAS Language Reference: Concepts\)](#)

ステートメント:

- [“BY ステートメント” \(57 ページ\)](#)
- [“MERGE ステートメント” \(252 ページ\)](#)
- [“MISSING ステートメント” \(261 ページ\)](#)
- [“MODIFY ステートメント” \(261 ページ\)](#)
- [“SET ステートメント” \(356 ページ\)](#)

システムオプション:

- [“MISSING= システムオプション” \(SAS システムオプション: リファレンス\)](#)

WHERE ステートメント

SAS データセットから特定の条件を満たすオブザベーションを選択します。

該当要素: DATA ステップまたは PROC ステップ

カテゴリ:	アクション CAS In-Database 処理
種類:	宣言
注:	WHERE ステートメントで乱数関数を使用すると、サブセット化 IF ステートメントで乱数関数を使用する場合とは異なる結果セットが生成される場合があります。SAS により基準がどのように内部で最適化され、予想される動作であるかがこの違いが発生する可能性があります。
ヒント:	SAS チュートリアルビデオ DATA ステップの SAS テーブルのフィルタリング は、WHERE ステートメントを使用してデータをフィルタリングする方法を示します。

構文

WHERE *where-expression-1*
<*logical-operator where-expression-n*>;

引数

where-expression

オペランドや演算子で構成される算術式または論理式を指定します。

ヒント 次のいくつかのセクションで説明されるオペランドや演算子は WHERE= データセットオプションでも使用できます。

where-expression を複数指定することができます。

logical-operator

AND、AND NOT、OR、OR NOT を指定できます。

詳細

基本情報

WHERE ステートメントを使用すると、入力データセットからすべてのオブザベーションを読み込む必要がなくなるため、SAS プログラムの処理効率が向上します。

WHERE ステートメントは条件を指定して実行することはできません。そのため、IF-THEN ステートメントの中で使用することはできません。

WHERE ステートメントには、論理演算子で連結した複数の WHERE 式を指定できません。

注: WHERE 式を使用して SAS データセットにあるオブザベーションのサブセットにアクセスするときに、インデックス付きの SAS データセットを使用するとパフォーマンスが大幅に向上します。インデックス付きデータセットを使用した WHERE 式の処理についての説明は、[インデックスと WHERE 式](#)を参照してください。

DATA ステップでの使用

WHERE ステートメントは、直前にある SET、MERGE、MODIFY、UPDATE の各ステートメントに指定されているすべてのデータセットに適用されます。WHERE ステートメントで使用する変数は、WHERE ステートメントが適用されるすべてのデータセットに含まれている必要があります。WHERE ステートメントは、POINT=オプションを指定した SET ステートメントや MODIFY ステートメントとともに使用することはできません。

OBS=および FIRSTOBS=の処理を WHERE ステートメントの処理に適用できます。詳細については、[“WHERE 式を使用したデータセットおよびシステムオプション” \(SAS プログラマガイド: エッセンシャル\)](#)を参照してください。

WHERE ステートメントには、1 つ以上の入力 SAS データセットが必要です。生データを読み取る DATA ステップで WHERE ステートメントを指定する場合、DATA ステップでも SET、MERGE、MODIFY、または UPDATE ステートメントのいずれかを指定する必要があります。

生データとは、SAS が外部ファイルから読み取るデータ、または SAS プログラムに存在するインストリームデータであり、SAS が DATALINES ステートメントを使用して読み取るデータです。

DATA ステップを繰り返すたびに、SET、MERGE、MODIFY、UPDATE の各ステートメントの最初の操作として、入力データセット内のオブザベーションが WHERE ステートメントの条件を満たしているかどうかの確認が実行されます。入力データセットオプションの適用直後、DATA ステップ内の他のステートメントを実行する前に、WHERE ステートメントが適用されます。DATA ステップでのオブザベーションの結合に MERGE、MODIFY、UPDATE ステートメントとともに WHERE ステートメントを使用する場合、各入力データセットからオブザベーションを選択した後で結合します。

DATA ステップ内の WHERE ステートメントと BY ステートメント

DATA ステップに WHERE ステートメントと BY ステートメントの両方が含まれている場合、BY グループを作成する前に WHERE ステートメントが実行されます。そのため、BY グループは、WHERE ステートメントで選択したオブザベーションサブセットのオブザベーショングループを反映します。元の入力データセットのオブザベーションの現行の BY グループを反映しません。

BY グループの処理の詳細については、[“BY-Group Processing in the DATA Step” \(SAS Language Reference: Concepts\)](#)を参照してください。

PROC ステップでの使用

WHERE ステートメントは、SAS データセットを読み込む SAS プロシジャとともに使用できます。WHERE ステートメントは、プロシジャで処理する元のデータセットをサブセット化する場合に便利です。*Base SAS プロシジャガイド*では、複数のデータセットの指定可能なプロシジャでの WHERE ステートメントのアクションのみが説明されています。他の場合、WHERE ステートメントはこのトピックで説明されているように動作します。

インデックスの使用

次の演算子と関数のいずれかと組み合わせてインデックス付き変数が使用されている場合、DATA ステップまたは PROC ステップは利用可能なインデックスを使用してデータの選択を最適化しようとします。

- BETWEEN-AND 演算子
- 比較演算子(コロン修飾子あり/なし)
- CONTAINS 演算子
- IS NULL および IS NOT NULL 演算子
- LIKE 演算子
- TRIM 関数
- SUBSTR 関数(場合による)

SUBSTR では次の引数が必要です。

```
where substr(variable,position,length)
      ='character-string';
```

SUBSTR 関数の引数が次のすべての条件を満たす場合、インデックスが処理に使用されます。

- *position* が 1 に等しい
- *length* が *variable* の長さよりも短い、またはその長さに等しい
- *length* が *character-string* の長さに等しい

WHERE 式で使用されるオペランド

WHERE 式で使用されるオペランドには次の値が含まれます。

- 定数
- 日付と時刻の値
- SAS データセットから取得した変数の値
- WHERE 式自身で作成された値

DATA ステップで作成される変数は WHERE 式で使用できません(たとえば、FIRST.*variable*、LAST.*variable*、_N_、または割り当てステートメントで作成される変数)。これは、DATA ステップまたは PROC ステップにオブザベーションを読み込む前に WHERE ステートメントが実行されるためです。WHERE 式に比較演算子が使用されている場合、フォーマットされていない変数の値が比較されます。

WHERE 式でオペランドを使用する例をいくつか次に示します。

- `where score>50;`
- `where date>='01jan1999'd and time>='9:00't;`
- `where state='Mississippi';`

他の SAS 式の場合と同様に、数値変数の名前は単独で指定できます。この場合、値が 0 または欠損値の場合は偽になります。その他の値の場合は真になります。これらの例では、WHERE 式に数値変数 EMPNUM と SSN が含まれています。

- `where empnum;`

■ where empnum and ssn;

WHERE 式では、文字リテラルまたは文字変数の名前も単独で使用できます。
WHERE 式として文字変数を単独で指定した場合、文字変数の値が空ではないオブザベーションが選択されます。

WHERE 式で使用される演算子

WHERE ステートメントには SAS 演算子と WHERE 式の特殊な演算子を使用できます。
演算子の詳細なリストについては、[“WHERE 式の演算子” \(SAS プログラマガイド: エッセンシャル\)](#)を参照してください。

このテーブルは、WHERE ステートメントの演算子を示します。

表 2.7 WHERE ステートメント演算子

演算子の種類	記号またはニーモニック	説明
算術演算子		
	*	乗算
	/	除算
	+	加算
	-	減算
	**	累乗
比較演算子 ⁴		
	= または EQ	等しい
	^=、^!=、~=、または NE ¹	等しくない
	>または GT	より大きい
	<または LT	未満
	>=または GE	以上
	<=または LE	以下
	IN	リストのどれかと等しい
論理演算子(ブール演算子)		

演算子の種類	記号またはニーモニック	説明
	& または AND	論理積
	または OR ²	論理和
	~, ^, ¬, または NOT ¹	論理否定
その他の演算子		
	³	文字変数の連結
	()	評価の順序を示す
	+ 接頭辞	正の数
	- 接頭辞	負の数
WHERE 式のみ		
	BETWEEN-AND	対象範囲
	? または CONTAINS	文字列
	IS NULL または IS MISSING	欠損値
	LIKE	パターンマッチ
	=*	類似
	SAME-AND	元の句を再入力せずに、既存の WHERE ステートメントに句を追加

¹ キャレット(^)、チルダ(~)、否定(¬)はすべて論理否定を示しています。お使いのキーボード上の文字を使用するか、または対応するニーモニックを使用します。

² OR 記号(|)、ブロークンバーティカルバー(|)、感嘆符(!)はすべて論理和を表します。お使いのキーボード上の文字を使用するか、または対応するニーモニックを使用します。

³ 連続する 2 つの OR 記号(||)、ブロークンバーティカルバー(| |)、感嘆符(!!)はすべて連結を表します。お使いのキーボード上の文字を使用します。

⁴ 比較演算子とともにコロンの修飾子(:)を使用すると、文字列にある指定した接頭辞のみを比較できます。

エスケープ文字は、一連の文字の中で、その後続く文字が別の意味を持つことを示す 1 つの文字です。LIKE 演算子の場合、エスケープ文字は、特殊文字関数を実行する代わりに、変数値内の%および_文字のリテラルインスタンスを検索することを意味します。

たとえば、変数 X に値 abc、a_b、および axb が含まれている場合、エスケープ文字を含む次の LIKE 演算子は、値 a_b のみを選択します。エスケープ文字(/)は、パタ

ーンが文字 a と b で囲まれたリテラル '_'を検索することを指定します。エスケープ文字(/)は検索の一部ではありません。

```
where x like 'a/_b' escape '/';
```

エスケープ文字を使用しないと、次の LIKE 演算子は値 a_b と axb を選択します。検索パターン内の特殊文字アンダースコアは、アンダースコア付きの値を含む任意の b 文字 1 文字と一致します。

```
where x like 'a_b';
```

エスケープ文字を指定するには、パターンマッチング式にその文字を含めてから、キーワード ESCAPE の後にエスケープ文字式を続けます。エスケープ文字を含める場合は、パターンマッチング式を引用符で囲む必要があります、列名を含めることはできません。エスケープ文字式は 1 文字として評価されます。オペランドは文字または文字列リテラルである必要があります。それが 1 文字の場合は、引用符で囲みます。

```
LIKE 'pattern-matching-expression' ESCAPE 'escape-character-expression'
```

比較

- SAS/FSP では、編集および参照するために、WHERE コマンドを使用してデータをサブセット化できます。WHERE ステートメントと WHERE=データセットオプションのどちらも、ウィンドウ環境プロシジャでは使用できます。また、WHERE コマンドと組み合わせることもできます。
- SET、MERGE、MODIFY、UPDATE の各ステートメントで複数のデータセットを指定するときに個々のデータセットからオブザベーションを選択するには、WHERE=データセットオプションを各データセットに適用する必要があります。DATA ステップ内で WHERE ステートメントと WHERE=データセットオプションが同じデータセットに適用される場合、WHERE=データセットオプションが優先され、WHERE ステートメントは無視されます。WHERE データセットオプションが適用されていない他のデータセットでは、WHERE ステートメントが使用されます。
- DATA ステップにおける WHERE ステートメントとサブセット化 IF ステートメントの重要な違いを次に示します。
 - WHERE ステートメントでは、オブザベーションがプログラムデータベクトルに読み込まれる *前* にオブザベーションを選択します。そのため、プログラムの処理効率が向上します。サブセット化 IF ステートメントでは、オブザベーションがプログラムデータベクトルに読み込まれた後に、オブザベーションを操作します。
 - BY ステートメントが SET、MERGE、UPDATE の各ステートメントとともに使用される場合、WHERE ステートメントとサブセット化 IF ステートメントは異なるデータセットを作成します。データセットに違いが発生するのは、サブセット化 IF ステートメントではオブザベーションを選択する前に BY グループが作成され、WHERE ステートメントではオブザベーションを選択した後に BY グループが作成されるためです。
 - WHERE ステートメントは IF ステートメントの一部として条件を指定して実行できませんが、サブセット化 IF ステートメントは条件を指定して実行できます。

- WHERE ステートメントでは、SAS データセットにあるオブザベーションのみを選択します。ただし、サブセット化 IF ステートメントでは、既存のデータセットや INPUT ステートメントで作成されたオブザベーションからもオブザベーションを選択します。
- SAS ウィンドウ環境プロシジャでは、サブセット化 IF ステートメントを使用してオブザベーションをサブセット化して、参照および参照できません。
- WHERE ステートメントを DROP または KEEP ステートメントと混同しないでください。DROP と KEEP ステートメントでは、処理対象の変数を選択します。WHERE ステートメントでは、オブザベーションを選択します。

例

例 1: SAS DATA ステップで WHERE ステートメントを指定する

この DATA ステップは、name の値が **Mac** で始まり、city の値が **Charleston** または **Atlanta** であるデータセット customer のオブザベーションのみを含む SAS データセットを作成します。

```
data testmacs;
  set customer;
  where substr(name,1,3)='Mac' and
    (city='Charleston' or city='Atlanta');
run;
```

例 2: CAS DATA ステップで WHERE ステートメントを指定する

この例では、Sashelp.Class データセットから、Age の値が **14** より大きいオブザベーションのみを含む CAS テーブルを作成します。

```
cas casauto sessopts=(caslib='casuser'); /* 1 */
caslib_all_assign;
libname mycas cas;

data mycas.class; /* 2 */
  set sashelp.class;
run;

data mycas.class_out;
  set mycas.class;
  where Age>14; /* 3 */
run;
```

- 1 CAS に接続します。CAS で DATA ステップを実行するには、まず CAS サーバーに接続し、CAS セッションを開始する必要があります。詳細については、“[例のコードの設定](#)” (*SAS Cloud Analytic Services: DATA ステッププログラミング*)を参照してください。
- 2 Sashelp ライブラリから CAS にサンプルデータをいくつか読み込みます。出力テーブルで CAS エンジンの libref を指定します。¹
- 3 CAS で DATA ステップを実行します。読み込んだ CAS テーブル mycas.class をフィルタリングするために WHERE ステートメントを指定します。DATA ステップ

では、mycas.class からのフィルタリングされた出力を、出力テーブル mycas.class_out に書き込みます。

DATA ステップと SAS Cloud Analytic Services の詳細については、“[CAS での DATA ステップ処理](#)” (*SAS Cloud Analytic Services: DATA ステッププログラミング*)を参照してください。

例 3: WHERE ステートメントのみで使用可能な演算子を使用する

- BETWEEN-AND 演算子の例:

```
where empnum between 500 and 1000;
```

- CONTAINS 演算子の例:

```
where company ? 'bay';
where company contains 'bay';
```

- IS NULL および IS MISSING 演算子の例:

```
where name is null;
where name is missing;
```

- 文字 D で始まるすべての名前を選択する LIKE 演算子の例:

```
where name like 'D%';
```

- 次の名前リストからパターンマッチを行う LIKE 演算子の例:

```
Diana
Diane
Dianna
Dianthus
Dyan
```

WHERE ステートメント	選択される名前
where name like 'D_an';	Dyan
where name like 'D_an_';	Diana、Diane
where name like 'D_an__';	Dianna
where name like 'D_an%';	リストに含まれるすべての名前

- SOUNDS-LIKE 演算子を使用すると、“Smith”に似た名前が選択されます。

```
where lastname=*'Smith';
```

- SAME-AND 演算子の例:

```
where year>1991;
```

1. SAS データセットを CAS に読み込むために DATA ステップを使用した場合、DATA ステップは CAS サーバー内で実際には実行されません。DATA ステップが CAS で実行されるためには、入力テーブルと出力テーブルの両方がインメモリ CAS テーブルとして存在し、入力テーブルと出力テーブルの両方に CAS エンジンの libref が含まれている必要があります。

```
...more SAS statements...  
where same and year<1999;
```

この例にある 2 番目の WHERE ステートメントは、次の WHERE ステートメントに相当します。

```
where year>1991 and year<1999;
```

関連項目

- [Base SAS プロシジャガイド](#)
- [“BY グループ処理の定義” \(SAS プログラマガイド: エッセンシャル\)](#)
- [SAS SQL Query Window ユーザーガイド](#)
- [SAS/IML ユーザーガイド](#)
- [“Understanding SAS Indexes” \(SAS Language Reference: Concepts\)](#)
- [“WHERE 式” \(SAS プログラマガイド: エッセンシャル\)](#)

データセットオプション:

- [“WHERE= データセットオプション” \(SAS データセットオプション: リファレンス\)](#)

ステートメント:

- [“IF ステートメント: サブセット化” \(137 ページ\)](#)

X ステートメント

SAS セッションから動作環境のコマンドを発行します。

注: [X ステートメント](#)は SAS グローバルステートメントに移動しました。

3

SAS ステートメント環境変数のディクショナリ

ディクショナリ	399
SAS_FTP_AUTHTLS 環境変数	399

ディクショナリ

SAS_FTP_AUTHTLS 環境変数

TLS (Transport Layer Security) 認証を有効化します。

注: [SAS_FTP_AUTHTLS 環境変数](#)は SAS グローバルステートメントに移動しました。

