Data Mine Tech Ltd.

Data Bring New Insight to Your Business

2010/11/1(2012/11/8 一部改訂)

SAS 講習会用テキスト(Base SAS 編)

目標

SAS 言語を用いたデータ入力・加工・集計・レポート作成を行うプログラミングの基礎を習得します。

内容

1. SAS 概要 2

1.1 SAS の基本的概念、文法、制限、データライブラリと使い方 ~ DATA ステップと PROC ステップ、命名規則、LIBNAME ステートメント、関数、演算子、欠損値、エラー対応 2

1.2 データ読取り・変数作成と横方向の集計処理 ~ FILENAME、LIBNAME、INPUT、SET、IMPORT、EXPORT、割り当てステートメント、集計関数、データセットオプション、PRINT、SORT プロシジャ 16

2. データ加工 36

2.1 ファイルの連結・マージ・更新を含むプログラミングステートメント ~ SET,MERGE,UPDATE ステートメント,DO ループ,配列処理 36

2.2 日付処理・関数・フォーマットの利用と DATA ステップを用いたレポーティング ~ 日付フォーマット、日付インフォーマット、日付関数、FORMAT プロシジャ、BY グループ処理、PUT ステートメント 55

3. レポーティング 69

3.1 集計、データセットの転値、SQL プロシジャ ~ FREQ、SUMMARY、TRANSPOSE、SQL プロシジャ 69

3.2 レポーティングとマクロ処理 ~ TABULATE プロシジャとマクロ処理 89

[別表 1] DATA ステップで使えるステートメント一覧 106

[別表 2] PROC ステップの種類 107

[別表 3] グローバルステートメント一覧 107

[別表 4] 演算子一覧 108

[別表 5] 関数一覧 108

[別表 6] フォーマット一覧 112

[別表 7] インフォーマット一覧 119

[別表 8] merge ステートメントによる 2 つのデータセットのマージ例 122

1. SAS 概要

SAS 言語[1]は PL/1,C,Fortran,BASIC などと同じようにプログラミング言語の一種です。 ただし、元々はコンピュータ専門家ではない統計解析専門家のために開発されたデータ入力・加工・解析用の言語です。そのため、PL/1,C などの手続き型言語と比較すると、はるかに使いやすい仕様となっています。SAS 言語は主にデータ加工を柔軟に行う目的を実現するために豊富な関数やプログラミング機能を有する DATA ステップ[2]と、特定のデータ分析・集計・レポートを行うために用意されているパラメータ指定仕様の PROC ステップ[3]の 2 つの性格の異なるコンポーネントを有しており、これらを組合せてプログラミングを行うようになっています。

1.1 SAS の基本的概念、文法、制限、データライブラリと使い方
~ DATA ステップと PROC ステップ、命名規則、LIBNAME ステートメント、関数、演算子、欠損値、エラー対応

SAS の操作方法、基本的概念および SAS 言語の要素などについて学びます。

[SAS の起動と終了]

まず、SAS を起動します。デスクトップに SAS 起動アイコンがある場合は、それをダブルクリックします。無ければ、Window から SAS 起動プログラム選択します。

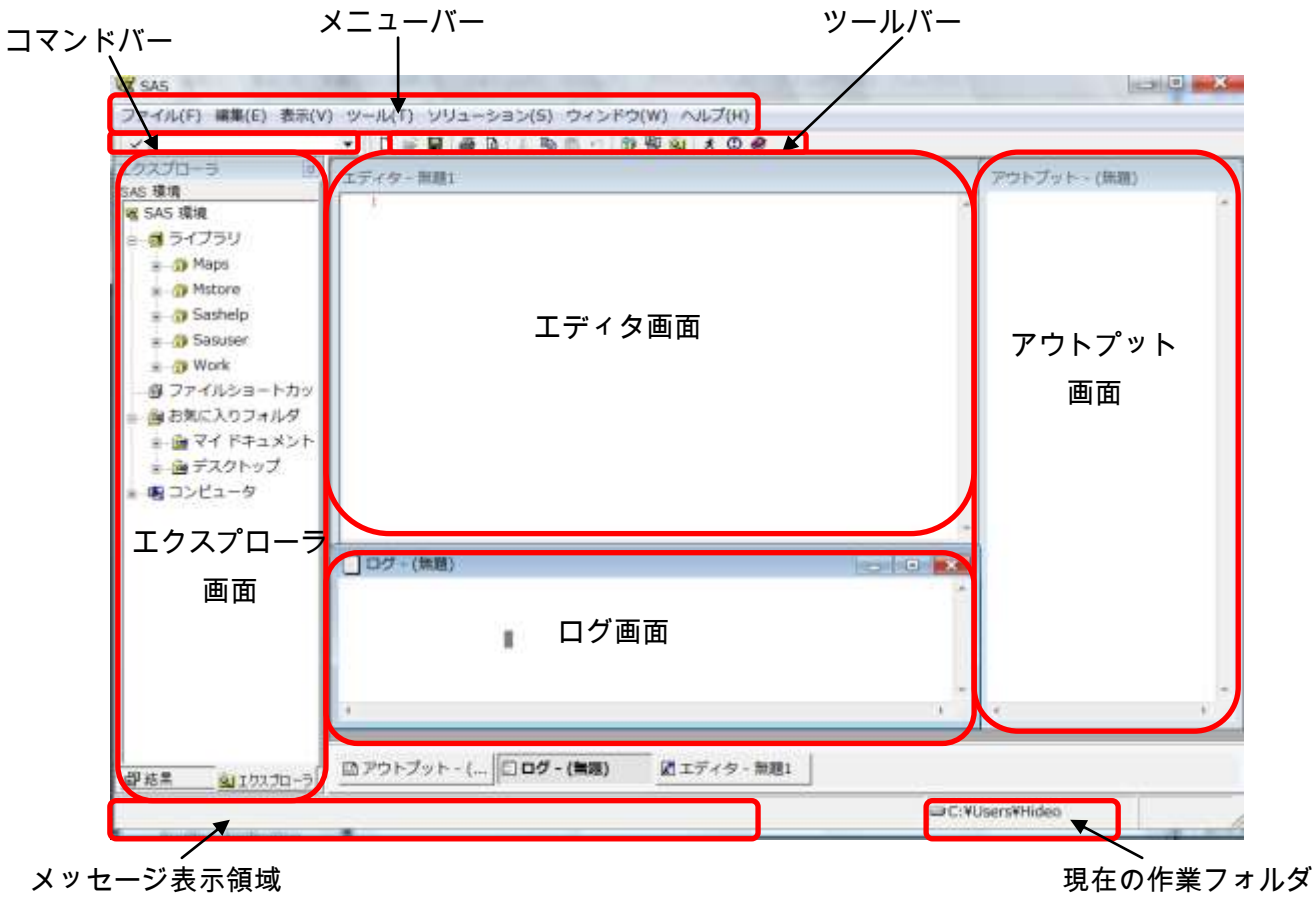
SAS の最新バージョンは 9.2 ですがここでは 9.1 を用いています。このテキストの範囲ではいずれのバージョンでもほとんど同じです。

起動すると、以下のようにいくつかの画面(window)が表示されます。

1 SAS は 1973 年ごろから開発がはじまり、当時は第四世代言語と呼ばれていました。

2 使いやすいだけでなく、扱えるデータ量やファイル形式などに制約が無く、また、複雑な処理も高級言語なみに実行できる機能を持っています。なお DATA ステップはコンパイル言語です。

3 PROC ステップの個々の構成要素をプロシジャと呼んでいます。これは DLL(動的結合ライブラリ)ファイルとなっています。



SAS にはたくさんの画面(WINDOW)がありますが、ユーザが良く使う主な画面は上記の図の中央の上部に表示されている**「エディタ」画面(「プログラム編集」画面)**、中央下部に表示されている**「ログ」画面**、そして右側に表示されている**「アウトプット」画面(「リスト出力」画面)**の３画面です。
ユーザは**エディタ画面**の中に処理したい内容を SAS 言語で書いたプログラムとしてコーディングを行い、**サブミットしたい範囲を選択状態にしてから**ツールバーの**サブミットアイコン(人が走っている形のアイコン)**をクリックします。次に、プログラムにエラーがなかったかどうか、読込みや書き込みを行ったデータセットの情報とか実行時間などの情報を**ログ画面で確認し**、統計計算などの処理結果を表示するプログラムの場合は、**アウトプット画面**または他の出力画面(例えば **HTML 出力画面**や**グラフ表示画面**)に思った通りの実行結果が出力されていいるかどうかを確認します。途中で、作成中または完成したプログラムをファイルに保存しておきます。このような操作を繰り返してデータ処理プログラムを完成させていきます。

左側の画面は右側の表示エリアと区別された領域となっており、このときはドッキングした状態になっています。下の方にあるタブをクリックすることにより**「エクスプローラ」(SAS エクスプローラ)**と**「結果」画面**を切り替えて表示できる画面です。
エクスプローラ画面は SAS ファイルを検索する**「ライブラリ」**を良く使うことになります。結果画面はリスト出力結果やグラフ出力結果を個々のアイテムとしてアイコン表示してあり、実行済みで保持されている結果アイテムをすばやく検索するときに使います。

「メニューバー」には**「ファイル」**、**「編集」**、**「表示」**、**「ツール」**、**「ソリューション」**、**「ウィンドウ」**、**「ヘルプ」**の７つのメニューが並んでいます。これらはプルダウンメニューになっており、この中に含まれているコマンド群は SAS 以外の他のウィンドウアプリケーションと大体同じくくりになっています。

メニューバーから良く使うコマンドは以下のとおりです。

・**「エディタ」をアクティブ**にした上で**「ファイル」**メニューから**「プログラムの新規作成」**、**「プログラムを開く」**、**「上書き保存」**、**「名前を付けて保存」**のサブメニューを選択します.
・**対象画面をアクティブ**にしておいて**「編集」**メニューから**「すべて選択」**した後**「コピー」**、また、**「元に戻す」**、**「やり直し」**の操作、そして**「検索」**、**「置換」**などの操作を行います.
・**「表示」**メニューから**「拡張エディタ」**を選択して新しいエディタ画面を開く. また閉じてしまった**「ログ」**、**「アウトプット」**、**「結果」**、**「エクスプローラ」**などの画面を再度表示します.
・**「ツール」**メニューから「**テキストエディタ」**を開く（ただしこれは上記「表示」メニューから「拡張エディタ」を選択して新しいエディタ画面を開くと同じです）
・**「ツール」**メニューから**「ユーザ設定」**や**「オプション」**を選択し設定を確認したり変更します..

メニューバーの下にはユーザが直接コマンドを入力する**「コマンドバー」**と**「ツールバー」**があります。コマンドバーはメニューバーにあるコマンドをメニュー選択ではなく直接コマンドとして入力できる機能です。例えば log とタイプして左のチェックマークを押すかエンターキーを押すと、ログ画面がアクティブになります。ツールバーにはメニューバーにあるコマンドの中で**「サブミット」**や**「コピー」**、**「ペースト」**など良く使うコマンドをアイコンとして格納しています。

画面の下にはコマンドを実行したときに SAS から出る**メッセージを表示する領域**と、**現行の作業フォルダのディレクトリパス**を表示するエリアがあります。現行の作業フォルダは重要になる場合があります。例えば作成中の SAS プログラムを保存するのに、コマンドバーから　file "temp.sas" とファイルのパスを指定せずに名前だけをタイプしてエンターを押したり、SAS プログラムの中で file "temp.txt"; などとファイルをフルパス指定せずにサブミットした場合、SAS はこの現行の作業フォルダをカレントディレクトリとみなし、このフォルダの中にファイルを作成します。というわけでどこに作成されたかを知るために重要というわけです。なお、ここでいう作業フォルダは後述する一時 SAS データセットを格納する WORK ライブラリの所在とは全く異なる点に注意してください。(WORK ライブラリの所在は libname コマンドでわかります)

SAS セッションを終了するには**「ファイル」**メニューから**「終了」**を選択します。「SAS」セッションを終了しますか？」と聞いてきますので「OK」を選択すると SAS は終了します。右上の X のところをクリックしても同じです。

では、SAS を**終了しない**でそのままで続けます。

[DATA ステップと PROC ステップ]

SAS 言語によるプログラミングは、DATA ステップと呼ばれる実行単位と PROC ステップと呼ばれる実行単位を組合せて行います。 DATA ステップはデータ検索加工機能を実行するためのプログラミング言語部分であり、PROC ステップは基本統計やレポーティングその他の特定機能を実行するために用意されている組み込みモジュールです。 DATA ステップは DATA ステートメントで始まり、PROC ステップは PROC ステートメントで始まり、どちらも RUN ステートメントで明示的に終了します。 ただし、RUN ステートメントを記述しなくても、次の DATA ステップまたは PROC ステップを開始すればその前の DATA ステップまたは PROC ステップは暗黙的に終了します。 なお、DATA ステップと PROC ステップに属しない汎用ステートメントもあります。

では SAS の使い方を実習するために、プログラミングを体験してみましょう。以下のプログラムをエディタに入力してください。入力が終わったらサブミットしてください。なお、/* */ で囲んだテキスト部分

はコメントですので、入力しなくても結構です。

(プログラム 1.1-1)

```
options nocenter;          /*オプション文*/ /*汎用ステートメント*/
data hello;                /*データ文*/ /*DATAステップの開始*/
  message="Hello, World"; /*割り当て文*/ /*DATAステップで使えるステートメント*/
run;                       /*ラン文*/ /*DATAステップの終了*/
proc print;                /*プロックプリント文*/ /*PROCステップの開始*/
run;
```

(ログ)

```
1    options nocenter;          /*オプション文*/ /*汎用ステートメント*/
2    data hello;                /*データ文*/ /*DATA ステップの開始*/
3      message="Hello, World"; /*割り当て文*/ /*DATA ステップで使えるステートメント*/
4    run;

NOTE: データセット WORK.HELLO は 1 オブザベーション、 1 変数です。
NOTE: DATA ステートメント 処理 (合計処理時間):
       処理時間                0.22 秒
       CPU 時間                0.03 秒


4  !                            /*ラン文*/ /*DATA ステップの終了*/
5    proc print;                /*プロックプリント文*/ /*PROC ステップの開始*/
6    run;

NOTE: データセット WORK.HELLO から 1 オブザベーションを読み込みました。
NOTE: PROCEDURE PRINT 処理 (合計処理時間):
       処理時間                0.82 秒
       CPU 時間                0.32 秒
```

(アウトプット)

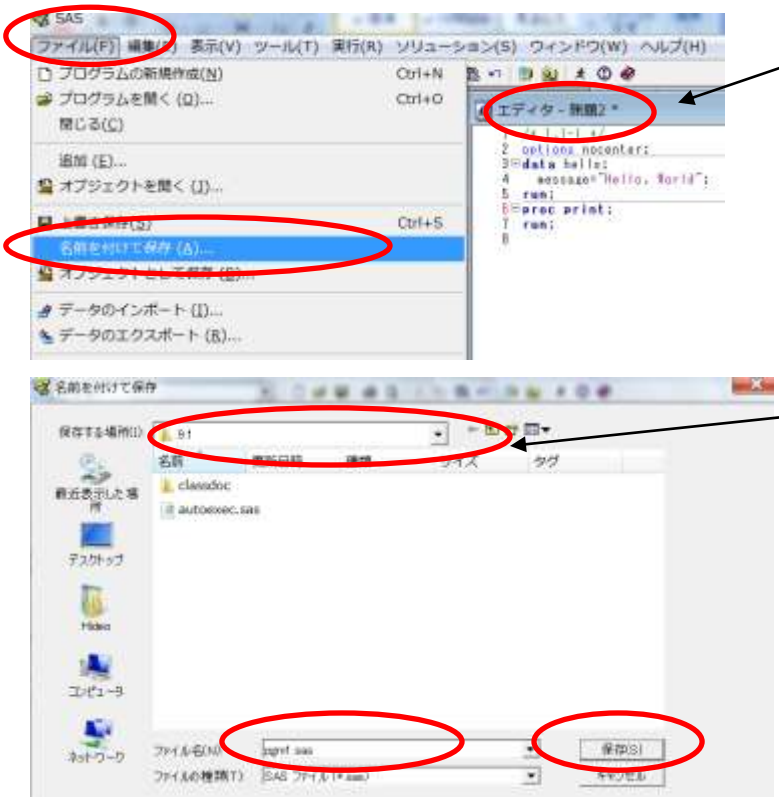
```
OBS      message

 1     Hello, World
```

ログにエラーが出た場合や出力結果が出ない場合は、プログラムにタイプミスがありますので、訂正して再度サブミットしてください。ここではとにかく上手く結果が出るまでやってください。

[プログラムの保存]

上手く動いたプログラムはファイルに保存しておきましょう。プログラム編集画面をアクティブにしておいてから、メニューバーの「ファイル」から「名前を付けて保存」を選択して、プログラム名をつけて保存します。以降、ときどき名前を変えて保存しておくと良いと思います。保存先の標準フォルダ名は c:¥USERS¥ユーザ名¥DOCUMNETS¥MY SAS Files¥9.1[4]です。

4 ここは標準の SASUSER ディレクトリにもなっています。



保存したプログラムをエディタによびだすには**ファイル**メニューの**プログラムを開く**から保存したファイルを選択します。または、Window の Explore から呼び出したい SAS プログラムファイルを**エディタ画面に Drag&Drop** しても同じ結果になります。開いたプログラムファイルの名前の付いたエディタが新たにオープンします。

[SAS データセット]

SAS で管理するデータセットを SAS データセットと呼びます。SAS データセットは EXCEL と同じような表形式のデータ集合の形をしており、行（レコード）方向を**オブザベーション**、列（カラム）方向を**変数**と呼びます。[5] EXCEL には行や列の数に制限がありますが、SAS には制限がありません。[6] SAS データセットの特徴は、データ部と呼ばれるデータ部分に加えて、ディスクリプタ部と呼ばれるデータセットの記述部分が備わっていることです。ディスクリプタ部にはデータセット全体に関する情報（インデクス情報など）や各変数情報（型・長さ・ラベル・フォーマットなど）が格納されています。

これ以降、プログラムを入力してサブミットするときは、必ず実行したい部分を選択（反転状態）してからサブミットしてください。

5 オブザベーション、変数という呼び名は SAS 言語が元々統計解析用のアプリケーション開発言語として開発された歴史に由来しています。

6 OS の制限（SAS8.2 以前では変数の数の上限は 32767 でしたが、SAS9.1 からそれ以上、オブザベーションは容量の制約）内になります。

(プログラム 1.1-2)

```
data sample; /*データセットの作成*/
  input ID name $ sex $ age height weight; /*変数の名前と型を決めて読み込みを実行する文*/
cards;       /*データはこれ以降に入力するという意味の文*/
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;            /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(ログ)

```
7    data sample; /*データセットの作成*/
8      input ID name $ sex $ age height weight; /*変数の名前と型を決めて読み込みを実行する文*/
9    cards;

NOTE: データセット WORK.SAMPLE は 4 オブザベーション、 6 変数です。
NOTE: DATA ステートメント 処理 (合計処理時間):
       処理時間              0.00 秒
       CPU 時間              0.00 秒


9  !              /*データはこれ以降に入力するという意味の文*/
14   ;            /*データを CARDS 文で入力する場合のデータ入力終了の合図*/
15   proc print data=sample;  /*データ部を表示するプロシジャ*/
16   run;

NOTE: データセット WORK.SAMPLE から 4 オブザベーションを読み込みました。
NOTE: PROCEDURE PRINT 処理 (合計処理時間):
       処理時間              0.03 秒
       CPU 時間              0.01 秒
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	77

[DATA ステップのループ実行の仕組み]

DATA ステップは入力するオブザベーションが無ければ 1 回だけ実行されます。しかし外部データからの入力(INPUT ステートメント)や SAS データセットからの入力(SET ステートメントなど)があるときは、読み取るデータ行(SAS データセットの読み取りの場合はオブザベーション)が尽きるまで、自動的に繰り返し実行されます。これを **DATA ステップのループ実行**と呼びます。そして SAS データセットに書きこみを行うステートメントである **OUTPUT ステートメント**がその DATA ステップの中に存在しなければ、DATA ステップの終わり(RUN ステートメントもしくは CARDS ステートメント)の**直前に OUTPUT ステートメントを自動的に挿入**し、読み込むデータ行またはオブザベーションが尽きるまで DATA ステップで作成する各変数値の値(これを**プログラムデータベクトル**と呼んでいます。)を持つ 1 オブザベーションを作成する SAS データセットへの書き込みを繰り返し実行します。この例では実行ステートメントは INPUT ステートメントのみとなっており、ここで読み込まれた 6 個の変数値を持つオブザベーションをデータ行の数である 4 回 DATA ステップがループ実行され、都合 4 オブザベーションを持つ SAS データセット SAMPLE

が作成されます。

[SAS データセットのディスクリプタ部の表示]

(プログラム 1.1-3)

```
proc contents data=sample; /*ディスクリプタ部を表示するプロシジャ*/
run;
```

(ログ)

```
NOTE: PROCEDURE CONTENTS 処理 (合計処理時間):
      処理時間              0.06 秒
      CPU 時間              0.01 秒
```

(アウトプット)

```
                                                    2010 年 09 月 27 日 月曜日 午前 11 時 02 分 57 秒    9

CONTENTS プロシジャ

データセット名        WORK.SAMPLE                              オブザベーション数              4
メンバータイプ        DATA                                     変数の数                        6
エンジン              V9                                       インデックス数                  0
作成日時              2010 年 09 月 27 日 月曜日 午後 02 時 42 分 20 秒    オブザベーションのバッファ長    48
更新日時              2010 年 09 月 27 日 月曜日 午後 02 時 42 分 20 秒    削除済みオブザベーション数      0
保護                                                           圧縮済み                        NO
データセットタイプ                                             ソート済み                      NO
ラベル
データ表現            WINDOWS_32
エンコード            shift-jis  Japanese (SJIS)


                                  エンジン/ホスト関連情報

データセットのページサイズ  4096
データセットのページ数      1
データページの先頭          1
ページごとの最大 OBS 数     84
先頭ページの OBS 数         4
データセットの修復数        0
ファイル名                  C:¥Users¥Hideo¥AppData¥Local¥Temp¥SAS Temporary
                            Files¥_TD7592¥sample.sas7bdat
作成したリリース            9.0101M3
作成したホスト              WIN_PRO


    変数と属性の昇順リスト

#    変数      タイプ    長さ

1    ID        数値         8
4    age       数値         8
5    height    数値         8
2    name      文字         8
3    sex       文字         8
6    weight    数値         8
```

[変数の型]

SAS の変数のタイプ（型）は以下のとおり、**数値タイプ**と**文字タイプ**の 2 通りしかありません。

変数のタイプ

・数値タイプ

内部的に 3 バイト～８バイト[7]の浮動小数点形式で格納される。
・文字タイプ
1～32767 バイトの長さまでの文字列を値として持つことができる。

数値変数は固定小数点形式で持つことができませんので、丸め誤差にシビアなアプリケーションに SAS を使うような場合は注意が必要です。
文字変数に定数を与えるには、ダブルクオテーション（"）で囲むか、シングルクオテーション（'）で囲んで指定します。
例
name=”Robert Edison”
sex='M'

[SAS カタログ]

SAS で管理するファイルには SAS データセットの他にもグラフィック情報、フォーマット定義情報を保存したものなどがあります。これらはそれぞれ SAS グラフィックカタログ、SAS フォーマットカタログなどと呼び、合わせて SAS カタログと呼びます。SAS データセットと SAS カタログを合わせて SAS ファイルと呼びます。

(プログラム 1.1-4)

```
proc format;  /*ユーザ定義フォーマットの作成*/
  value $a
    "fujita"="藤田です。"
    "tanaka"="田中です。"
    other="その他です。";
run;
proc print data=sample;
  format name $a.;  /*変数 name の値に 定義したフォーマット($a)を適用して表示*/
run;
```

(ログ)

```
NOTE: 出力形式 $A を作成しました。
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	藤田です。	M	30	175	70
2	2	その他です。	F	29	168	59
3	3	その他です。	M	32	180	85
4	4	田中です。	M	40	178	77

(プログラム 1.1-5)

```
proc catalog cat=formats; /*SASカタログを管理するプロシジャ*/
  contents;   /*カタログ内のフォーマット定義名を確認*/
run;
```

(アウトプット)

```
                    カタログの内容 : WORK.FORMATS

#    名前    タイプ                作成日                更新日      説明
-----------------------------------------------------------------------
```

7 LENGTH ステートメントで長さを定義しなかった場合は８バイトに設定されます。

1	A	FORMATC	27SEP2010:14:49:12	27SEP2010:14:49:12

[一時ライブラリと永久ライブラリ　libname ステートメント]

SAS ファイルは、SAS 起動時に自動的に割り当てられた **WORK** と呼ばれる**ライブラリ参照名**で参照する物理ディレクトリの中に一時的に作られ、SAS 終了時に自動的に消去されます。WORK という名前以外のライブラリ参照名でユーザ指定した物理ディレクトリに保存するよう指定すると SAS 終了後も残ります。このようにして保存した SAS ファイルを永久 SAS データセットあるいは永久 SAS ファイルと呼びます。

(プログラム 1.1-6)

```
libname mydata "C:¥temp"; /*物理ディレクトリ C:¥temp をライブラリ参照名 mydata で参照させる*/
data mydata.sample;       /*永久データセットの作成*/
  input ID name $ sex $ age height weight; /*変数の名前と型を決めて読み込みを実行する文*/
cards;                    /*データはこれ以降に入力するという意味の文*/
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;                         /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=mydata.sample;      /*データ部を表示するプロシジャ*/
run;
proc contents data=mydata.sample; /*ディスクリプタ部を表示するプロシジャ*/
run;
```

しかしながら、SAS ユーザの便宜にために、SAS システムはセッション開始時にユーザドキュメント(c:¥USERS¥DOCUMENTS)の下の My SAS Files¥9.1 フォルダを **SASUSER** というライブラリ参照名で自動的に割り当てています。これを利用すれば、libname ステートメントを実行しなくても自由に SAS ファイルを永久保存することができます。

(プログラム 1.1-7)

```
data sasuser.sample;       /*SASUSERライブラリに永久データセットを作成する*/
  input ID name $ sex $ age height weight; /*変数の名前と型を決めて読み込みを実行する文*/
cards;                    /*データはこれ以降に入力するという意味の文*/
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;                         /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sasuser.sample;      /*データ部を表示するプロシジャ*/
run;
proc contents data=sasuser.sample; /*ディスクリプタ部を表示するプロシジャ*/
run;
```

［命名規則(SAS 名の規則)］

SAS データセット名、変数名などつけ方には基本的に以下の規則が適用されます。

データセット名、変数名の命名規則
・長さ 32 文字以内（すべて半角)。
・先頭の 1 文字はアルファベット（A～Z）もしくはアンダースコア（_）のいずれかでなければなりません。
・2 文字目以降はアルファベット（A～Z）もしくはアンダースコア（_）もしくは数字（0～9）のいずれかが使えます。

名前の中に漢字やブランクが使えないことに注意してください。[8]
なお、データセット名、変数名のアルファベットの大文字小文字の区別については、以下のように取り扱われます。

データセット名、変数名の大文字小文字の区別
・データセット名は大文字小文字の区別はなく、すべて SAS 内部で大文字として認識されます
・変数名は DATA ステップや PROC ステップのプログラミングレベルおよび実行中は大文字小文字の区別はなく同じ文字として認識されます。 ただし、データセットに格納する変数名や SAS からの出力では最初に定義したとおりに大文字小文字を区別して保存され表示されます。[9]

基本的には、同じ命名規則が、配列名、ステートメントに置くラベル名、マクロ定義名、マクロ変数名、カタログ名、ライブラリ参照名などにも適用されます。ただし、ユーザー定義フォーマット名では文字フォーマット名は"$"で始まること、またライブラリ参照名などは OS の制約も受けるなど若干異なる場合もありますので注意が必要です。特に C 言語など大文字小文字を区別するプログラミング言語で書かれたプログラムを SAS 言語で書き直すような場合は注意が必要です。

(問題)

★SAS 名(データセット名または変数名)として有効か無効かを考えて自分で確認してください。
A20101010
A2010/10/10
_2010_10_10
ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdef

［ステートメント(文)］

SAS 言語には通常のプログラミング言語と同じようにプログラミングステートメント（文）が用意されています。 1 個のステートメントは、**キーワード（定型語）**で始まり**セミコロン（;）**で終了します。キーワードとセミコロン以外にステートメントに含まれる他の要素は、変数名、データセット名、関数名、フォーマット名、定数、パラメータなどの 1 文字以上の長さを持つワード（語）と演算子、記号およびブランク（空白）などの 1 文字の長さの特殊文字です。 ワードの区切り文字はブランク 1 個以上です。 ただし、演算子（+-*/など）や記号（¥やカッコや引用符など）も通常ワード間の区切り文字として認識されます。そのため、これらの特殊文字をワードとワードの間に記述する場合、ブランクはあってもなくてもかまいません。 また、1 つのステートメントを複数行にわたって書いても、1 つの行に複数のステートメントを書いてもかまいません。 ただし、ワードの途中で改行することは許されません。 なお、キーワードを含むワードのアルファベットは大文字（A～Z）・小文字（a～z）いずれで書いてもかまいません。

[別表 1] に SAS の DATA ステップで使える主要なステートメントの一覧を表示しましたので、ざっと見てみましょう。

8 変数名には 256 文字までのラベルを付けることができ、漢字も使えます。

9 CONTENTS プロシジャを用いて変数名をデータセット出力するような場合に注意が必要となります。

[別表 2] に SAS の Base プロダクトで使える主要な PROC ステップの一覧を表示しましたので、ざっと見てみましょう。
[別表 3] に SAS の主要なグローバルステートメントの一覧を表示しましたので、ざっと見てみましょう。

［関数］

SAS は豊富な関数を備えています。統計関数や分布関数や乱数発生関数といったデータ解析のための関数は勿論のこと、文字関数なども充実しています。

[別表 5] に主要な関数を表示しましたので見てみましょう。

［演算子］

[別表 4] に SAS で使える主要な演算子を表示しましたので見てみましょう。

ちなみに、SAS の論理演算結果は真の場合は値 1 を返し、偽の場合は値 0 を返します。
(プログラム 1.1-8)

```
data _null_;              /*データセットを作らないでDATAステップを開始する*/
  set sample;             /*SASデータセットsampleを読み込む*/
  check=(height>=170);    /*height>=170の真偽値を変数checkに格納する*/
  put height= check=;     /*変数height値とcheck値をログに書き出す*/
run;
```

(ログ)

```
height=175 check=1
height=168 check=0
height=180 check=1
height=178 check=1
```

［欠損値］

SAS 言語では数値タイプ変数の欠損値はピリオド 1 個(.)で与えます。また標準的な表示も同じです。また._および.A～.Z までの 27 個の特殊欠損値を通常の欠損値と区別して持たせることが可能です。[10]
一方、文字タイプ変数の欠損値はヌル値""またはブランク 1 個" "で与えます。新たに定義する文字変数の場合はどちらも長さ 1 のブランクの値として SAS データセットに格納され、既に存在する文字変数の場合は定義された長さのブランク文字列を値として SAS データセットに格納されます。文字タイプ変数の欠損値には特殊欠損値はありません。また、長い方の長さに足りない方の文字列はブランクを埋めてから比較されますので、長さの異なる文字タイプ欠損値の比較結果は等しくなります。
欠損値は統計量を計算する上での有効な値としてカウントせず、別途取り扱われます。文字タイプの欠損値も標準では集計表の画面には現れないなど別扱いされる場合がほとんどです。欠損を表す値を"999999"と入力するような他のシステムから SAS に乗り換える場合は SAS の欠損値を考慮した変換が必要です。

欠損値を定数で割り当てます。
(プログラム 1.1-9)

```
data kesson;
  a="";
  b=" ";
  c="  ";
  d=.;
```

10 欠損値にも比較順序があり ._<.<.A<…<.Z の順となっています。

```
  e=.a;
run;
proc print data=kesson;run;
proc contents data=kesson;run;
```

(PRINT アウトプット)

```
OBS    a    b    c    d    e

 1                   .    A
```

(CONTENTS アウトプットの一部)

```
    変数と属性の昇順リスト

#    変数    タイプ    長さ

1    a       文字        1
2    b       文字        1
3    c       文字        2
4    d       数値        8
5    e       数値        8
```

カードデータから欠損値を入力します。fujita さんの性別と suzuki さんの年齢と tanaka さんの体重を欠損値として入力します。(注意：このようなカードイメージデータからのリスト入力方法のときは、ブランクはデータの区切り文字として認識されるため、文字タイプ変数の欠損値もピリオド 1 個で与えます。)

(プログラム 1.1-10)

```
data sample;
  input ID name $ sex $ age height weight;
cards;
001 fujita . 30 175 70
002 suzuki F . 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 .
;
proc means data=sample;  /*基本統計を計算するプロシジャ*/
run;
proc freq data=sample;
  table sex;
run;
```

(アウトプット)

```
MEANS プロシジャ

変数       N            平均         標準偏差          最小値          最大値
-------------------------------------------------------------------------------
ID         4       2.5000000       1.2909944       1.0000000       4.0000000
age        3      34.0000000       5.2915026      30.0000000      40.0000000
height     4     175.2500000       5.2519838     168.0000000     180.0000000
weight     3      71.3333333      13.0511813      59.0000000      85.0000000

FREQ プロシジャ

                                        累積          累積
sex       度数       パーセント       度数       パーセント
-----------------------------------------------------------
```

```
F              1        33.33              1        33.33
M              2        66.67              3       100.00

欠損値の度数 = 1
```

［エラー対応］

ログはプログラムの実行状況を報告する役割を持ち、**エラー(ERROR)**、**警告(WARNING)**、**ノート(NOTE)**の各メッセージにより実行状況を確認できます。
これまでの例ではエラーと警告は発せられず、ノートのみのメッセージとなっており、少なくとも文法的なエラーや実行時のエラーは出現しなかったということを意味しています。 ただし、意図した結果が得られたかどうかは、ノートに記された作成されたデータセットのオブザベーション数と変数の数が１つのポイントとなります。 特に作成したデータセットのオブザベーション数が 0（ゼロ）になっていないかどうかを確認することが実際上は重要です。 エラーが無い場合でもこの部分は必ず確認しておきましょう。

エラーは**コンパイル時のエラー（文法エラー）**と**実行時のエラー**に大きく分かれます。

文法エラー以下のような場合に発生します。

・キーワードのタイプミス（存在しないプロシジャ名や関数名、ステートメントやオプション）
・ワード間のブランク忘れ
・文末のセミコロン（;）忘れ
・全角のブランクを入力してしまった場合
・Do～End 文のネストで対応がとれていない場合

文法エラーはプログラムのコンパイル時に発見され、実行は中止されます。このような場合は**最初に発生したエラーメッセージの近辺に注意**してタイプミス、**全角ブランクの有無**、**セミコロンの有無**などを調べて訂正します。

なお、よりやっかいなのは**引用符(”または’)が片方閉じていないまま**になっている場合やマクロ処理を行っている場合で問題が起きた場合です。このような場合はエラーメッセージすら出てこないときがあり、何度サブミットを行っても SAS からの応答が無い状態になることがあります。この場合は、以下のおまじないで解決できる場合がありますので試してください。

(おまじないのプログラム)

```
;*';*";*/;quit;run;
```

実行時エラーは以下のような場合に発生します。

・データ入力において読み取ろうとするデータ形式と読み取り形式が異なる、対応がとれない場合など
・未定義の変数に対する処理を行おうとした場合
・無効な関数の引数の指定（存在しない日付を指定した場合など）
・ゼロで割る処理を行った場合

このような場合、実行は中断されず、結果を欠損値に設定するなどして実行が続きます。

なお、Windows のコンピュータ資源不足や書き込み禁止などのファイルセキュリティなどの理由でエラーが発生する場合もあります。

1.2 データ読取り・変数作成と横方向の集計処理
~ FILENAME、LIBNAME、INPUT、SET、IMPORT、EXPORT、割り当てステートメント、集計関数、データセットオプション、PRINT、SORT プロシジャ

SAS でデータの集計や分析等の処理を行うためには、通常、集計や分析に用いるデータを一旦 SAS データセットに読み込む必要があります。SAS ファイル以外のファイルを SAS から見て「**外部ファイル**」と呼んでいます。まず、外部ファイルから SAS データセットにデータを読み込むために用いる DATA ステップの SAS ステートメントとして重要な INPUT ステートメントと INFILE ステートメントを主に学びます。次に SAS データセットからのデータの読込を行う SET ステートメント、CSV 形式などの外部ファイルの入力に用いることができる IMPORT プロシジャなどを学びます。

さらに DATA ステップで用いるステートメントの中で割り当てステートメントその他のプログラミングステートメントの一部、関数の一部、データセットオプションなどを学びます。
また、PRINT, SORT などのプロシジャの文法を理解します。

[INPUT ステートメントによるデータの読み取り]
まず、プログラムの中に読み取りデータを書いた例から始めます。
冒頭に出てきたプログラム 1.1-2 をもう一度取り上げます。
下記のプログラムをエデイタに呼び出してください。
なお、そのエデイタ画面をアクティブにした上で、書いたプログラムのどの部分も選択せずにサブミットすると、そのエディタ画面に書かれたプログラム全部が一度にサブミットされます。エラーが発生したときは、上手くいった DATA ステップや PROC ステップ部分は再実行する必要ありませんので、修正したプログラム部分を含む DATA ステップもしくは PROC ステップ以降を選択してからサブミットしてください。
また、/* */ で囲んだテキスト部分はコメントですので、入力してもしなくても結構です。

(プログラム 1.1-2)

```
data sample; /*データセットの作成*/
  input ID name $ sex $ age height weight; /*変数の名前と型を決めて読み込みを実行する文*/
cards;       /*データはこれ以降に入力するという意味の文*/
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;            /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

SAS システム 1
2010 年 10 月 10 日 日曜日 午後 03 時 29 分 36 秒

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	77

(DATA ステートメント)　宣言ステートメント
DATA ステップを開始し作成する SAS データセット名を明示的に指定します。ここでは sample という名前の SAS データセットを WORK ライブラリに 1 個作成するという宣言をしています。 DATA ステップは 1 つの DATA ステートメントで開始し、RUN ステートメントで明示的に終了します。また他の DATA ステートメントや PROC ステートメントに遭遇することによっても終了します。

・データセット名を省略すると _data_ という自動 SAS データセット名が指定されたものとみなされます。これは SAS セッションの最初に指定された場合は DATA1 という名前、2 番目は DATA2, 以下同様に DATA3,...,DATAn という名前の SAS データセットを作成します。
(例) data; set sample; run;
・同時に複数の SAS データセット名を指定可能です。入力データから条件検索によって別々のデータセットを作成する場合などに使います。
(例) data male female; set sample; if sex="M" then output male; else output female; run;
・データセットを作成しないで DATA ステップを実行する特殊なデータセット名 _null_ があります。
(例) data _null_; set sample; if sex="M" then put _all_; run;

(INPUT ステートメント)　実行ステートメント
INPUT ステートメントは外部データのレコード行から、どのデータをどういう変数名と型で読み取るかを指定し実行するステートメントです。ここでは「**リスト入力**」と呼ばれる最も単純な方法で、読み取りたい項目の変数名と型をリストするだけの指定を行っています。

(CARDS ステートメント)　宣言ステートメント
CARDS ステートメントは INPUT ステートメントで読み込むデータはこのステートメントの次の行から 1 個のセミコロン(;)が書かれた行の間に入っていることを知らせるステートメントです。

[INPUT ステートメントの指定方法]
3 通りの入力方法と 6 つの修飾子(modifier) (@,@@,+,#,:,/)について学びます。

[(1) リスト入力]
(INPUT ステートメントの指定)

```
   input ID name $ sex $ age height weight;
```

データ項目はブランク 1 個以上を区切り文字としてテキスト形式で並んでいる場合に用いることができます。CARDS ステートメントで入力するデータがある場合は、指定が簡単なので良く使われます。

データ項目はブランクで区切られていますのでリスト入力向きです。
(1 行目のデータ行)

```
001 fujita M 30 175 70
```

・データ項目の並び順に対応して、SAS データセットになる変数名と型 (文字型の場合は$, 数値型の場合は何も指定しません.)を順次指定していきます.
・欠損値はピリオド 1 個で入力されている必要があります.
上記データの並びに対応して、

ID=1　(数値)
name="fujita"　(文字)

sex="M"　(文字)
age=30　(数値)
height=175　(数値)
weight=70　(数値)

の値を持つオブザベーションが input ステートメントによって読み込まれます。

この**リスト入力には次の制約**があります。
(A)文字型データを読む際に 8 バイトの長さに切られます。("takahashi"は"takahash" と最初の 8 文字までしか読まれていません。)
(B)テキスト形式の値しか読み取れません(バイナリ形式などの編集形式のデータは読み取れません)。

(A)の問題は次のカラム入力やフォーマット入力でも解決できますが、フォーマット入力とリスト入力をミックスした**コロン(:)フォーマット指定付きリスト入力** が実用的です。なおコロン(:)は INPUT ステートメントで使える修飾子の 1 つです。

[コロンフォーマット指定による INPUT ステートメントの指定]

(プログラム 1.2-1)

```
data sample;  /*データセットの作成*/
  input ID name :$10. sex :$1. age height weight;
cards;        /*データはこれ以降に入力するという意味の文*/
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahashi	M	32	180	85
4	4	tanaka	M	40	178	77

name :$10. のコロンフォーマット指定付きリスト入力を行ったため、takahashi とちゃんと入力できています。

(B)の問題はフォーマット入力で解決できます。

[(2)カラム入力]

外部データレコード上の各項目のカラム位置が決まっているテキスト形式データを入力する場合に使います。読み取りたいカラムの項目だけ選択して読み取ることができます。

カラム入力は INPUT ステートメントに
変数名　タイプ($もしくは指定なし) 開始カラム位置-終了カラム位置
を 1 セットとして読み取りたい項目分を指定する方法です。

(プログラム 1.2-2)

```
data sample;  /*データセットの作成*/
  input ID 1-3 name $4-13 sex $14 age 15-16 height 17-19 weight 20-21 sahw $14-21;
cards;        /*データはこれ以降に入力するという意味の文*/
001fujita    M3017570
002suzuki    F2916859
003takahashi M3218085
004 tanaka   M4017877
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight	sahw
1	1	fujita	M	30	175	70	M3017570
2	2	suzuki	F	29	168	59	F2916859
3	3	takahashi	M	32	180	85	M3218085
4	4	tanaka	M	40	178	77	M4017877

[(3) フォーマット入力]

フォーマット入力は INPUT ステートメントに

@カラムポインター位置　変数名　読み取り編集形式(INFORMAT)名.

を 1 セットとして読み取りたい項目分を指定する方法です。@は INPUT ステートメントで使える修飾子の 1 つで、その行における読み取りポインタの**絶対カラム位置**を与えます。@の後には正の整数もしくは正の整数を値に持つ変数名を指定します。

(プログラム 1.2-3)

```
data sample;  /*データセットの作成*/
  input @14 sex $1. @4 name $10. @1 ID 3.;
cards;        /*データはこれ以降に入力するという意味の文*/
001fujita    M3017570
002suzuki    F2916859
003takahashi M3218085
004 tanaka   M4017877
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	sex	name	ID
1	M	fujita	1
2	F	suzuki	2
3	M	takahashi	3
4	M	tanaka	4

フォーマット入力は外部データ項目がどのような編集形式であってもほとんど読み取ることができるという意味で SAS が大変強力なデータ入力編集機能を持っている証の 1 つになっています。SAS では特定の編集形式で書かれた外部データ値を SAS 変数値に読み込む場合の編集形式の指定をインフォーマット(入

力フォーマット)と呼んでいます。逆に SAS 変数値を指定の編集形式で外部データ値として書き出す場合の編集形式の指定をフォーマット（出力フォーマット）と呼んでいます。
[別表 7] に主要なインフォーマット、[別表 6] にフォーマットの一覧を表示してますので、見てみましょう。

[+相対カラム位置移動修飾子]

@ポインタ指定がカラムの絶対位置指定であったのに対し、**+修飾子**は同一行の中でのポインタを相対的に移動させる機能を持っています。+の後には整数(負の整数も可)もしくは整数を値として持つ変数名を指定します。

(プログラム 1.2-4)

```
data sample;  /*データセットの作成*/
  input @14 sex $1. +2 height 3. +(-6) sahw $8. @1 ID 3.;
cards;        /*データはこれ以降に入力するという意味の文*/
001fujita    M3017570
002suzuki    F2916859
003takahashi M3218085
004 tanaka   M4017877
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	sex	height	sahw	ID
1	M	175	M3017570	1
2	F	168	F2916859	2
3	M	180	M3218085	3
4	M	178	M4017877	4

[複数レコードをあたかも 1 レコードとして読み取る場合]

1 人の顧客属性項目が複数レコード行にわたって書かれているような場合、修飾子の 1 つである #行ポインタ指定を用いるとうまく読み込むことができます。

(プログラム 1.2-5)

```
data sample;  /*データセットの作成*/
  input #1 @1 ID 3. @4 name $10.
        #2 @1 sex $1. @7 weight 2.;
cards;        /*データはこれ以降に入力するという意味の文*/
001fujita
M3017570
002suzuki
F2916859
003takahashi
M3218085
004 tanaka
M4017877
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	ID	name	sex	weight
1	1	fujita	M	70
2	2	suzuki	F	59
3	3	takahashi	M	85
4	4	tanaka	M	77

#行ポインタ修飾子が INPUT ステートメントに出現すると、外部レコードの行は、#ポインタの最大値の行数を１単位として読み込むモードになります。#の後には正の整数もしくは正の整数値を持つ変数名を指定します。

[複数の INPUT ステートメントの指定]

#ポインタと@ポインタを使うと複数行にわたって、どの行のどのカラム位置からでも自由にデータを読みとれますし、読み取りポインタの位置を前後左右、自由に行き来できるという利点があります。

しかしながら、複数行のレコードを１単位として、１番目の行から順に読み取る場合は、#ポインタを用いずに、複数の INPUT ステートメントを書いて読み込むことができます。

(プログラム 1.2-6)

```
data sample;  /*データセットの作成*/
  input @1 ID 3. @4 name $10.;
  input @1 sex $1. @7 weight 2.;
cards;        /*データはこれ以降に入力するという意味の文*/
001fujita
M3017570
002suzuki
F2916859
003takahashi
M3218085
004 tanaka
M4017877
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

OBS	ID	name	sex	weight
1	1	fujita	M	70
2	2	suzuki	F	59
3	3	takahashi	M	85
4	4	tanaka	M	77

ここで重要なのは、各 INPUT ステートメントの最後のセミコロン(;)はそれぞれ読んでいる外部データ行の読込を終了して、次の外部データ行の最初のカラムに読込ポインタを進める合図だということです。

なお、/修飾子を用いて以下のように指定しても改行して読み取る指定になります。

```
  input ID 3. @4 name $10. / sex $1. @7 weight 2.;
```

注: INPUT ステートメントや/修飾子の後の最初の@1 は省略できます。

[@;で終わる INPUT ステートメント]

@修飾子はカラム絶対位置を指定する役割の他に、INPUT ステートメントを@;で終了させた場合はポインタを次の行に移動させず、直前のデータ読み取り位置に留めておく機能があります。
例えば、最初の 1 カラム目の文字を検索して、値によって次の読み取り項目を変更するような場合に有用です。

(プログラム 1.2-6)

```
data sample;  /*データセットの作成*/
  input point 1. @;
  input +point char $1.;
cards;        /*データはこれ以降に入力するという意味の文*/
1abcdefghijklmn
5abcdefghijklmn
;             /*データをCARDS文で入力する場合のデータ入力終了の合図*/
proc print data=sample;  /*データ部を表示するプロシジャ*/
run;
```

(アウトプット)

```
OBS    point    char

 1       1       b
 2       5       f
```

[@@;で終わる INPUT ステートメント]

@@修飾子で終わる INPUT ステートメントは、カードイメージデータからデータを入力する場合に特に便利です。

(プログラム 1.2-7)

```
data sample;
  input x y @@;
  cards;
1 2 3 4 5 6 7 8
9 10 11 12
;
proc print data=sample;run;
```

(アウトプット)

```
OBS     x     y

 1      1     2
 2      3     4
 3      5     6
 4      7     8
 5      9    10
 6     11    12
```

@@;で終わる INPUT ステートメントは DATA ステップの次のループに対しても外部データ上のポインタ保留位置を維持する役割があります。
@;で終わる INPUT ステートメントに変えて実行してみてください。

(プログラム 1.2-8)

```
data sample;
  input x y @;
  cards;
1 2 3 4 5 6 7 8
9 10 11 12
;
proc print data=sample;run;
```

(アウトプット)

OBS	x	y
1	1	2
2	9	10

@1 個だと DATA ステップはデータ行の数だけのループになります。

以上で INPUT ステートメントの説明は終了です。

[FILENAME ステートメントと INFILE ステートメント]

次は、CARDS ステートメントの中ではなく、外部ファイルにデータがある場合にデータを読む方法を学びます。外部ファイルにアクセスする場合、アクセス先を指定する実行ステートメント（INFILE ステートメント(入力)と FILE ステートメント(出力)）において、そのファイルの**物理パス名**を記述する代わりに、あらかじめ FILENAME ステートメントでそのファイルをアクセスするための**ファイル参照名**を定義しておき、INFILE ステートメントや FILE ステートメントでファイル参照名を使用することができます。SAS プログラムでアクセスする外部ファイルを SAS プログラムコードの冒頭に書いておけるのでプログラムが大変読みやすくスマートになります。

練習用に、以下のデータ項目を SAS エディタもしくはメモ帳で作成して、ユーザディレクトリ(C:USERS¥ユーザ名¥DOCUMENTS¥My SAS Files¥9.1 など)の中に sample1.dat という名前で保存してください。

(sample1.dat の内容)

```
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
```

(プログラム 1.2-9)　**注意 ユーザ名 の箇所はあなたのログオンユーザ名を入力してください。**

```
filename in "c:¥users¥ユーザ名¥documents¥my sas files¥9.1";
data sample;
  infile in(sample1.dat);
  input ID name $ sex $ age height weight;
proc print data=sample;
run;
```

(ログ)

```
179  filename in "c:¥users¥hideo¥documents¥my sas files¥9.1";
180  data sample;
181    infile in(sample1.dat);
182    input ID name $ sex $ age height weight;

NOTE: 入力ライブラリ IN:
      ディレクトリ=c:¥users¥ユーザ名¥documents¥my sas files¥9.1

NOTE: 入力ファイル IN(sample1.dat) :
      ファイル名=c:¥users¥ユーザ名¥documents¥my sas files¥9.1¥sample1.dat,
      レコードフォーマット=V,論理レコード長=256

NOTE: 4 レコードを入力ライブラリ IN から読み込みました。
       最小レコード長は 22 です。
       最大レコード長は 25 です。
NOTE: 4 レコードを入力ファイル IN(sample1.dat) から読み込みました。
       最小レコード長は 22 です。
       最大レコード長は 25 です。
NOTE: データセット WORK.SAMPLE は 4 オブザベーション、 6 変数です。
NOTE: DATA ステートメント 処理 (合計処理時間):
       処理時間              0.04 秒
       CPU 時間              0.00 秒


183  proc print data=sample;
184  run;

NOTE: データセット WORK.SAMPLE から 4 オブザベーションを読み込みました。
NOTE: PROCEDURE PRINT 処理 (合計処理時間):
       処理時間              0.06 秒
       CPU 時間              0.01 秒
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	77

(FILENAME ステートメント指定方法)　宣言ステートメント
FILENAME ファイル参照名 "物理パス名";

(INFILE ステートメントの指定方法)　実行ステートメント
INFILE ファイル参照名 オプション;

ただし、ファイル参照名は上記プログラム例のようにフォルダに指定した場合は (データセット名)を指定しフォルダ内の特定のファイルを参照させる必要があります。
ファイル参照名は SAS 名の制約を受けます。ただし長さは 8 文字以内です。

また、INFILE ステートメントで指定できる**特殊なファイル参照名として CARDS** があります。
これは CARDS ステートメントの後に入力したデータのレコード形式(dsd や dlm=オプション)を指定でき

るようにするためです。(ただしカードイメージデータの最大レコード長はデフォルトの 256 を超えて設定不能です。)

その他 EXCEL ファイルを読むときのための **DDE 指定**などがあります。

[INFILE ステートメントで良く使うオプション]

・lrecl=論理レコード長
省略すると **256 が設定されます**ので、固定長の外部ファイルを読むときはその長さ、csv 形式などの可変長の外部ファイル を読む場合は lrecl=32000 と大きな値を指定します。
・recfm=レコードフォーマット
OS 環境によって指定可能な値が異なりますが、Windows 環境では
f(固定長),v(可変長),n(不定長)の 3 つのいずれかになります。
・dsd　オプション　と
・dlm="区切り文字" オプション
csv 形式の可変長レコードを読むときに必要になります。(ただし後述の IMPORT プロシジャのおかげで csv 形式のファイルは DATA ステップを使わなくても読めますが、変数のタイプなどを的確に指定したい場合などのために知っておいた方が良いと思います。)

(プログラム 1.2-10) INFILE CARDS 指定と dsd,dlm=オプション

```
data sample;
  infile cards dsd dlm=",";
  input ID name $ sex $ age height weight;
  cards;
001,fujita,M,30,175,70
002,suzuki,F,29,168,59
003,takahashi,M,32,180,85
004,tanaka,M,40,178,,
;
proc print data=sample;
run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	.

以上のように、外部ファイルをアクセスする場合、読み取りは INFILE ステートメントと INPUT ステートメントを用います。(逆に、後述するように、書き込みは FILE ステートメントと PUT ステートメントを用います。)

なお、FILENAME ステートメントを省略して、以下のように書いても実行できます。

```
data sample;
  infile "c:¥users¥hideo¥documents¥my sas files¥9.1¥sample1.dat";
  input ID name $ sex $ age height weight;
```

```
run;
```

しかしながら、プログラムの冒頭で、そのプログラムがアクセスする物理ファイルをすべてあらかじめ FILENAME ステートメントでファイル参照名を定義しておくと、プログラムが見やすくなります。

[SET ステートメントによる SAS データセットの読込み]

FILENAME ステートメント、CARDS ステートメント、INFILE ステートメント、INPUT ステートメントは外部データの読込に用いられるステートメントでした。一方、SAS データセットの値を読み込むステートメントは LIBNAME ステートメントと SET、MERGE、UPDATE ステートメントです。
一旦 SAS データセットにデータが格納されていると、SAS データセットはデータ部以外にディクショナリ部を持っていますので、データセット名を指定するだけで SAS はすべての項目の情報が分かってしまうということです。SET、MERGE、UPDATE の各ステートメントの違いは後で学ぶことになりますが、ここでは単一の SAS データセットを読み込む場合は LIBNAME ステートメントと SET ステートメントを使えば良いということを覚えておけば十分です。

プログラム 1.1-6 で既に libname の基本は出ていますので、SET ステートメントといくつかのステートメントを使った例を実行してみます。

(プログラム 1.2-11)

```
data sample2;
  set sample;
  if sex="F" then output;
  else delete;
run;
proc print data=sample2;run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	2	suzuki	F	29	168	59

(SET ステートメントの指定方法)　実行ステートメント

SET SAS データセット名 オプション;

SAS データセット名
・複数の SAS データセットをブランクで区切って指定できます。

オプション
・end=変数名
最後のオブザベーションを読んだ時点で指定の変数値=1 となります。

(IF ステートメント)　実行ステートメント
(else ステートメント)　実行ステートメント
条件選択を行うステートメントです。別途詳しく学びます。

(OUTPUT ステートメントの指定方法)　実行ステートメント
SAS データセットにプログラムデータベクトルの値を１オブザベーションとして書き込みます。

OUTPUT 出力 SAS データセット名;

出力 SAS データセット名には複数の SAS データセット名をブランクで区切って指定できます。(同じオブザベーションが出力されます。)

(DELETE ステートメント)　実行ステートメント
このステートメント実行時点でオブザベーションの書き出しを行わずに DATA ステップループの最初に戻ります。(プログラムデータベクトルの値は DATA ステップのループに戻った時点ですべて欠損値にリセットされます。)
なお、OUTPUT ステートメントが明示的に１つでも存在すれば、DATA ステップの最後の OUTPUT ステートメントの自動挿入は行われませんので、この例では else delete; は無くても結果は同じになります。しかし、あった方が明示的です。

[IMPORT プロシジャと EXPORT プロシジャ]

テキストファイル、特に csv ファイル(Comma Separeted Values, カンマ区切りデータ)を読み取るとき重宝するプロシジャです。

まず、EXPORT プロシジャを用いて、SAS データセット SAMPLE の内容を CSV 形式ファイルにします。

(プログラム 1.2-12)

```
proc export data=sample outfile="sample.csv" replace;
run;
```

メモ帳などで現在の作業フォルダ(マイドキュメント)の下に sample.csv が出来ていることを確認してください。

1 行目に SAS データセットの変数名が書き出されている点に注意してください。
残念ながら変数名の書き出しを抑制するオプションはありません。
そうしたい場合は、以下のように options source2; を指定してから proc export を実行すると、ログに export を実行する DATA ステッププログラムが出現しますので、これを編集して再実行すると良いでしょう。

(プログラム 1.2-13) EXPORT プロシジャを DATA ステップに展開したログを得る。

```
options source2;
proc export data=sample outfile="sample.csv" x;
run;
```

(ログ)

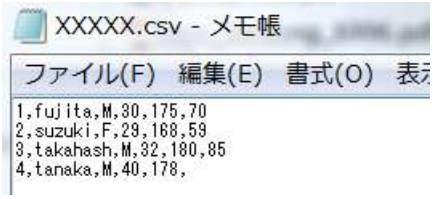
```
143   /*************************************************************************
144   *   プロダクト:   SAS
145   *   バージョン:   9.1
146   *   作成者:       外部ファイルインターフェイス
147   *   作成日:       14OCT10
148   *   説明:         生成された SAS データステップ コード
149   *   テンプレート ソース:  (指定なし)
150   *************************************************************************/
151      data _null_;
152      set  SAMPLE                                          end=EFIEOD;
153      %let _EFIERR_ = 0; /* エラー検出のマクロ変数を設定します */
154      %let _EFIREC_ = 0;     /* エクスポートレコードカウントのマクロ変数をクリアします */
155      file 'sample.csv' delimiter=',' DSD DROPOVER lrecl=32767;
156         format ID best12. ;
157         format name $8. ;
158         format sex $8. ;
159         format age best12. ;
160         format height best12. ;
161         format weight best12. ;
162      if _n_ = 1 then        /* 列名を書き出します */
163       do;
164         put
165         'ID'
166         ','
167         'name'
168         ','
169         'sex'
170         ','
171         'age'
172         ','
173         'height'
174         ','
175         'weight'
176         ;
177       end;
178       do;
179         EFIOUT + 1;
180         put ID @;
181         put name $ @;
182         put sex $ @;
183         put age @;
184         put height @;
185         put weight ;
186         ;
187       end;
188      if _ERROR_ then call symput('_EFIERR_',1);  /* エラー検出のマクロ変数を設定します */
189      if EFIEOD then call symputx('_EFIREC_',EFIOUT);
```

```
190        run;
```

上記ログをエディタ画面にコピーし、上記ログ番号の 162 行目から 177 行目に相当する箇所を削除して、かつ 156 行目の file "sample.csv"; の箇所を file "XXXXX.csv"; に変更した上でサブミットします。

(プログラム 1.2-14)　ログをエディタにコピーし、行番号を削除し、不要な箇所を削除してサブミット

```
/*************************************************************************
*   プロダクト:   SAS
*   バージョン:   9.1
*   作成者:       外部ファイルインターフェイス
*   作成日:       14OCT10
*   説明:         生成された SAS データステップ コード
*   テンプレート ソース:  (指定なし)
*************************************************************************/
   data _null_;
   set  SAMPLE                                          end=EFIEOD;
   %let _EFIERR_ = 0; /* エラー検出のマクロ変数を設定します */
   %let _EFIREC_ = 0;     /* エクスポートレコードカウントのマクロ変数をクリアします */
   file 'XXXXX.csv' delimiter=',' DSD DROPOVER lrecl=32767;
      format ID best12. ;
      format name $8. ;
      format sex $8. ;
      format age best12. ;
      format height best12. ;
      format weight best12. ;
    do;
      EFIOUT + 1;
      put ID @;
      put name $ @;
      put sex $ @;
      put age @;
      put height @;
      put weight ;
      ;
    end;
   if _ERROR_ then call symput('_EFIERR_',1);  /* エラー検出のマクロ変数を設定します */
   if EFIEOD then call symputx('_EFIREC_',EFIOUT);
   run;
```



(EXPORT プロシジャの指定方法)　SAS データセットの値を外部テキストファイルに書き出す。

PROC EXPORT DATA=入力ＳＡＳデータセット名　OUTFILE="出力外部データセット名" オプション;

(PROC EXPORT ステートメントの良く使うオプション)

・REPLACE
既存の出力ファイルの内容を置き換えることを許可します。

次に今書き込んだ CSV 形式ファイルを INPORT プロシジャを用いて SAS データセットに読み込みます。

(プログラム 1.2-15)　IMPORT プロシジャの実行

```
proc import datafile="sample.csv" out=samplex replace;
run;
proc print data=samplex;run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	.

読み取る sample.csv の 1 行目を SAS 変数名とみなして自動的に読み取っている点に注意してください。

(IMPORT プロシジャ)　外部テキストファイルを SAS データセットに読み込む

PROC IMPORT DATAFILE="入力外部データセット名" OUT=出力 sas データセット名　オプション;
オプションデータソースステートメント;

(PROC IMPORT ステートメントの良く使うオプション)
・REPLACE
既存の出力 SAS データセットの内容を置き換えることを許可します。

(良く使われるオプションデータソースステートメント)
・GETNAMES=YES | NO
1 行目を変数名として読み取るかどうかを選択します。デフォルトは YES です。
・GUESSINGROWS=1-32767
変数の型の推測を読み取る外部データを何行読んでから決定するかを指定します。
・DATAROW=正の整数
何行目から読み始めるかを指定します。

(プログラム 1.2-16)　1 行目を変数名として読む getnames=yes(デフォルト)の効果

```
proc import datafile="XXXXX.csv" out=samplex replace;
run;
proc print data=samplex;run;
```

(アウトプット)

OBS	_	fujita	M	_0	_75	VAR6
1	2	suzuki	F	29	168	59
2	3	takahash	M	32	180	85
3	4	tanaka	M	40	178	.

(プログラム 1.2-17)　1 行目をデータ行として読む getnames=no 指定の追加

```
proc import datafile="XXXXX.csv" out=samplex replace;
  getnames=no;
run;
proc print data=samplex;run;
```

(アウトプット)

OBS	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6
1	1	fujita	M	30	175	70
2	2	suzuki	F	29	168	59
3	3	takahash	M	32	180	85
4	4	tanaka	M	40	178	.

[Excel への書き出し]

Excel に直接書出すには DDE を使います。[11]

DDE(Dynamic Data Exchange) は Windows の機能の 1 つで、対応する別々のアプリケーション間での動的なデータ交換機能を実行します。SAS も Excel も DDE に対応していますので、SAS からは FILENAME ステートメントで dde エンジンを指定して呼び出します。)

Excel を起動し Sheet1 シートが開いた状態にあることを確認してから、以下のプログラムを実行してください。

(プログラム 1.2-18) DDE を使った Excel への書き込み

```
filename out dde "Excel|Sheet1!r1c1:r4c6";
data _null_;
  set sample;
  file out lrecl=32000;
  put id name sex age height weight;
run;
```

FILE ステートメントと put ステートメントは INFILE ステートメントと INPUT ステートメントの逆に、DATA ステップで外部ファイルにデータを書きだすためのステートメントです。

(FILE ステートメント)　実行ステートメント
INFILE ステートメントと同じようなオプションが使えます。

(PUT ステートメント)　実行ステートメント
INPUT ステートメントと同じような指定（リスト出力、カラム出力、フォーマット出力および@,#などの

11 EXPORT プロシジャで直接書き込みを行うためには SAS/ACCESS to PC File Formats プロダクトが必要です。

修飾子）が使えます。

[割り当てステートメント]

一般のプログラミングステートメントにあるものと同じような実行ステートメントです。

変数名=式;

右辺の式の値を左辺の変数の値に割り当てます。

[集計関数]

データセットのデータ値を横方向に集計したい場合は DATA ステップの集計関数を用います。

統計(集計)	CSS	修正済平方和	x=css(5,10,20,16,0,5);
	CV	変動係数(%表示)	x=cv(5,10,20,16,0,5);
	KURTOSIS	尖度	x=kurtosis(5,10,20,16,0,5);
	MAX	最大値	x=max(5,10,20,16,0,5);
	MEAN	平均値	x=mean(5,10,20,16,0,5);
	MIN	最小値	x=min(5,10,20,16,0,5);
	N	非欠損値の数を返す	n=n(1,3,.,5,10);
	NMISS	欠損値の数を返す	nmiss=nmiss(1,3,.,5,10);
	RANGE	範囲	x=range(5,10,20,16,0,5);
	SKEWNESS	歪度	x=skewness(5,10,20,16,0,5);
	STD	標準偏差	x=std(5,10,20,16,0,5);
	SUM	合計	x=sum(5,10,20,16,0,5);
	USS	修正前平方和	x=uss(5,10,20,16,0,5);
	VAR	不偏分散	x=var(5,10,20,16,0,5);

(プログラム 1.2-19)　集計関数の例

```
data sales;
 input dept$ s01-s05;
 tot_sales=sum(s01,s02,s03,s04,s05);
 avr_sales=mean(of s01-s05);
 active_year=n(of s:);
 cards;
A 125 100 181 201 225
B 80 115 300 262 193
C 10 11 12 . .
;
proc print data=sales;run;
```

(アウトプット)

OBS	dept	s01	s02	s03	s04	s05	tot_ sales	avr_ sales	active_ year
1	A	125	100	181	201	225	832	166.4	5
2	B	80	115	300	262	193	950	190.0	5
3	C	10	11	12	.	.	33	11.0	3

sum()は合計、mean()は平均、n()は非欠損データ件数を返す関数です。mean は Excel の関数名と異なる点に注意。

[変数リスト省略のための修飾子]

上記 INPUT ステートメントや関数の中で個々の変数名をブランクたカンマで区切ってリストする代わりに　変数名 prefix+数字-変数名 pefix 数字 の形式でハイフン(-)省略指定が使えます。
そのほかにも、ハイフンハイフン (--)とコロン(:) 省略指定があります。

例)
・x1-x100
変数 x1,x2,...,x100 の 100 個の変数名を指定。
・s03--avr_sales
プログラムデータベクトルに定義された変数名の並び順で s03 を開始変数名、avr_sales を終了変数名としてその間に存在する全変数名を指定。
・s:
プログラムデータベクトルに定義された変数の中で s で始まるすべての変数名を指定。

[データセットオプション]

SAS データセットを読み書きする場合、特定の検索条件やインデクスをつけたり、データセットにパスワードや圧縮指定をおこなったりすることができます。これをデータセットオプションと呼びます。

(プログラム 1.2-20)　データセットオプションの例

```
data sales2(compress=yes password=himitsu index=(dept));
  set sales(where=(tot_sales>=800));
run;
proc contents data=sales2;run;
data sales3;
  set sales2;
run;
```

(ログ)

```
NOTE: データセット WORK.SALES から 2 オブザベーションを読み込みました。
      WHERE tot_sales>=800;
NOTE: データセット WORK.SALES2 は 2 オブザベーション、 9 変数です。
NOTE:  圧縮によって、データセット WORK.SALES2 のサイズを 50.00 パーセント 増加しました。
       圧縮時のサイズは 3 ページです。 ( 非圧縮時は 2 ページが必要です )
NOTE: DATA ステートメント 処理 (合計処理時間):
       処理時間              0.14 秒
       CPU 時間              0.01 秒
```

(アウトプット)

```
CONTENTS プロシジャ

データセット名        WORK.SALES2                              オブザベーション数          2
メンバータイプ        DATA                                     変数の数                    9
エンジン              V9                                       インデックス数              1
作成日時              2010 年 10 月 14 日 木曜日 午後 03 時 36 分 50 秒    オブザベーションのバッファ長  72
更新日時              2010 年 10 月 14 日 木曜日 午後 03 時 36 分 50 秒    削除済みオブザベーション数    0
保護                  READ/WRITE/ALTER                         圧縮済み                    CHAR
データセットタイプ                                             再利用スペース              NO
ラベル                                                         オブザベーションへのポイント  YES
データ表現            WINDOWS_32                               ソート済み                  NO
エンコード            shift-jis  Japanese (SJIS)

                                エンジン/ホスト関連情報
```

```
データセットのページサイズ  4096
データセットのページ数      3
インデックスのページサイズ  4096
インデックスのページ数      2
データセットの修復数        0
ファイル名                  C:¥Users¥Hideo¥AppData¥Local¥Temp¥SAS Temporary Files¥_TD5300¥sales2.sas7bdat
作成したリリース            9.0101M3
作成したホスト              WIN_PRO


       変数と属性の昇順リスト

#    変数           タイプ    長さ

9    active_year    数値         8
8    avr_sales      数値         8
1    dept           文字         8
2    s01            数値         8
3    s02            数値         8
4    s03            数値         8
5    s04            数値         8
6    s05            数値         8
7    tot_sales      数値         8


 インデックスと属性の昇順リスト

                    一意な
#    インデックス    値の数

1    dept                2
```

(主なデータセットオプション)

・FIRSTOBS=

・OBS=

SAS データセットを読み取るときにのみ有効。オブザベーションの読み取り開始番号と終了番号をそれぞれ指定します。

・where=(条件)

()でくくって SQL の where 条件を指定します。IN 節などで複数のアイテムを指定する場合の区切り文字は SQL の区切り文字カンマ(,)でも SAS の区切り文字ブランク 1 個以上のいずれでも良い仕様になっています。

例) where=(dept in ("A" "B") or uriage>=1000)

・keep=変数リスト　または

・drop=変数リスト

作成する、または読み込むデータセットの一部の変数のみ保存する、または除外することを指定します。

・compress=YES | NO

作成する SAS データセットの圧縮指定です。ログに圧縮率が表示されますが、必ずしも容量が減るとは限りません。

・index=(インデクス変数)

・password=パスワード

これらは SAS データセットをデータベースのように使う場合に便利な機能です。

[PRINT プロシジャ]

PROC PRINT data=入力データセット名 オプション;
VAR 変数リスト;
BY 変数リスト;
ID 変数リスト;
SUM 変数リスト;
RUN;

(主なオプション)
・LABEL
ラベルが定義された変数は、変数名の代わりに変数ラベルが表示されるようになります。この指定を行わないと、たとえ変数ラベルが定義されていても PRINT アウトプット出力に変数ラベルは使われない点に注意。
・NOOBS
標準で出力される一番左にオブザベーション番号の表示を抑制します。

(VAR ステートメント)
PRINT 表示したい変数名をリストします。全プロシジャで共通に使えるステートメント。

(BY ステートメント)
この変数値ごとに、別々に出力されます。全プロシジャで共通に使えるステートメント。
前もって指定された変数の値の大きさの順に入力データセットが並んで（ソートされて）いることが必要です。

(BY ステートメントで指定する**変数の前**につけるオプション)
DESCENDING
ソートシーケンス(アルファベット順)の逆順(通常**降順**と呼びます)にオブザベーションが並んでいる場合に指定します。指定が無いとその変数はアルファベット順(**昇順**)に並んでいるものとみなされます。

例)
BY dept descending sales;
dept のアルファベット順の中で sales が大きいものから小さいものの順にオブザベーションが並んでいることを知らせています。

(ID ステートメント)
識別に用いたい変数名をリストします。指定された変数値が一番左に表示されるようになり、同じ ID 値のオブザベーションは最初のオブザベーションのみ値を表示するようになります。このステートメントは多くのプロシジャで共通に指定できるステートメントです。

(SUM ステートメント)
数値タイプ変数名をリストできます。全オブザベーションについての合計値や(BY ステートメントが指定されていれば)小計値が表示されるようになります。PRINT プロシジャ独自のステートメントです。

[SORT プロシジャ]
オブザベーションを指定の変数値の大きさによって並べ替えを行います。

PROC SORT data=入力データセット名 out=出力データセット名　オプション;

BY 変数リスト;
RUN;

(主なオプション)
・NODUPKEY
BY ステートメントの値が同じオブザベーションは最初のオブザベーションのみ残して残りを削除します。この指定によって出力データセットにおける BY 変数値はユニークになります。このオプションを使った SORT プロシジャはキー変数の重複が存在するかどうかをチェックする目的で良く使います。

(BY ステートメント)
並べ替えを行う変数名をリストします。
(BY ステートメントで指定する**変数の前**につけるオプション)
DESCENDING
後に指定した 1 個の変数をソートシーケンス(アルファベット順)の逆順(通常**降順**と呼びます)にオブザベーションを並べ替えます。 指定が無いとその変数はアルファベット順(**昇順**)に並べ変えます。

[SORT プロシジャの重要な注意点]
PROC SORT ステートメントで **out=出力データセット名を指定しなかった場合**、data=入力データセットが並べ替えられたデータセットに置き換わってしまいます。これは一般に**復元できません**ので大変重要な注意点です。

(プログラム 1.2-21) SORT と PRINT の例

```
proc sort data=sample out=samp2;
  by sex;
run;
proc print data=samp2 label;
  var height weight age;
  by sex;
  id sex;
  sum age;
  label height="身長";
run;
```

(アウトプット)

```
sex    身長    weight    age

 F      168       59      29

 M      175       70      30
        180       85      32
        178        .      40
---                        ---
 M                         102
                           ===
                           131
```

以上で 1.2 データ読み取り・変数作成と横方向の集計処理の学習が終わりです。

2. データ加工
2.1 ファイルの連結・マージ・更新を含むプログラミングステートメント

~ SET,MERGE,UPDATE ステートメント,DO ループ,配列処理

SAS は強力なデータ入出力機能とデータ加工機能を持っています。この節ではデータ加工機能について学習します。まず、SAS データセットの読み取りを行う 3 つのステートメント、SET,MERGE,UPDATE の使い方を学習します。
続いて、配列処理を含む主要なプログラミングステートメントについて学習します。

［データセットの縦の連結： 1 つの SET ステートメント］

既に学習してきたように、SET ステートメントは SAS データセットのオブザベーションを読み取る実行ステートメントです。1 つの SET ステートメントに複数の SAS データセットを指定するとそれぞれの SAS データセットのオブザベーションを逐次的に読み取り DATA ステートメントに指定した出力 SAS データセットにオブザベーションを書きこむことを、読み取る SAS データセットのオブザベーションが尽きるまで自動的にループ実行します。結果的に SET ステートメントに指定した複数データセットは縦方向に連結された形になります。

(SET ステートメントの指定方法)　実行ステートメント

SET SAS データセット名(データセットオプション) オプション;

SAS データセット名
・複数の SAS データセットをブランクで区切って指定できます。
・複数の SAS データセットを指定した場合は連続的に指定された SAS データセットのオブザベーションを読み取ります。

(良く使うデータセットオプション)
・in=変数名
そのデータセットのオブザベーションを読み取っているときはは 1、そうで無いときは 0 の値を持つ一時的な変数を定義し、後に続くプログラムで参照できるようにします。この変数は出力データセットには出力されません。
・rename=(変数名=新変数名)
変数名を変更します。複数の変数名を変更する場合は、
rename=(age=nenrei height=shincho)
というように変数名＝新変数名のパターンをブランクで区切って並べます。
・where=(条件式)
SQL の where 句を記述し、読み取るオブザベーションを条件選択します。
例)
where=(sex="F" or name in ("suzuki","tanaka"))

(良く使いオプション)
・end=変数名
最後のオブザベーションを読んだ時点で値=1 となる一時的な変数を定義します。この変数は出力データセットには出力されません。また point=指定を行った場合は無効です。
・nobs=変数名
SET ステートメントに指定された SAS データセットの合計オブザベーション数を値に持つ一時的な変数うを定義します。

・point=変数名
定義された変数の値をオブザベーション番号として、SAS データセットの任意のオブザベーションを読み取ることができるようになります。このオプションは 1 個の SAS データセットのみ指定する、BY,WHERE ステートメントと一緒に指定できない、DATA ステップの自動ループ実行を終了させるために STOP ステートメントがどこかに必ず必要など、使う上での重要な注意点があります。

(プログラム 2.1-1)　2 つの SAS データセットの縦の連結

```
data japan;
  input ID :$3. name :$10. sex :$1.;
cards;
001 fujita M
002 suzuki F
003 takahashi M
004 tanaka M
;
data us;
  input ID :$3. name :$10. age height weight;
cards;
101 browne 30 160 51
102 gibson 40 160 48
;
data set1;
  set japan us;
run;
options nocenter;
proc print data=set1;run;
```

(ログ)

```
213  data set1;
214    set japan us;
215  run;

NOTE: データセット WORK.JAPAN から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.US から 2 オブザベーションを読み込みました。
NOTE: データセット WORK.SET1 は 6 オブザベーション、 6 変数です。
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	001	fujita	M	.	.	.
2	002	suzuki	F	.	.	.
3	003	takahashi	M	.	.	.
4	004	tanaka	M	.	.	.
5	101	browne		30	160	51
6	102	gibson		40	160	48

SET ステートメントは指定した SAS データセットを逐次的に読み取ります。複数の SAS データセットに同じ変数が定義されていた場合は最初に定義された変数の型と長さに設定されます。いずれかのデータセットにのみ存在する変数では、存在しなかったデータセットからの読み取り値はすべて欠損値にセットされます。

(プログラム 2.1-2)　IN=データセットオプション

```
data set1;
```

```
  set japan(in=in1) us(in=in2);
  if in1=1 then country="JAPAN";
  else if in2=1 then country="UNITED STATES";
run;
options nocenter;
proc print data=set1;run;
```

(ログ)

```
218  data set1;
219    set japan(in=in1) us(in=in2);
220    if in1=1 then country="JAPAN";
221    else if in2=1 then country="UNITED STATES";
222  run;

NOTE: データセット WORK.JAPAN から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.US から 2 オブザベーションを読み込みました。
NOTE: データセット WORK.SET1 は 6 オブザベーション、 7 変数です。
```

(アウトプット)

OBS	ID	name	sex	age	height	weight	country
1	001	fujita	M	.	.	.	JAPAN
2	002	suzuki	F	.	.	.	JAPAN
3	003	takahashi	M	.	.	.	JAPAN
4	004	tanaka	M	.	.	.	JAPAN
5	101	browne		30	160	51	UNITE
6	102	gibson		40	160	48	UNITE

IN 変数は一時的で、データセットには出力されない点に注意してください。
また、DATA ステップでの**変数定義(型と長さ)は DATA ステップの中で最初に定義されたときに設定されます**。ここではまず、SET ステートメントにより SET された SAS データセットの全変数が定義され、続いて IF ステートメントの中で変数 country が出現しており country="JAPAN"という割り当てステートメントにより長さ 5 の文字型として定義されます。したがって、次の ELSE ステートメントの変数 country の割り当て値は 5 文字で切られてしまっています。これを回避するには最初の割り当てにおいて、country="JAPAN　　";　といったように後ろにブランクを付加して全体の長さを指定しておく方法と、LENGTH ステートメントを用いる方法があります。

(プログラム 2.1-3)　LENGTH ステートメント

```
data set1;
  length country $14;
  set japan(in=in1) us(in=in2);
  if in1=1 then country="JAPAN";
  else if in2=1 then country="UNITED STATES";
run;
options nocenter;
proc print data=set1;run;
```

(アウトプット)

OBS	country	ID	name	sex	age	height	weight
1	JAPAN	001	fujita	M	.	.	.
2	JAPAN	002	suzuki	F	.	.	.

3	JAPAN	003	takahashi	M	.	.	.
4	JAPAN	004	tanaka	M	.	.	.
5	UNITED STATES	101	browne		30	160	51
6	UNITED STATES	102	gibson		40	160	48

(LENGTH ステートメント) 定義ステートメント
DATA ステップにおける変数の名前と型を明示的に定義します。定義ステートメントですが、DATA ステップでの位置は重要です。定義したい変数が LENGTH ステートメントより前に、SET ステートメントや INPUT ステートメントや割り当てステートメントなどの変数定義を伴うステートメントで既に定義されていた場合、その変数の LENGTH 指定は無効です。したがって、LENGTH ステートメントは、SAS データセット内の変数の出現順が気にならなければ、常に DATA ステートメントの次に指定すべきです。
(SAS データセットの中の変数は LENGTH ステートメントで指定したものが先に並ぶことになります。)

LENGTH　変数名 型および長さ ... ;

文字型に定義する場合、$を付けます。

(プログラム 2.1-4)　SET ステートメントの END=オプション

```
data set1;
  length country $14;
  set japan(in=in1) us(in=in2) end=end;
  if in1=1 then country="JAPAN";
  else if in2=1 then country="UNITED STATES";
  endflg=end;
run;
options nocenter;
proc print data=set1;run;
```

(アウトプット)

OBS	country	ID	name	sex	age	height	weight	endflg
1	JAPAN	001	fujita	M	.	.	.	0
2	JAPAN	002	suzuki	F	.	.	.	0
3	JAPAN	003	takahashi	M	.	.	.	0
4	JAPAN	004	tanaka	M	.	.	.	0
5	UNITED STATES	101	browne		30	160	51	0
6	UNITED STATES	102	gibson		40	160	48	1

［ランダムアクセスモード：SET ステートメントの POINT=オプション］

(プログラム 2.1-5)　SET ステートメント POINT=オプション

```
data rand1;
  p=2;
  set set1 point=p;
  output;
  p=6;
  set set1 point=p;
  output;
  stop;              /* 必要 */
run;
proc print data=rand1;run;
```

(アウトプット)

OBS	country	ID	name	sex	age	height	weight	endflg
1	JAPAN	002	suzuki	F	.	.	.	0
2	UNITED STATES	102	gibson		40	160	48	1

通常の SAS のデータ読み取りモードは**シーケンシャル読み取り(Sequential Read) モード**です。SET ステートメントに **POINT=**オプションを指定すると、SET ステートメントに指定した SAS データセットはオブザベーション番号に基づく**ランダムアクセス方式で読み取りするモード**に変わります。シーケンシャル読み取りモードの場合は、最後のオブザベーションを読んだ後、DATA ステップで実行する最後の実行ステートメントを実行後に DATA ステップを終了させるべきことが SAS にわかります。ランダムアクセスモードになった場合は、最後のオブザベーションから逆順にオブザベーションを読むといったこともあり得ますので、**自動的に DATA ステップを終了するタイミングが SAS にはわかりません**。そこで、ランダムアクセスする SAS データセットのみを入力する DATA ステップの場合、必ずどこかに **STOP ステートメントを指定**して、明示的に DATA ステップのループ実行を終了させる指定を行う必要があります。

(STOP ステートメント) 実行ステートメント
DATA ステップの実行を終了させます。
DATA ステップは読み取りデータがない場合は 1 回だけ実行、ある場合は読み取るデータが尽きるまで繰り返し実行される(ループ実行する)ルールですが、STOP ステートメントはこの DATA ステップのループ実行を明示的に終了させます。

なお、DATA ステップを完了せず中止させるには ABORT ステートメントを指定します。この場合 DATA ステップはエラーとなり SAS データセットは作成されません。また、ABORT ステートメントに ABEND オプションを付けた ABORT ABEND; を実行すると SAS は終了します。(SAS セッションが終わりますのでここでは決して実行しないでください。)

(プログラム 2.1-6) SET ステートメント POINT=オプションを用いたオブザベーションの逆読み

```
data rand2;
  do p=nobs to 1 by -1;
    set set1 point=p nobs=nobs;
    output;
  end;
  stop;        /* STOPステートメント 忘れないこと */
run;
proc print data=rand2;run;
```

(アウトプット)

OBS	country	ID	name	sex	age	height	weight	endflg
1	UNITED STATES	102	gibson		40	160	48	1
2	UNITED STATES	101	browne		30	160	51	0
3	JAPAN	004	tanaka	M	.	.	.	0
4	JAPAN	003	takahashi	M	.	.	.	0
5	JAPAN	002	suzuki	F	.	.	.	0
6	JAPAN	001	fujita	M	.	.	.	0

［繰り返し DO ループ］
DO インデクス変数名=開始値 TO 終了値 BY 増分値;

(実行ステートメント;)
END;

・「TO 終了値」および「BY 増分値」の部分は指定を省略できます。
・DO ステートメントと END ステートメントに挟まれた部分の実行ステートメントを、変数値に開始値を代入して、必ず 1 回はループ実行します。各ループ実行後、変数値＋増分値<=終了値の条件を満たす場合は DO ループを繰り返し実行します。
・「インデクス変数名＝開始値」の部分は カンマで区切って 繰り返し DO ループ実行時のインデクス変数値を個別に指定することもできます。
例)
do i=1,3,10; i=1,i=3,i=10 の値を与えて DO ループを 3 回繰り返します。
do i=1,3,5 TO 10 BY 2; i=1,3,5,7,9; と書いたのと同じ結果になります。
do i=5,1,4 TO 3 BY -1,2; i=5,1,4,3,2; と同じ。

(プログラム 2.1-7) 繰り返し DO ループの例

```
data rand3;
  do p=5,1,4 TO 3 BY -1,2;
    set set1 point=p nobs=nobs;
    output;
  end;
  stop;        /* STOPステートメント 忘れないこと */
run;
proc print data=rand3;run;
```

(アウトプット)

OBS	country	ID	name	sex	age	height	weight	endflg
1	UNITED STATES	101	browne		30	160	51	0
2	JAPAN	001	fujita	M	.	.	.	0
3	JAPAN	004	tanaka	M	.	.	.	0
4	JAPAN	003	takahashi	M	.	.	.	0
5	JAPAN	002	suzuki	F	.	.	.	0

［データの横の連結(1) 複数の SET ステートメントの指定］

(プログラム 2.1-8) 複数の SET ステートメントの指定

```
options nodate nonumber nocneter;
title "Japan";proc print data=japan;run;
title "US";proc print data=us;run;
data set2;
  set japan;
  set us;
run;
title "SET2";proc print data=set2;run;
```

(ログ)

```
NOTE: データセット WORK.JAPAN から 3 オブザベーションを読み込みました。
NOTE: データセット WORK.US から 2 オブザベーションを読み込みました。
NOTE: データセット WORK.SET2 は 2 オブザベーション、 6 変数です。
```

(アウトプット)

```
Japan
```

OBS	ID	name	sex
1	001	fujita	M
2	002	suzuki	F
3	003	takahashi	M
4	004	tanaka	M

US

OBS	ID	name	age	height	weight
1	101	browne	30	160	51
2	102	gibson	40	160	48

SET2

OBS	ID	name	sex	age	height	weight
1	101	browne	M	30	160	51
2	102	gibson	F	40	160	48

データセットの横の連結で最も簡単な方法は、複数の SET ステートメントを用いることです。
この例では、同じ項目を持つ 2 つの SAS データセット japan と us を横につなげる場合です。

DATA ステップのプログラムデータベクトルを考えると、最初の SET japan;ステートメントの実行時に 3 個の変数(ID,name,sex)の変数が定義され値がセットされます。そして次の SET us; ステートメントで既に定義されている 2 個の変数(ID,name)は変数値を上書きします。そして 3 つの変数 age,height,weight は新たに定義され値がセットされます。変数 sex は直前のセットされた値がそのまま残ります。このようなプロセスで 1 番目と 2 番目のオブザベーションはいずれのデータセットにも存在しますので、DATA ステップのループ実行が行われ、出力されています。

ログを見ると、japan データセットは 3 オブザベーション読込み、us データセットは 2 件読込んだとあります。 これは、DATA ステップの 3 回目のループを開始し、JAPAN データセットから 3 件目のオブザベーションを SET することろは行われたことを示しています。ところが次の SET us; ステートメントでオブザベーションが無いことが分かり、この時点で DATA ステップの 3 回目のループは中止されたことを表しています。

結果的に**複数の SET ステートメントによるデータセットの横の連結はオブザベーション数が少ない方の件数**分だけ処理され出力されるということに注意。

[SET ステートメントと BY グループ処理]

SET ステートメントと BY ステートメントを同時に指定した場合、入力 SAS データセットのオブザベーションの読み取りは通常と異なり、BY 変数値の順に行われます。これを **BY グループ処理**と呼びます。同じ DATA ステップの中で入力されるすべての SAS データセットは、**事前に同じ BY ステートメントを指定した PROC SORT を実行済み**でなければなりません。

(プログラム 2.1-9)　SET ステートメントと BY ステートメント

```
proc sort data=japan out=japan2;by name;run;  /* 事前にSORTしておく */
proc sort data=us out=us2;by name;run;        /* 事前にSORTしておく */
data set_without_by;
  set japan2 us2;
run;
title "Single SET Statement Without By name;";
proc print data=set_without_by;run;
data set_with_by;
  set japan2 us2;
  by name;                                    /* BY ステートメント */
run;
title "Single SET Statement With By name;";
proc print data=set_with_by;run;
```

(アウトプット)

```
Single SET Statement Without By name;

OBS    ID     name         sex    age    height    weight

 1     001    fujita        M       .        .         .
 2     002    suzuki        F       .        .         .
 3     003    takahashi     M       .        .         .
 4     004    tanaka        M       .        .         .
 5     101    browne                30      160        51
 6     102    gibson                40      160        48

Single SET Statement With By name;

OBS    ID     name         sex    age    height    weight

 1     101    browne                30      160        51
 2     001    fujita        M       .        .         .
 3     102    gibson                40      160        48
 4     002    suzuki        F       .        .         .
 5     003    takahashi     M       .        .         .
 6     004    tanaka        M       .        .         .
```

BY ステートメントを指定しない場合は japan2、us2 データセットのオブザベーションを単に順番に読み込んだ結果になっていますが、**BY ステートメントを指定すると、その DATA ステップで入力指定した SAS データセットをすべて同時にオープンし、BY 変数値の順にオブザベーションを読みこむモード**に変わります。なお、BY ステートメントがあると複数の SET ステートメントを指定した場合でも同じモードに入ってしまい、1 つの SET ステートメントに複数の SAS データセットを指定した場合と同じ結果が得られます。

［データの横の連結（２）MERGE ステートメント］

BY ステートメント無しの場合に複数の SET ステートメントの指定を行うとオブザベーション数の少ない方のオブザベーション数の回数しか DATA ステップのループが実行されないため出力されるオブザベーション数は数が少ない方のオブザベーション数になります。これに対して、MERGE ステートメントに複数の SAS データセットを指定すると、オブザベーションの多い方に合わせて横の連結を行います。

(MERGE ステートメント)　実行ステートメント

MERGE データセット名 1(データセットオプション)　データセット名 2(データセットオプション) ...　オプション;

・データセット名は最低 2 個指定します。(ただし 1 個でもエラーにはなりません。この場合 SET と同じ意味になります。)
・データセットオプションは IN=変数など SET と同じものが指定できます。
・オプションは END=変数のみ指定可能です。

(プログラム 2.1-10)　MERGE ステートメント

```
data merge1;
  merge japan us;
run;
title;
proc print data=merge1;run;
```

(ログ)

```
NOTE: データセット WORK.JAPAN から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.US から 2 オブザベーションを読み込みました。
NOTE: データセット WORK.MERGE1 は 4 オブザベーション、 6 変数です。
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	101	Browne	M	30	160	51
2	102	Gibson	F	40	160	48
3	003	takahashi	M	.	.	.
4	004	tanaka	M	.	.	.

MERGE ステートメントは指定した SAS データセットを同時にオープンし、同じオブザベーション番号同士のオブザベーションを横に連結します。(プログラム 2.1-8)と比較すると、ログでは JAPAN データセットを 4 件、US データセットを 2 件読んでおり、いずれも全件読んでいることがわかります。アウトプットを比較すると、OBS1 と OBS2 は全変数の値が同じになっています。そして OBS3 と OBS4 が出力されており、いずれも JAPAN データセットからのオブザベーション値が入っています。

［マッチマージ：　MERGE ステートメントと BY グループ処理］

BY ステートメントを MERGE ステートメントと一緒に用いると、同じオブザベーション番号ではなく、BY 変数値の値が同じオブザベーションを横に揃えて連結しようとするモードに変わります。
一方の SAS データセットの BY 変数値がユニークな顧客番号とその顧客属性情報を格納したデータセット、もう一方は顧客番号と取引情報を格納したデータセットのようなとき、取引情報に顧客番号で紐づけられた顧客属性項目を付加したい場合などに大変良く使われる指定です。

(プログラム 2.1-11)　マッチマージの例

```
title "USデータセット";
proc print data=us;run;
data trans;
  input ID :$3. yyyymmdd :$8. itemno :$3. sales;
cards;
101 20101001 A01 256
102 20101003 B02 600
103 20101004 C01 123
```

```
102 20101005 B03 850
101 20101006 A03  45
;
proc sort data=us;by ID;run;
proc sort data=trans;by ID;run;
data match_merged;
  merge us trans;
  by ID;
run;
title "マッチマージの結果";
proc print data=match_merged;run;
```

(ログ)

```
NOTE: データセット WORK.US から 2 オブザベーションを読み込みました。
NOTE: データセット WORK.TRANS から 5 オブザベーションを読み込みました。
NOTE: データセット WORK.MATCH_MERGED は 5 オブザベーション、 8 変数です。
```

(アウトプット)

USデータセット

OBS	ID	name	age	height	weight
1	101	browne	30	160	51
2	102	gibson	40	160	48

マッチマージの結果

OBS	ID	name	age	height	weight	yyyymmdd	itemno	sales
1	101	browne	30	160	51	20101001	A01	256
2	101	browne	30	160	51	20101006	A03	45
3	102	gibson	40	160	48	20101003	B02	600
4	102	gibson	40	160	48	20101005	B03	850
5	103		.	.	.	20101004	C01	123

［ｎ対ｍのマージ］

マッチマージが片方のデータセットの BY 変数値がユニーク、もう片方はユニークで無い場合（1 対ｎと呼ぶ）の同じ BY 変数値のオブザベーションの横の連結を意味し、結果は明らかになるのに対して、両方のデータセットの BY 変数値がユニークで無い場合（ｎ対ｍと呼ぶ）はどのようになるでしょう？

(プログラム 2.1-12)　n 対 m のマージの例

```
data cardinfo;
  input ID :$3. cardnum :$9.;
cards;
101 CARD101_1
102 CARD102_1
102 CARD102_2
102 CARD102_3
;
data trans;
  input ID :$3. yyyymmdd :$8. itemno :$3. sales;
cards;
101 20101001 A01 256
101 20101006 A03  45
```

```
102 20101003 B02 600
102 20101005 B03 850
;
data merge_n_m;
  merge cardinfo trans;
  by ID;
run;
title "n対mのマージ";
proc print data=merge_n_m;run;
```

(ログ)

```
NOTE: MERGE ステートメントに BY 値を繰り返すデータセットが複数あります。
NOTE: データセット WORK.CARDINFO から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.TRANS から 4 オブザベーションを読み込みました。
NOTE: データセット WORK.MERGE_N_M は 5 オブザベーション、 5 変数です。
```

ログに「**MERGE ステートメントに BY 値を繰り返すデータセットが複数あります**」というメッセージが出ることに注目。

(アウトプット)

n対mのマージ

OBS	ID	cardnum	yyyymmdd	itemno	sales
1	101	CARD101_1	20101001	A01	256
2	101	CARD101_1	20101006	A03	45
3	102	CARD102_1	20101003	B02	600
4	102	CARD102_2	20101005	B03	850
5	102	CARD102_3	20101005	B03	850

OBS=5 のオブザベーションに注目。TRANS データセットには ID="102"は 2 件しかありませんが、CARDINFO データセットには ID="102"は 3 件あります。これが n 対 m のマージになっています。多い方の 3 件がアウトプットされますが、3 件目のオブザベーションの TRANS データセットからの変数 yyyymmdd,itemno,sales の値は 2 件目の値がコピーされています。SAS では MERGE ステートメントに BY 値を指定した場合、このような特別な**「値保持モード」**になるのがデフォルトです。

［n 対mのマージでデータ値が存在しない場合は欠損値にセットする方法］

上記の結果は一見不自然（理不尽）にも思えますが、1 対 n のマッチマージを考えると、「オブザベーション数が足りない部分のマージ後の変数値は、マージできた最後のオブザベーションの変数値をそのままコピーする」、という仕方によってマッチマージの結果が得られているわけで、同じ理屈を n 対 m でも実行しているのです。

足りない部分のオブザベーションの変数値を欠損にするには、以下のように IN=データセットオプションを用いて、MERGE ステートメント実行の前に IN 変数値=0 に設定します。

(プログラム 2.1-13)　n 対 m のマージの例で足りないオブザベーション側の変数値を制御する

```
data merge_n_m;
  trans=0;                          /* 制御したいデータセットのIN変数をリセット */
  merge cardinfo trans(in=trans); /* 制御したい変数の入っているデータセットにIN変数を定義 */
  by ID;
```

```
  if trans=0 then do;              /* IN変数が0、すなはち読み取りデータが無い場合のDO処理を開始
*/
    yyyymmdd=.;itemno="";          /* 変数値をセット */
  end;                             /* DO処理の終了 */
run;
title "n対mのマージ";
proc print data=merge_n_m;run;
```

(アウトプット)

n対mのマージ

OBS	ID	cardnum	yyyymmdd	itemno	sales
1	101	CARD101_1	20101001	A01	256
2	101	CARD101_1	20101006	A03	45
3	102	CARD102_1	20101003	B02	600
4	102	CARD102_2	20101005	B03	850
5	102	CARD102_3	.		850

DATA ステートメントの次の最初の割り当てステートメント trans=0; は必要です。MERGE ステートメントの特別な「値保持モード」は DATA ステップのループ実行の間ずっと有効になるからです。DATA ステップの各ループ処理ごとにこの値保持モードを常にリセットするために最初に指定する必要があります。なお、両方の入力データセットについてこれを行うときは、IN=データセットオプションを両方につけて、冒頭で同じように割り当てステートメントを書いて、if ～end 処理を同じように書き加えてください。

参考として、merge ステートメントと by ステートメントを用いたデータの横の連結指定において、in=データセットオプションの指定や値保持モードのリセットを指定した場合の結果の差異を[別表 8] に例示しました。

［データセットの値の更新　UPDATE ステートメント］

SAS データセットを更新するには UPDATE ステートメント、もしくは MODIFY ステートメントを使います。MODIFY ステートメントは直接更新したいデータセットのデータを修正してしまうもので、ここでは取り扱いません。UPDATE ステートメントは、ある SAS データセット（Master）に対する修正情報を格納した SAS データセット（Transaction）を用意し、別のデータセットに更新された SAS データセットを作成する方法を実行するものです。

(プログラム 2.1-14)　UPDATE ステートメントによるデータセットの更新

```
data japan;
  input ID :$3. name :$10. sex $:1.;
cards;
001 fujita M
002 suzuki F
003 takahashi M
004 tanaka M
;
data modinfo;
  input ID :$3. name :$10. sex $:1.;
cards;
003 takashi .
004 . F
;
```

```
data japan2;
  update japan modinfo;
  by ID;
run;
title;
proc print data=japan2;run;
```

(アウトプット)

OBS	ID	name	sex
1	001	fujita	M
2	002	suzuki	F
3	003	takashi	M
4	004	tanaka	F

(UPDATE ステートメント)　実行ステートメント

UPDATE マスターデータセット名　トランザクションデータセット名 オプション;

・マスターデータセットは更新前の SAS データセットを指定します。
・BY ステートメントを指定し、マスターデータセットの BY 変数値はユニークでなければなりません。
・トランザクションデータセットの BY 値はユニークでなくてかまいません。更新情報を以下のように入力しておきます。

・更新したい項目はマスターと同じ変数名を持たせます。
・BY 変数値をユニークキーとして更新したいデータ値は更新情報、更新しないデータには欠損値をセットしておきます。
・マスターに存在しない BY 値は新規オブザベーションとして追加されます。

以上でファイルの連結・マージ・更新のテーマは終了です。

[DO ループ処理]

繰り返し DO ループについては既に学びましたので、その他の DO ループ処理について学びます。

(DO WHILE ステートメント)　実行ステートメント

DO WHILE (条件式);
　(実行ステートメント;)
END;

DO WHILE ステートメントは **DO ループの開始前**に条件判断を行い、条件を満たす場合 DO ループを実行します。繰り返し DO ループと DO UNTIL ループが 1 度は必ず DO ループを実行するのに対して、DO WHILE ループは一度も実行されない場合もあり得ます。

(プログラム 2.1-15)

```
data _null_;
  i=10;
```

```
  do while (i>=1);
    put i=;
    i=i-1;
  end;
run;
```

(ログ)

```
1023  data a;
1024    i=10;
1025    do while (i>=1);
1026      put i=;
1027      i=i-1;
1028    end;
1029  run;

i=10
i=9
i=8
i=7
i=6
i=5
i=4
i=3
i=2
i=1
```

(DO UNTIL ステートメント)　実行ステートメント

DO UNTIL (条件式);
　（実行ステートメント;）
END;

DO UNTIL ステートメントは **DO ループの終了後**に条件判断を行い、条件を満たさない場合 DO ループを再度実行します。1 度は必ず DO ループを実行します。

(プログラム 2.1-16)

```
data _null_;
  i=10;
  do until (i>=1);
    put i=;
    i=i-1;
  end;
run;
```

(ログ)

```
1030  data _null_;
1031    i=10;
1032    do until (i>=1);
1033      put i=;
1034      i=i-1;
1035    end;
1036  run;
```

```
i=10
```

［配列処理］

DATA ステップの複数の変数をまとめて配列として配列名で宣言しておけば、同じ処理を複数の変数に対して行いたい場合、変数ごとに処理を書かなくて済むようになります。SAS の配列はそのような目的で使うものと考えてください。

(ARRAY ステートメント)　宣言ステートメント

ARRAY 配列名 {要素数} 型と長さ 変数名リスト;

例)
array x{100} x1-x100;
array var {*} TOKYO OSAKA NAGOYA;

・要素数の部分は要素変数の数を書いてもかまいませんが、通常、省略値 * を書けば十分です。SAS がカウントします。
　要素数の指定部分は、インデクス開始値:終了値 x{0:99}、多次元配列指定 x{4,25}、といった指定も可能です。いずれも要素数は 100 になりますが、配列要素を参照するインデクス番号が異なることになります。例えば、26 番目の要素変数を参照するには、x{100} 配列定義の場合は x{26}、x{0:99}配列定義の場合は x{25}、x{4,25}配列定義の場合は x{2,1}と参照することになります。
・配列名は SAS の名前付けルールに従います。(最大 32 文字)
・文字変数は文字変数同士、数値変数は数値変数同士しか同じ配列名で定義できません。
・既に入力データセットに存在する変数やあらかじめ LENGTH ステートメントなどで定義済みの変数を配列定義する場合は矛盾する型と長さを指定してはいけません。

配列名から各変数の参照方法は、以下の通りです。

配列名{要素番号}

例)
x{5}=100;　　上記 array 定義の場合、x5=100; と書くのと同じ。
total=x{1}+x{2};　　同様に、 total=x1+x2;
sumsales=var{2}+var{3}　　同様に、 sumsales=OSAKA+NAGOYA;

(プログラム 2.1-17)　配列なしのプログラミング

```
data array_nasi;
  set  sales;
  s01=s01*10;
  s02=s02*10;
  s03=s03*10;
  s04=s04*10;
  s05=s05*10;
  tot_sales=tot_sales*10;
  avr_sales=avr_sales*10;
run;
proc print data=array_nasi;run;
```

(ログ)

```
1218  data array_nasi;
1219    set  sales;
1220    s01=s01*10;
1221    s02=s02*10;
1222    s03=s03*10;
1223    s04=s04*10;
1224    s05=s05*10;
1225    tot_sales=tot_sales*10;
1226    avr_sales=avr_sales*10;
1227  run;
NOTE: 欠損値を含んだ計算により、以下の箇所で欠損値が生成されました。
      ( 回数 )( 行 : カラム )
      1 1223:10   1 1224:10
NOTE: データセット WORK.SALES から 3 オブザベーションを読み込みました。
NOTE: データセット WORK.ARRAY_NASI は 3 オブザベーション、 9 変数です。
```

(アウトプット)

OBS	dept	s01	s02	s03	s04	s05	tot_ sales	avr_ sales	active_ year
1	A	1250	1000	1810	2010	2250	8320	1664	5
2	B	800	1150	3000	2620	1930	9500	1900	5
3	C	100	110	120	.	.	330	110	3

ログに欠損値との演算結果が**欠損値になった箇所と DATA ステップループ回数が表示**されている点に注意。

(プログラム 2.1-18)　配列を用いたプログラミング

```
data array_ari;
  set  sales;
  array s {*} s01-s05 tot_sales avr_sales;
  do i=1 to dim(s);
    s{i}=s{i}*10;
  end;
run;
proc print data=array_ari;run;
```

(アウトプット)

OBS	dept	s01	s02	s03	s04	s05	tot_ sales	avr_ sales	active_ year	i
1	A	1250	1000	1810	2010	2250	8320	1664	5	8
2	B	800	1150	3000	2620	1930	9500	1900	5	8
3	C	100	110	120	.	.	330	110	3	8

do i=1 to dim(s);　の dim(s) は配列の要素数を返す関数です。この DO ループのインデクス変数 i も drop しないと出力データセットに含まれてしまう点に注意。

[RETAIN ステートメントと合計ステートメント]

ここで、**DATA ステップでオブザベーションの縦方向の集計などを行う場合**に用いなければならなくなる RETAIN ステートメントと合計ステートメントについて学習しておきます。

(RETAIN ステートメント)　宣言ステートメント

DATA ステップのループ実行時は、MERGE ステートメントの BY 処理が行われるときのような特別な場合以外、基本的にデータベクトルの値はリセットされ、欠損値がセットされます。それでは DATA ステップで複数のオブザベーションにわたる計算を行うことができません。そこで、DATA ステップの新しいループを実行しても前の DATA ステップループ時の最後の値を保持するために、RETAIN ステートメントが用意されています。指定は非常に簡単で、

RETAIN 変数名 初期値;

です。初期値は指定しなくてもかまいません。その場合は最初の DATA ステップのループ開始時は欠損値にセットされます。なお、変数名も省略し、RETAIN;と書くとその DATA ステップに登場する全変数が RETAIN されます。

例)
retain x y 1 z; x と y は初期値 1,z は欠損値にセット
retain a "ABC" b; a は"ABC",b は""(" ")に初期値セット

(プログラム 2.1-19) RETAIN ステートメント

```
data retain1;
  array sum {*} 8 sum1-sum5;
  retain sum1-sum5 0;
  set sales end=end;
  array s {*} s01-s05;
  do i=1 to 5;
    sum{i}=sum{i}+s{i};
  end;
run;
proc print data=retain1;run;
```

(アウトプット)

OBS	sum1	sum2	sum3	sum4	sum5	dept	s01	s02	s03	s04	s05	tot_ sales	avr_ sales	active_ year	i
1	125	100	181	201	225	A	125	100	181	201	225	832	166.4	5	6
2	205	215	481	463	418	B	80	115	300	262	193	950	190.0	5	6
3	215	226	493	.	.	C	10	11	12	.	.	33	11.0	3	

★RETAIN ステートメントをコメントにして実行してみてください。

欠損値との足し算で結果が sum4、sum5 は欠損値となってしまいましたが、欠損を足しても欠損をゼロとみなして足しこみ計算（累積）を行いたい場合があります。そこで、合計ステートメントが登場します。

(合計ステートメント) 実行ステートメント

変数名+式;

一寸変わっていますが、+がキーワードです。変数名に該当する変数が合計値を取る変数名として認識されます。

例)

a+1;　　DATA ステップのループ実行ごとに 変数 a に 1 を足した値を新たな a の値にします。
a+b;
sum+x{i};

a+1; が a=a+1; と異なるのは a が RETAIN されることです。
a+b; が a=a+b; と異なるのは a が RETAIN されることと、b の値が欠損であっても 0 とみなして足しこみを行うことです。

・合計ステートメントに指定した変数は RETAIN ステートメントに指定しなくても RETAIN されます。
・定数を差し引きしたい場合は＋はキーワードなので、 +(-1) のように指定します。

(プログラム 2.1-20)　合計ステートメント

```
data retain2;
  array sum {*} 8 sum1-sum5;
  set sales end=end;
  array s {*} s01-s05;
  do i=1 to 5;
    sum{i}+s{i};
  end;
run;
proc print data=retain2;run;
```

(アウトプット)

OBS	sum1	sum2	sum3	sum4	sum5	dept	s01	s02	s03	s04	s05	tot_ sales	avr_ sales	active_ year	i
1	125	100	181	201	225	A	125	100	181	201	225	832	166.4	5	6
2	205	215	481	463	418	B	80	115	300	262	193	950	190.0	5	6
3	215	226	493	463	418	C	10	11	12	.	.	33	11.0	3	

2.2 日付処理・関数・フォーマットの利用と DATA ステップを用いたレポーティング ～ 日付フォーマット、日付インフォーマット、日付関数、FORMAT プロシジャ、BY グループ処理、PUT ステートメント

ここでは SAS における時間の取り扱いと DATA ステップを用いたレポーティングについて学習します。SAS では日時の取り扱いは **1960 年 1 月 1 日を起点とする経過日数**を値とする「**SAS 日付値**」、**1960 年 1 月 1 日午前 0 時 0 分 0 秒を起点とする経過秒数**を値とする「**SAS 日時値**」、そして、**午前 0 時 0 分 0 秒を起点とする経過秒数**を値とする「**SAS 時間値**」という 3 通りの基本的時間概念があります。SAS では時間タイプといった特別なタイプの変数はありません。すべて通常の数値タイプの変数です。変数値が 1960 年 1 月 1 日からの経過日数を表すものとみなせば SAS 日付値になり、経過秒数を表すものとみなせば SAS 日時値になるという仕組みです。

外部データ上の日付や時間を表すさまざまな表現形式のデータを SAS 日付値として読み込むには INPUT ステートメントのフォーマット入力に**日付や時間に関するインフォーマット**を用います。逆に SAS 日付値や SAS 日時値を年月日表示などの表現形式で外部データやアウトプットとして書き出すには**フォーマット**を用います。
SAS 日付値を通常の年月日表現の文字値に変換する、逆に年月日の表現形式で記述された文字変数値から SAS 日付値に変換する **PUT 関数**や **INPUT 関数**があります、また顧客の誕生日から現在の年齢を計算したり、最初に顧客になってからの経過日数を計算することができる **INTCK 関数**、任意の時間経過後の SAS 日付値を算出する **INTNX 関数**などがあります。

時間に関する学習の後は、DATA ステップで集計レポートを作成するプログラミングを学習します。一般的な営業所別部門別販売取引データセットを入力し、営業所別、部門別売上合計を計算して集計表を作成するようなことが行われます。部門別集計を行うとき活躍するのは BY グループ処理です。BY グループ処理については SET,MERGE,UPDATE ステートメントにおける BY グループ処理が登場しましたが、ここでは**コントロールブレイク**と呼ばれる **BY 変数値の変化のタイミングをとらえる FIRST.BY 変数, LAST.BY 変数** という特殊な機能について理解します。また、DATA ステップを用いたレポーティングには PUT ステートメントが活躍します。

［日付値、日時値、時間値］

まず、それぞれ定数で与える方法と、与えらた値の内部値(SAS 変数値)を確認します。

(プログラム 2.2-1)　SAS 日付値、日時値、時間値

```
data _null_;
  date1="01JAN1960"D;
  date2="31DEC60"D;
  date3="17DEC10"D;
  put date1= date2= date3=;

  datetime1="01JAN1960:00:00:00"DT;
  datetime2="31DEC60:23:59:59"DT;
  put datetime1= datetime2=;

  time1="00:00:00"T;
  time2="24:00:00"T;
  time3="39:15:30.9"T;
```

```
  put time1= time2= time3=;

  date4="31MAR20"D;
  put date4=;
run;
```

(ログ)

```
date1=0 date2=365 date3=18613
datetime1=0 datetime2=31622399
time1=0 time2=86400 time3=141330.9
date4=-14520
```

日付値は""D, 日時値は ""DT, 時間値は""T という特別な表現で定数を与えます。
""の中は、日付は ddmmmyy または ddmmmyyyy ただし mmm の部分は英語の月名を表す 3 文字が入ります。時間部分は hh:mm:ss.s の形式で与えます。 日時値の場合、日付と時間部分の区切りにも":"を付けます。

"ddmmmyy" は DATE7 フォーマット、 "ddmmmyyyy" は DATE9 フォーマットと呼ばれます。DATE7 フォーマットのときは、西暦が下 2 桁で表現されていますので、19xx なのか 20xx のいずれを表しているのかが問題になります。(2000 年問題) SAS ではこの問題に対して、**yearcutoff=というオプション**で対応しており、現在のオプション設定値は次の指定を実行するとログに表示されます。

(プログラム 2.2-2) SAS 設定オプション YEARCUTOFF=の確認

```
proc options;run;
```

(ログ)

```
YEARCUTOFF=1920      SAS 日付処理の 100 年単位の基準年を指定します
```

これは **2 桁の西暦年表示は 1920 年から 2019 年の間にあるものとみなす**という意味です。したがって、2020 年を表すつもりで date="31MAR20"D; と書いても今の YEARCUTOFF 設定では SAS は 1920 年 3 月 31 日と認識します。そのため上記の例での SAS 変数値 date4 の値は起点の 1960 年から 40 年前の-14520 という値になります。
なお 1960 年は閏年なので 366 日あったことに注意。

[日付フォーマット]

SAS 日付値を経過日数で表示しても何のことかわかりません。これを年月日表示してみましょう。SAS 変数値を特定の編集形式で書き出すにはフォーマットを使います。[別表 6]にある SAS 日付値、日時値、時間値ごとに使えるフォーマットの一覧を掲載してありますので、参考にしてください。

(プログラム 2.2-3) FORMAT の指定

```
data _null_;
  date="17DEC10"D;
  put date yymmdd. +1 date yymmddn8. +1 date yymmdds10. +1 date yymmdd4.;

  datetime="31DEC60:23:59:59"DT;
  put datetime datetime. +1 datetime dtdate.;

  time="39:15:30.9"T;
  put time time. time timeampm.;
run;
```

(ログ)

```
10-12-17 20101217 2010/12/17 1012
31DEC60:23:59:59 31DEC60
39:15:31 3:15:31 PM
```

[PUT 関数]

PUT 関数は、PUT(変数名,フォーマット)　という文法で、1 番目の引数の変数の値を 2 番目の引数に指定したフォーマットで編集した値を返します。

SAS 日付値を日付フォーマットで書き出した値に変換することが PUT 関数で可能です。なお、PUT 関数の結果は必ず文字タイプになります。

例)

x=1234567;

date=put(x,comma12.);

date="1,234,567" になります。

(プログラム 2.2-4)　PUT 関数

```
data _null_;
  date="17DEC10"D;
  date2=put(date,yymmdds10.);

  datetime="31DEC60:23:59:59"DT;
  datetime2=put(datetime,datetime.);

  time="39:15:30.9"T;
  time2=put(time,timeampm.);

  put date2 / datetime2 / time2;
run;
```

(ログ)

```
2010/12/17
31DEC60:23:59:59
3:15:31 PM
```

[インフォーマットと INPUT 関数]

フォーマットに対してインフォーマットがあり、PUT 関数に対して INPUT 関数があります。

インフォーマットは外部データの編集形式に合わせて SAS 変数に読み取るときに使います。

(プログラム 2.2-5)　INFORMAT を使い日付を表すデータを SAS 日付値として読み込む

```
data _null_;
  input date :yymmdd8.;
  put date "→" +1 date date9.;
cards;
19600101
20101018
;
```

(ログ)

```
0 → 01JAN1960
18553 → 18OCT2010
```

INPUT 関数は、INPUT(変数名,インフォーマット) という文法で、 1 番目の引数の変数の値を 2 番目の引数に指定したインフォーマットで読み込んだときの SAS 変数値の値を返します。この関数の結果はインフォーマットが文字型なら文字タイプ、数値型なら数値タイプになります。これは INPUT ステートメントでインフォーマットを指定した場合と同じです。

(プログラム 2.2-6)　INPUT 関数

```
data _null_;
  input date $ 1-8;
  sasdate_value=input(date,yymmdd8.);
  put date "→" +1 sasdate_value;
cards;
19600101
20101018
;
```

(ログ)

```
19600101 → 0
20101018 → 18553
```

[経過時間の計算など]

INTCK 関数を使えば、2 つの SAS 日付値、日時値、時間値の経過期間を計算することができます。ただし、年齢計算には向きません。日付値などから年、月、日などを取り出す関数などについてもここで学習します。

(プログラム 2.2-7)　INTCK 関数

```
data sample;
  input ID name :$10. birth :yymmdd8.;
cards;
001 fujita 19580409
002 suzuki 19850131
003 takahashi 19921201
004 tanaka 20091231
;
data intck;
  set sample;
  today=today();
  keikayear=intck("year",birth,today);
  keikamonth=intck("month",birth,today);
  keikaday=intck("day",birth,today);
  keikaday2=today-birth;
run;
proc print data=intck;
  format birth today yymmdds10.;
run;
```

(アウトプット)

```
OBS   ID   name            birth        today    keikayear   keikamonth   keikaday   keikaday2
```

1	1	fujita	1958/04/09	2010/10/17	52	630	19184	19184
2	2	suzuki	1985/01/31	2010/10/17	25	309	9390	9390
3	3	takahashi	1992/12/01	2010/10/17	18	214	6529	6529
4	4	tanaka	2009/12/31	2010/10/17	1	10	290	290

(INTCK 関数の指定方法)

INTCK("時間単位",開始値,終了値)

"時間単位"には year, qtr, month, week, day, hour, minute, second が指定できます。開始値、終了値は SAS 日付値などとみなされます。同じ仲間の関数に今から 3 年後の日付を計算するといった用途に用いる INTNX 関数があります。

INTNX("時間単位",開始値,増分)
例)
3month_after=intnx("month",today(),3);

INTCK 関数を使った年数計算は、開始値、終了値ともにその時間単位のスタート時点（時間単位が"年"なら開始値、終了値とも月日部分は 1 月 1 日）とみなして計算します。したがって 12 月 31 日の誕生日と翌日の 10 月 17 日を INTCK 関数で経過年数を計算すると 1 が返ってくるわけです。

［年齢計算］

というわけで、INTCK 関数は使わずに、年齢計算を行います。

(プログラム 2.2-8)　年齢計算

```
data nenrei;
  set sample;
  /* 普通の年齢計算方法 */
  today=today();
  by=year(birth); bm=month(birth); bd=day(birth);  /* 年、月、日とそれぞれ取り出す */
  ty=year(today); tm=month(today); td=day(today);  /* 年、月、日とそれぞれ取り出す */
  nenrei1=ty-by-(bm*100+bd>tm*100+td);   /* 年の差をとる。誕生日が基準日で未到来（大きい）なら
１を引く */

  /* もう1つの簡単な方法  */
  nenrei2=int(input(put(today,yymmddn8.),8.4)-input(put(birth,yymmddn8.),8.4));
run;
proc print data=nenrei;
  format birth today yymmdds10.;
run;
```

(アウトプット)

OBS	ID	name	birth	today	by	bm	bd	ty	tm	td	nenrei1	nenrei2
1	1	fujita	1958/04/09	2010/10/17	1958	4	9	2010	10	17	52	52
2	2	suzuki	1985/01/31	2010/10/17	1985	1	31	2010	10	17	25	25
3	3	takahashi	1992/12/01	2010/10/17	1992	12	1	2010	10	17	17	17
4	4	tanaka	2009/12/31	2010/10/17	2009	12	31	2010	10	17	0	0

［数値－文字変換］

PUT 関数を使えば、数値変数値を別の文字タイプの変数に持たせることができます。

(プログラム 2.2-9)　数値－文字変換

```
options nocenter;
data a;
  input x;          /* 数値タイプとして読む */
  length c1 $8;     /* 文字タイプに定義 */
  c1=x;             /* 自動変換 */
  c2=put(x,5.);     /* フォーマットで文字変数の長さを指定します */
  c3=put(x,6.2);    /* フォーマットで文字変数の長さを指定します */
  c4=put(x,best12.);
cards;
12345
123.45
1.23456e10
;
proc print data=a;
proc contents data=a;run;
```

(ログ)

```
490  options nocenter;
491  data a;
492    input x;          /* 数値タイプとして読む */
493    length c1 $8;     /* 文字タイプに定義 */
494    c1=x;             /* 自動変換 */
495    c2=put(x,5.);     /* フォーマットで文字変数の長さを指定します */
496    c3=put(x,6.2);    /* フォーマットで文字変数の長さを指定します */
497    c4=put(x,best12.);
498  cards;

NOTE: 以下の箇所で数値を文字値に変換しました。
      ( 行 : カラム )
      494:6
NOTE: データセット WORK.A は 3 オブザベーション、 5 変数です。
NOTE: 数値をプリントするには小さすぎる W.D 出力形式がありました。 "BEST"
      出力形式によって小数点がシフトされる場合があります。
```

(アウトプット)

```
OBS                 x        c1        c2        c3          c4

 1           12345.00     12345     12345     12345       12345
 2             123.45    123.45       123    123.45      123.45
 3     12345600000.00  1.235E10       12E9      123E8  12345600000

   変数と属性の昇順リスト

#    変数     タイプ     長さ

2    c1       文字          8
3    c2       文字          5
4    c3       文字          6
5    c4       文字         12
1    x        数値          8
```

文字タイプ定義された変数に数値タイプ変数を割り当てると、文字変数に**自動変換**されることに注意。

また、文字フォーマット w. で小数点以下の桁数を指定する w.d の d は指定しても無視されます。

［文字ー数値変換］

逆に、PUT 関数と INPUT 関数を使えば、数字が入っている文字変数の値を別の数値変数に持たせることができます。

(プログラム 2.2-10)　文字ー数値変換

```
options nocenter;
data a;
  input c :$10.;    /* 文字タイプとして読む */
  length x1 8;      /* 数値タイプに定義 */
  x1=c;             /* 自動変換 */
  x2=input(c,5.);     /* フォーマットで文字変数の長さを指定します */
  x3=input(c,6.2);    /* フォーマットで文字変数の長さを指定します */
  x4=input(c,best12.);
cards;
12345
123.45
1.23456e10
;
proc print data=a;
proc contents data=a;run;
```

(ログ)

```
505  options nocenter;
506  data a;
507    input c :$10.;    /* 文字タイプとして読む */
508    length x1 8;      /* 数値タイプに定義 */
509    x1=c;             /* 自動変換 */
510    x2=input(c,5.);     /* フォーマットで文字変数の長さを指定します */
511    x3=input(c,6.2);    /* フォーマットで文字変数の長さを指定します */
512    x4=input(c,best12.);
513  cards;

NOTE: 以下の箇所で文字値を数値に変換しました。
      ( 行 : カラム )
      509:6
NOTE: データセット WORK.A は 3 オブザベーション、 5 変数です。
```

(アウトプット)

```
OBS     c                       x1          x2        x3                 x4

 1      12345                12345.00    12345.00   123.450          12345.00
 2      123.45                 123.45      123.40   123.450            123.45
 3      1.23456e10      12345600000.00        1.23     1.235     12345600000.00

   変数と属性の昇順リスト

#     変数     タイプ     長さ

1     c        文字         10
2     x1       数値          8
3     x2       数値          8
4     x3       数値          8
5     x4       数値          8
```

数値タイプのインフォーマットの w.d の指定は、読み取る元のデータに小数点が無ければ小数点以下 d 桁の数値として読み込むという意味があります。データに小数点があれば、データの値が優先されます。

[$CHARw. フォーマット、インフォーマット]

文字変数値は普通の$w.インフォーマットで読むと、頭のブランクは左詰めされた形で読み込まれます。$CHARw.インフォーマットはブランクを左詰めせずにそのままの形で読み込むインフォーマットです。

(プログラム 2.2-11)　先頭のブランクを左詰めせずにデータ値を読み取る

```
data a;
  input @1 name $char10.;
cards;
斎藤
 斎藤
;
proc print data=a;run;
```

(アウトプット)

```
OBS     name

 1      斎藤
 2       斎藤
```

[FORMAT プロシジャ]

FORMAT プロシジャはユーザ独自のフォーマットを定義するプロシジャです。データ値は短いコード値で入力しておき、計算もコード値で集計しておき、集計結果をレポートするときにコード値を日本語の説明フォーマットで表示する目的などに非常に良く使われる機能です。

PROC FORMAT オプション;
VALUE ステートメント;
PICTURE ステートメント;
INVALUE ステートメント
RUN;

(主なオプション)
・LIBRARY=SAS ライブラリ参照名.カタログ名
作成するフォーマット定義カタログの保存先を指定します。
・CNTLOUT=SAS データセット名
フォーマット定義を生成するための決まったフォーマット定義用変数を含む出力 SAS データセット名を指定します。
・CNTLIN=SAS データセット名
決まったフォーマット定義用変数を含む入力 SAS データセットを指定しフォーマットカタログを生成します。通常の VALUE ステートメントに記載したものをコンパイルしてフォーマットを生成するよりも、はるかに高速にフォーマットを生成できますので良く使われます。
・FMTLIB
フォーマットカタログ情報をプリントします。内容が豊富な場合、非常にたくさんの出力が出る場合があ

りますので注意が必要です。

(良く使うステートメント)

(VALUE ステートメント)
個々の値に対するフォーマットを定義します。

VALUE フォーマット名 フォーマット定義;

(フォーマット名の規則)
フォーマット名は文字変数値のフォーマットを定義する場合は先頭は$で始め、2 文字目以降最後から 2 番目までは SAS 名前付けルールに従って名前を付けます。数値の場合は最後から 2 番目までは SAS 名のルールを適用して名前を付けます。いずれの場合も、最後の文字は数字は使用できません。長さは$を含めて 32 文字以内です。

(フォーマット定義方法)
数値フォーマットの場合は、値="フォーマット値"
文字フォーマットの場合は　"値"="フォーマット値"
の形式で個々の値に付けたいフォーマット値を定義します。
特別な定義として以下があります。
other=""
という指定を 1 つの VALUE ステートメントに 1 つだけ指定できます。これは、それまでに定義した値以外のすべての値に適用されるフォーマットを定義することを意味します。

例)
VALUE $SEX
"F"="女性"
"M"="男性"
other="記入なし"
;
数値フォーマットの場合は　値の部分を範囲定義できます。

VALUE AGE
low-20="若年"
20<-<40="中年"
40-high="老年"
;
-は前後に=があるものと解釈してください。
<-は左側の値は含まないという意味です。同様に -< は右側の値を含まないという意味。
low はマイナス∞とみなして結構ですが、欠損値は含まれません。
high は∞を表します。

(プログラム 2.2-12)　ユーザ定義フォーマットの作成

```
data sample;
  input ID name :$10. sex :$1. age height weight;
```

```
cards;
001 fujita M 30 175 70
002 suzuki F 29 168 59
003 takahashi M 32 180 85
004 tanaka M 40 178 77
;

proc format;
  value $sex
    "M"="男性"
    "F"="女性"
    other="不明"
    ;
  value age
    low-30="若年"
    31-40="中年"
    41-high="老年"
    other="不明"
    ;
run;
proc print data=sample;
  format sex $sex. age age.;
run;
```

(ログ)

```
1062  proc format;
1063    value $sex
1064      "M"="男性"
1065      "F"="女性"
1066      other="不明"
1067      ;
NOTE: 出力形式 $SEX を作成しました。
1068    value age
1069      low-30="若年"
1070      31-40="中年"
1071      41-high="老年"
1072      other="不明"
1073      ;
NOTE: Format AGE は既にライブラリに存在します。
NOTE: 出力形式 AGE を作成しました。
1074  run;
```

(アウトプット)

OBS	ID	name	sex	age	height	weight
1	1	fujita	男性	若年	175	70
2	2	suzuki	女性	若年	168	59
3	3	takahashi	男性	中年	180	85
4	4	tanaka	男性	中年	178	77

[コード表データからユーザ定義フォーマットを作成する例]

一般に Excel その他のテキストファイルに分析に用いる文字型データ項目のコードと対応する説明テキストが保管されている場合が多いと思います。情報が多い場合、VALUE ステートメントにコードとフォー

マットを転記しながら打ち込むと、手間もかかりますし、間違いも多くなります。また、コード情報が何千ともなると、これを書きこんだ VALUE ステートメントはコンパイルに長い時間がかかることになります。
このような場合は、**決まった変数項目を持つ CNTL 形式の SAS データセットにコード表情報を持たせておいて、FORMAT プロシジャの CNTLIN=指定を行う**と大変便利で、しかも高速にフォーマットカタログを生成してくれます。

(プログラム 2.2-13)　CNTL 形式 SAS データセットの作成と呼びこみ

```
data cntl1;
  input fmtname :$8. start :$32. label :$60.;
  type="C";
  end=start;
  hlo="";
cards;
seibetu M 男
seibetu F 女
seibetu 99999 不明
name fujita 藤田
name suzuki 鈴木
;
proc print data=cntl1;run;
proc format cntlin=cntl1;run;
proc print data=sample;
  format name $name. sex $seibetu.;
run;
```

(ログ)

```
61   data cntl1;
62     input fmtname :$8. start :$32. label :$60.;
63     type="C";
64     end=start;
65     hlo="";
66   cards;

NOTE: データセット WORK.CNTL1 は 5 オブザベーション、 6 変数です。
NOTE: DATA ステートメント 処理 (合計処理時間):
       処理時間                0.01 秒
       CPU 時間                0.00 秒

72   ;
73   proc print data=cntl1;run;

NOTE: データセット WORK.CNTL1 から 5 オブザベーションを読み込みました。
NOTE: PROCEDURE PRINT 処理 (合計処理時間):
       処理時間                0.00 秒
       CPU 時間                0.00 秒

74   proc format cntlin=cntl1;
NOTE: Format $SEIBETU は既にライブラリに存在します。
NOTE: 出力形式 $SEIBETU を作成しました。
NOTE: 出力形式 $NAME を作成しました。
```

(アウトプット)

```
OBS    fmtname    start     label    type    end       hlo
```

```
 1     seibetu    M          男         C       M
 2     seibetu    F          女         C       F
 3     seibetu    99999      不明       C       99999
 4     name       fujita     藤田       C       fujita
 5     name       suzuki     鈴木       C       suzuki

OBS    ID    name     sex     age     height     weight

 1      1    藤田     男       30       175         70
 2      2    鈴木     女       29       168         59
 3      3    taka     男       32       180         85
 4      4    tana     男       40       178         77
```

[DATA ステップによるレポート]

最後に DATA ステップでレポートの例を経験します。

(プログラム 2.2-14)　レポート例

```
proc sort data=sample;by sex;run;
data _null_;
  set sample end=end;
  by sex;

  file print header=header;

  if first.sex then do;
    n=0;          /* 性別の集計件数リセット */
    wk=0;         /* 性別のワーク変数のリセット */
  end;
  tot_n+1;        /* 全体件数のカウント */
  n+1;            /* 性別件数のカウント */
  tot_wk+height;/* 身長を足しこむ */
  wk+height;      /* 身長を足しこむ */
  if last.sex then do;
    mean_height=wk/n;  /* 性別の平均身長を計算 */
    put @5 "性別:" sex @15 "件数" +1 n "件" @30 "平均身長:" +1 mean_height 6.2 "cm";
    put @5 50*"-";
  end;
  if end=1 then do;
    tot_mean_height=tot_wk/tot_n;  /* 性別の平均身長を計算 */
    put @5 50*"=";
    put @5 "全体:"      @15 "件数" +1 tot_n "件" @30 "平均身長:" +1 tot_mean_height 6.2 "cm";
  end;
  return;

  header:
  put;
  put @18 "**** 計測データの集計 ***";
  put;
  tot_n=0;      /* 全体集計件数リセット */
  tot_wk=0;     /* 身長の平均を求めるためのワーク変数のリセット */
  retain tot_n tot_wk;
  return;
run;
```

(アウトプット)

```
                    **** 計測データの集計 ***

    性別:F     件数 1 件        平均身長: 168.00cm
    ------------------------------------------------
    性別:M     件数 3 件        平均身長: 177.67cm
    ------------------------------------------------
    ================================================
    全体:      件数 4 件        平均身長: 175.25cm
```

FIRST.BY 変数, LAST.BY 変数 の意味は以下のとおりです。これらの変数は値を与えることはできません。またこの DATA ステップの中で一時的に生成される変数です。

(プログラム2.2-15) FIRST.BY変数, LAST.BY変数の値

```
data _null_;
  set sample end=end;
  by sex;
  first_sex=first.sex;
  last_sex=last.sex;
  end_=end;
  put sex= first_sex= last_sex= end_=;
run;
```

(ログ)

```
sex=F first_sex=1 last_sex=1 end_=0
sex=M first_sex=1 last_sex=0 end_=0
sex=M first_sex=0 last_sex=0 end_=0
sex=M first_sex=0 last_sex=1 end_=1
```

[RETURN ステートメント, ラベルステートメント, LINK ステートメント, GOTO ステートメント]

RETURN ステートメントは実行中の DATA ステップのループ実行の途中で呼出元の位置に戻す役割があります。戻り位置が無い場合は、DATA ステップの最初に戻ります。

サブルーティン呼出しの代表的な例は、LINK ステートメントなどによるサブルーティンコールです。サブルーティンはラベルステートメントで始まり RETURN ステートメントで終わります。この例では FILE ステートメントの HEADER=ラベル指定によるサブルーティンコールの終了位置を示しており、呼び出した FILE ステートメントに戻ります。

(RETURN ステートメント) 実行ステートメント

RETURN;

呼出し元がある場合はラベル: RETURN; の形でラベルステートメントと対で指定します。この場合は RETURN ステートメントを実行すると、呼び出し元に戻ります。呼出元が無い場合は DATA ステップの最初の実行ステートメントに戻ります。

(ラベルステートメント) 宣言ステートメント

ラベル名:

このステートメントは文末にセミコロン(;)は不要です。(コロン(:)で終わるステートメントと解釈できます。)

(GOTO ステートメント)　実行ステートメント

GOTO　ラベル名;

プログラム実行をラベル名で参照する位置に移動します。

(LINK ステートメント)　実行ステートメント

LINK　ラベル名;

プログラム実行をラベル名で参照される位置に移動し、RETURN ステートメントで LINK ステートメントの次のステートメントに戻ります。

以上で 2 章データ加工は終了です。

3. レポーティング

3.1 集計、データセットの転値、SQL プロシジャ

～ FREQ、SUMMARY、TRANSPOSE、SQL プロシジャ

SAS のデータ集計機能は基本的に度数集計を行う FREQ プロシジャと連続変数の合計値や平均値を集計する MEANS(SUMMARY)プロシジャがあります。前半ではこのような統計計算行うプロシジャについて学習します。
プロシジャを用いたデータ集計は基本的に同じ変数（カラム）ごとにオブザベーション（行）方向に集計を行うものです。場合によっては、変数とオブザベーションが逆に並んでいるようなデータの形になっている場合もあり得ます。このような場合は、行列を逆にして（転値）しからプロシジャを使う必要がでてきます。そのために用意されている TRANSPOSE プロジャについて学習します。最後に SQL プロシジャを紹介します。

(プログラム3.1-1)　データの準備

```
data trans;
  input ID :$3. date :yymmdd8. itemno :$3. section :$10. sales;
  ym=put(year(date),4.)||"-"||put(month(date),z2.);
  itemgrp=substr(itemno,1,1);
  sectgrp=scan(section,1,"_");
  length=length(sectgrp);
cards;
101 20101001 A01 TOKYO_2   256
102 20101006 B02 TOKYO_1   600
103 20101102 A01 KANAGAWA_2   123
102 20101111 B03 TOKYO_2   850
103 20101125 A03 KANAGAWA_3   45
101 20101130 B03 TOKYO_3   520
;
options nocenter;
proc print data=trans;run;
proc contents data=trans;run;
```

(アウトプット)

OBS	ID	date	itemno	section	sales	ym	itemgrp	sectgrp	length
1	101	18536	A01	TOKYO_2	256	2010-10	A	TOKYO	5
2	102	18541	B02	TOKYO_1	600	2010-10	B	TOKYO	5
3	103	18568	A01	KANAGAWA_2	123	2010-11	A	KANAGAWA	8
4	102	18577	B03	TOKYO_2	850	2010-11	B	TOKYO	5
5	103	18591	A03	KANAGAWA_3	45	2010-11	A	KANAGAWA	8
6	101	18596	B03	TOKYO_3	520	2010-11	B	TOKYO	5

変数と属性の昇順リスト

#	変数	タイプ	長さ
1	ID	文字	3
2	date	数値	8
7	itemgrp	文字	3
3	itemno	文字	3

9	length	数値	8
5	sales	数値	8
8	sectgrp	文字	200
4	section	文字	10
6	ym	文字	7

［良く使う文字列操作演算子と文字関数］

ここで文字関数などについて紹介します。この例の DATA ステップでは文字列結合演算子(||)や文字関数(SUBSTR,SCAN)を用いて文字列をくっつけたり、一部を取り出す操作をしています。

・|| 演算子

|| 演算子は前後の文字列をつなぐ演算子です。[12]

例)

d=a||b||c;　文字変数 a,b,c の値をつなげた値を文字変数 d の値に割り当てます。注意：d の長さは右辺の文字変数の長さの合計になります。文字変数に入っているブランクはそのまま残ります。

a="ABC　"(長さ 5),

b="X　"(3 文字),

c="ZZ　"(4 文字)、このとき

d="ABC　X　ZZ　"(長さ 12)となります。

・SUBSTR 関数

SUBSTR(文字変数,開始位置,抽出文字数)　文字変数の開始位置から抽出文字数分の長さの文字列を抽出します。

例)

sub=substr(d,1,6);　sub="ABC　X" となります。注意：sub の長さは d の長さとなります。

・SCAN 関数

SCAN(文字変数,番号,"区切り文字")　指定の区切り文字のいずれかで文字列を区切ったとき、指定の番号の部分文字列を抽出します。

例)

a="ABCDEF11ABC1AB";

s1=scan(a,1,"ABC1");　s1="DEF"となります。 **注意：s1 の長さを事前に定義していなかった場合は SCAN 関数の結果の長さは 200 になります。**

・LENGTH 関数

LENGH(文字変数)　文字変数値の長さを返します。

・TRIM 関数　登場していませんが、文字列の後ろのブランクを取り除きます。

例)

d=trim(a)||trim(b)||trim(c);

さきほどの a,b,c の場合、d="ABCXZZ"　(長さ 6)になります。

・LEFT 関数　これも登場していませんが、数値を文字に変換したとき、右詰めされていますので、左詰

12 ||演算子と同じく文字列を連結する関数 CAT.CATS,CATT があります。CATS は文字列の前後のブランク、CATT は後ろのブランクのみを取り除いた上で連結を行います。

めするため用います。

LEFT(文字列)
例)

```
data _null_;
   do x=1 to 10;
      y=put(x, 2.);
      z="C"||left(y);
      put x= z=;
   end;
run;
```

(ログ)

```
x=1 z=C1
x=2 z=C2
x=3 z=C3
x=4 z=C4
x=5 z=C5
x=6 z=C6
x=7 z=C7
x=8 z=C8
x=9 z=C9
x=10 z=C10
```

LEFT 関数を使わないと、z=C 1 などと,間にブランクがあきます。

・COMPRESS 関数　登場していませんが、文字列の中に含まれるすべてのブランクを取り除きます。[13]

なお、**DBCS(Double Bytes Character Set)関数**という一連の文字列操作関数も SAS には用意されています。これらはいわゆる全角文字("漢字"、"カナ"、"かな"、"ＡＢＣ"など）と半角文字("abc","ABC","123"など)が混在した文字列に対して、全角文字半角文字それぞれを 1 個の文字として正しく認識して文字列操作を行うための関数です。(substr 関数に対して ksubstr 関数、scan 関数に対して kscan 関数、... というように頭に K が付いた文字列操作関数があります。) 例えば、a=length("あいう 123");と指定すると、a=9 ですが、b=klength("あいう 123");と指定すると b=6 となります。

では、本題の FREQ プロシジャの説明に入ります。

(プログラム 3.1-2) 単純集計

```
proc freq data=trans;
  tables ym itemgrp sectgrp;
run;
```

(アウトプット)

```
FREQ プロシジャ

                                        累積          累積
```

13 これに対して、文字列中のブランクは削除せず、さらに文字列の前後に 1 個ずつブランクをつけた形で連結を行う COMPBL 関数があります。

ym	度数	パーセント	累積 度数	累積 パーセント
2010-10	2	33.33	2	33.33
2010-11	4	66.67	6	100.00

itemgrp	度数	パーセント	累積 度数	累積 パーセント
A	3	50.00	3	50.00
B	3	50.00	6	100.00

sectgrp	度数	パーセント	累積 度数	累積 パーセント
KANAGAWA	2	33.33	2	33.33
TOKYO	4	66.67	6	100.00

(プログラム 3.1-3)　件数(度数)の多い順に表示

```
proc freq data=trans order=freq;
  tables ym itemgrp sectgrp;
run;
```

(アウトプット)

FREQ プロシジャ

ym	度数	パーセント	累積 度数	累積 パーセント
2010-11	4	66.67	4	66.67
2010-10	2	33.33	6	100.00

itemgrp	度数	パーセント	累積 度数	累積 パーセント
A	3	50.00	3	50.00
B	3	50.00	6	100.00

sectgrp	度数	パーセント	累積 度数	累積 パーセント
TOKYO	4	66.67	4	66.67
KANAGAWA	2	33.33	6	100.00

(プログラム 3.1-4)　クロス集計

```
proc freq data=trans;
  tables itemgrp*sectgrp;
run;
```

(アウトプット)

FREQ プロシジャ

```
表 : itemgrp * sectgrp

itemgrp        sectgrp

度数           |
パーセント     |
行のパーセント|
列のパーセント|KANAGAWA|TOKYO   |   合計
---------------+--------+--------+
A              |      2 |      1 |      3
               |  33.33 |  16.67 |  50.00
               |  66.67 |  33.33 |
               | 100.00 |  25.00 |
---------------+--------+--------+
B              |      0 |      3 |      3
               |   0.00 |  50.00 |  50.00
               |   0.00 | 100.00 |
               |   0.00 |  75.00 |
---------------+--------+--------+
合計                   2        4        6
                   33.33    66.67   100.00
```

(プログラム 3.1-5)　3 重クロス集計と TABLES ステートメントのオプション指定

```
proc freq data=trans;
  tables ym*itemgrp*sectgrp/norow nocol nopercent;
run;
```

(アウトプット)

```
FREQ プロシジャ

表 1 : itemgrp * sectgrp
層別変数 : ym=2010-10

itemgrp     sectgrp

度数     |KANAGAWA|TOKYO   |   合計
---------+--------+--------+
A        |      0 |      1 |      1
---------+--------+--------+
B        |      0 |      1 |      1
---------+--------+--------+
合計             0        2        2


表 2 : itemgrp * sectgrp
層別変数 : ym=2010-11

itemgrp     sectgrp

度数     |KANAGAWA|TOKYO   |   合計
---------+--------+--------+
A        |      2 |      0 |      2
---------+--------+--------+
B        |      0 |      2 |      2
---------+--------+--------+
```

合計	2	2	4

(プログラム 3.1-6)　度数集計結果のデータセット出力

```
proc freq data=trans;
  tables ym/noprint out=ym;
  tables itemgrp*sectgrp/noprint out=item_sect;
  tables ym*itemgrp*sectgrp/noprint out=ym_item_sect;
run;
proc print data=ym;run;
proc print data=item_sect;run;
proc print data=ym_item_sect;run;
```

(アウトプット)

```
OBS      ym       COUNT    PERCENT

 1     2010-10       2      33.3333
 2     2010-11       4      66.6667

OBS    itemgrp    sectgrp     COUNT    PERCENT

 1        A       KANAGAWA      2      33.3333
 2        A       TOKYO         1      16.6667
 3        B       TOKYO         3      50.0000

OBS      ym       itemgrp    sectgrp     COUNT    PERCENT

 1     2010-10       A       TOKYO         1      16.6667
 2     2010-10       B       TOKYO         1      16.6667
 3     2010-11       A       KANAGAWA      2      33.3333
 4     2010-11       B       TOKYO         2      33.3333
```

(プログラム 3.1-7)　出力データセットに行百分率と列百分率項目を追加する OUTPCT オプション

```
proc freq data=trans;
  tables ym/noprint out=ym(keep=ym count);
  tables itemgrp*sectgrp/noprint outpct out=item_sect;
run;
proc print data=ym;run;
proc print data=item_sect;run;
```

(アウトプット)

```
OBS      ym       COUNT

 1     2010-10       2
 2     2010-11       4

OBS    itemgrp    sectgrp     COUNT    PERCENT    PCT_ROW    PCT_COL

 1        A       KANAGAWA      2      33.3333     66.667       100
 2        A       TOKYO         1      16.6667     33.333        25
 3        B       TOKYO         3      50.0000    100.000        75
```

(プログラム 3.1-8)　カイ 2 乗検定を行いアイテム G とセクション G との関連の強さを調べる

```
proc freq data=trans;
  tables itemgrp*sectgrp/chisq;
```

```
run;
```

(アウトプット)

```
FREQ プロシジャ

表 : itemgrp * sectgrp

itemgrp        sectgrp

度数           |
パーセント     |
行のパーセント|
列のパーセント|KANAGAWA|TOKYO   |   合計
---------------+--------+--------+
A              |      2 |      1 |      3
               |  33.33 |  16.67 |  50.00
               |  66.67 |  33.33 |
               | 100.00 |  25.00 |
---------------+--------+--------+
B              |      0 |      3 |      3
               |   0.00 |  50.00 |  50.00
               |   0.00 | 100.00 |
               |   0.00 |  75.00 |
---------------+--------+--------+
合計                   2        4        6
                   33.33    66.67   100.00


itemgrp * sectgrp の統計量

統計量                              自由度          値       p 値
-------------------------------------------------------------------
カイ 2 乗値                            1       3.0000     0.0833
尤度比カイ 2 乗値                      1       3.8191     0.0507
連続性補正カイ 2 乗値                  1       0.7500     0.3865
Mantel-Haenszel のカイ 2 乗値          1       2.5000     0.1138
ファイ係数                                     0.7071
一致係数                                       0.5774
Cramer の V 統計量                             0.7071

 WARNING: セルの 100% において、期待度数が 5 より小さくなって
          います。カイ 2 乗検定は妥当な検定でないと思われます。


      Fisher の正確検定
-----------------------------
セル (1,1) 度数 (F)          2
左側 Pr <= F            1.0000
右側 Pr >= F            0.2000

表の確率 (P)            0.2000
両側    Pr <= P         0.4000

標本サイズ = 6
```

ここでは、どの商品もすべてのセクションで同じように売れているかどうかを**統計的に検定**した結果が表

示されています。この例ではアイテム A は神奈川でアイテム B は東京で売れているといった偏った傾向が見られます。このようにアイテム別の売上がセクション別で異なる場合、「**偏りがある**」、「**一様でない**」、「**関連がある**」、「**独立でない**」などと言います。データからこのことが普遍的かどうかを統計的に検定するときは、「**カイ 2 乗検定**」という「**統計的検定法**」を使います。TABLES ステートメントに **CHISQ オプション**を指定すると、クロス集計度数表からカイ 2 乗値を計算します。カイ 2 乗値の計算式は統計の本に掲載されていますので見ていただくとして、大事なのは、カイ 2 乗値を含むほぼすべての統計量は、「**帰無仮説**」とよばれる単純な状態（ここでは **2 つの項目には関連がなく一様な状態**）を仮定した場合に、このような**データが得られる確率**をカイ 2 乗値とカイ 2 乗値を求める 2 つ項目の (カテゴリー数-1) の積（これをカイ 2 乗値の「**自由度**」と言います) から計算することです。この確率を「**有意確率**」とか「**p 値**」と呼び、検定統計量の隣に表示されます。表示されたカイ 2 乗検定の有意確率 0.0833 は、2 つの項目が独立である分布状態からこのようなデータが得られる確率は 8.33%だということを示しています。**この有意確率値がある程度小さい場合**、帰無仮説を捨てて、2 つの項目には関連があると自信を持って結論づけるのが統計的仮説検定の目的です。帰無仮説に対して、採用する仮説の方を「**対立仮説**」と呼ぶこともあります。では、どのくらい関連があるかということが問題になりますが、1 回のサンプルデータで決める場合は、集計結果データに表れた偏りを推計結果とするのが妥当と考えます。

さて、最後に FREQ プロシジャの指定方法をまとめておきます。

[FREQ プロシジャ]

PROC FREQ DATA=入力データセット オプション;
TABLES 指定 / オプション;
BY 変数;
WEIGHT 変数;
RUN;

(PROC FREQ ステートメントの良く使うオプション)
・ORDER=DATA|FORMATTED|FREQ|INTERNAL
データセット出力、またはアウトプット表示する度数表のカテゴリの並び順を指定します。
デフォルトは INTERNAL(内部値)、つまりソート順です。
DATA ... 入力データセットにおける出現順,
FORMATTED ... フォーマットされた値の順(FORMAT ステートメントでフォーマット指定されている場合、また入力データセットにフォーマットが定義されている場合に有効)
FREQ ... 件数の多いカテゴリ順

(TABLES ステートメントの指定)

TABLES リクエスト;
リクエストは変数リストの形式(各変数の単純集計)または変数 1*変数 2*... 変数 k の形式の多重クロス集計のリクエストをブランクで区切って指定できます。
また TABLES ステートメントは複数指定できます。
例)
TABLES A B;
A,B それぞれの変数について単純集計を行います。

TABLES A*B;
変数 A と B のクロス集計を行います。

注意：数値変数を TABLES ステートメントに指定することは可能ですが、個々の値ごとの度数を集計するため、多量のアウトプットが表示されることがにつながります。数値変数の値の種類とそれぞれの該当件数を調べたい場合は NOPRINT オプションと OUT=オプションを指定してください。

(TABLES ステートメントで良く使うオプション)
・NOPRINT
アウトプット表示を行わないようにします。一般に次の OUT=オプションと一緒に使います。
・OUT=出力データセット名
度数集計結果を SAS データセットに出力します。自動変数 COUNT と PERCENT が追加されます。TABLES ステートメントに複数のリクエスト指定があると、最後のリクエストに関する集計結果のみ指定のデータセットに出力します。複数のリクエストの集計結果をすべてデータセット出力するには、個々のリクエストのみを TABLES ステートメントに指定し、TABLES ステートメントを複数指定してください。
・OUTPCT
多重集計リクエストで OUT=指定がある場合に有効。列百分率と行百分率を表す自動変数(PCT_COL, PCT_ROW)が出力データセットに追加されます。
・SPARSE
他次元クロス集計の場合、出現していない場合を含めて、すべての変数値の可能な組合せの集計件数をデータセットに出力します。
・CHISQ
2 次元度数集計表におけるカイ 2 乗検定を行います。
・MEASURES
カイ 2 乗関連以外のさまざまな関連度の指標をアウトプットします。

(WEIGHT ステートメント)

WEIGHT 変数名;

重み変数を指定します。指定された変数の値だけそのオブザベーションがあるものとみなします。集計データからクロス集計表を作成するときなどに使用します。変数値の小数点以下は切り捨てられます。また欠損値は 0 とみなされます。WEIGHT ステートメントは多くのプロシジャで共通に使えるステートメントです。

次は平均、合計、標準偏差といった数値変数の集計を行う MEANS プロシジャと SUMMARY プロシジャです。

(プログラム 3.1-9)　MEANS プロシジャ

```
proc means data=trans;
  var sales;
run;
```

(アウトプット)

```
MEANS プロシジャ

                        分析変数 : sales

N            平均          標準偏差          最小値          最大値
--------------------------------------------------------------------
```

```
6       399.0000000       309.9625784        45.0000000       850.0000000
-------------------------------------------------------------------------
```

(プログラム 3.1-10)　統計量のリクエスト

```
proc means data=trans min max range mean median sum uss css var std;
  var sales;
run;
```

(アウトプット)

分析変数 ： sales

最小値	最大値	範囲	平均	中央値	合計	無修正平方和
45.0000000	850.0000000	805.0000000	399.0000000	388.0000000	2394.00	1435590.00

修正済平方和	分散	標準偏差
480384.00	96076.80	309.9625784

Excel で計算

N	x	x^2	Σ x^2	avr(X)	x-avr(x)	(x-avr(x))^2	Σ (x-avr(x))^2	Σ (x-avr(x))^2/(N-1)	SQRT(Σ(x-avr(x))^2/(N-1))
6	256	65536	1435590	399	-143	20449	480384	96076.80	309.9625784
	600	360000		399	201	40401			
	123	15129		399	-276	76176			
	850	722500		399	451	203401			
	45	2025		399	-354	125316			
	520	270400		399	121	14641			
								Σ (x-avr(x))^2/N	SQRT(Σ(x-avr(x))^2/N)
								80064.00	282.9558269

連続変数値に関する統計量は、**位置**に関するもの(**平均値**や**中央値**)、そして**変動(バラツキ)**に関するもの(**範囲**や**分散**や**標準偏差**)があります。バラツキは**個々の値の全体平均値からの距離**（統計用語では**偏差**と呼びます）を集計して求めます。偏差はプラスマイナスがあり、単純に足し算するわけにいかないので、偏差を 2 乗した値をすべてのデータについて足しこみ、（これを**偏差平方和**と呼びます。Σ偏差^2 ＝Σ(x^avr(x)^2) と書いた記法がそれを表しています）偏差平方和を件数 N で割ってから平方根をとって元の変数の尺度に戻します(標準偏差)。したがって、**データ 1 個あたりの偏差の平均**というのが標準偏差の意味です。ただし、2 乗した値の平均をとってから平方根をとりますので、大きい偏差の値の重みが大きくなります。標準では、件数 N で無くて件数 N から 1 を引いた値で割っています。これはこの値がデータの**自由度**を表し、求める分散をこのデータを抽出した母集団における標準偏差を推計するために用いるためです。6 個のデータから平均値という分布の特性値(母数、パラメータと呼ばれます)を計算して得ているので、6 個の内 5 個のデータの値を知れば、残りの 1 個のデータ値は判明します。したがって手元にあるデータの自由度は 5 ということになり、偏差平方和を自由度で割った値を母集団における分散推計値としています。なお、このように自由度で割った分散のことを**不偏分散**とよび、件数 N で割った分散のことを**標本分散**といって区別します。なお、単に分散とか標準偏差と言った場合は、不偏分散や自由度で割った方の標準偏差（不偏標準偏差）を意味する場合が多いと思われます。

(プログラム 3.1-11)　標本分散と標本標準偏差のリクエスト

```
proc means data=trans vardef=N var std;
  var sales;
run;
```

(アウトプット)

```
MEANS プロシジャ

        分析変数 : sales

          分散          標準偏差
----------------------------
     80064.00       282.9558269
----------------------------
```

(プログラム 3.1-12)　NOPRINT オプションと統計量のデータセット出力

```
proc means data=trans noprint;
  var sales;
  output out=sales_stat mean=mean std=std min=min max=max n=n;
run;
proc print data=sales_stat;run;
```

(アウトプット)

```
OBS    _TYPE_    _FREQ_    mean      std       min    max    n

 1        0         6       399    309.963     45     850    6
```

(プログラム 3.1-13)　複数の変数の統計量を出力データセットにリクエスト

```
proc means data=trans noprint;
  var sales length;
  output out=sales_stat2 mean=mean1 mean2 std=std1 std2;
run;
proc print data=sales_stat2;run;
```

(アウトプット)

```
OBS    _TYPE_    _FREQ_    mean1    mean2      std1       std2

 1        0         6       399       6       309.963    1.54919
```

(プログラム 3.1-14)　AUTONAME オプション

```
proc means data=trans noprint;
  var sales length;
  output out=sales_stat2 mean= std=/autoname;
run;
proc print data=sales_stat2;run;
```

(アウトプット)

```
                              sales_     length_     sales_     length_
OBS    _TYPE_    _FREQ_       Mean        Mean       StdDev      StdDev

 1        0         6         399         6         309.963    1.54919
```

(プログラム 3.1-15)　グループ別集計

```
proc means data=trans n mean sum;
  class sectgrp itemgrp;
  var sales;
run;
```

(アウトプット)

```
MEANS プロシジャ

                                    分析変数 : sales

                                        オブザベーション
sectgrp                          itemgrp                  数  N          平均          合計
------------------------------------------------------------------------------------------
KANAGAWA                         A                         2  2    84.0000000   168.0000000

TOKYO                            A                         1  1   256.0000000   256.0000000

                                 B                         3  3   656.6666667       1970.00
------------------------------------------------------------------------------------------
```

(プログラム 3.1-16)　グループ別集計 アウトプット表示桁数の指定とデータセット出力

```
proc means data=trans n mean sum fw=10 maxdec=0;
  class sectgrp itemgrp;
  var sales;
  output out=group_stat n= mean= sum=/autoname;
run;
proc print data=group_stat;run;
```

(アウトプット)

```
MEANS プロシジャ
                                    分析変数 : sales

                                        オブザベーション
sectgrp                              itemgrp                  数  N        平均        合計
------------------------------------------------------------------------------------------
KANAGAWA                             A                         2  2          84         168

TOKYO                                A                         1  1         256         256

                                     B                         3  3         657        1970


                                                                 sales_     sales_
OBS    sectgrp     itemgrp    _TYPE_    _FREQ_    sales_N      Mean        Sum

 1                               0         6          6      399.000     2394
 2                  A            1         3          3      141.333      424
 3                  B            1         3          3      656.667     1970
 4     KANAGAWA                  2         2          2       84.000      168
 5     TOKYO                     2         4          4      556.500     2226
 6     KANAGAWA     A            3         2          2       84.000      168
 7     TOKYO        A            3         1          1      256.000      256
 8     TOKYO        B            3         3          3      656.667     1970
```

TYPE　自動変数の値は以下のようにどのような集計レベルを表すオブザベーションであるかを識別します。

CLASS ステートメントに指定した変数の数だけビットパターンを考えます。

この場合は sectgrp,itemgrp の 2 つの変数を指定していますので、2 の 2 乗=4 通りのビットパターンがあります。

_TYPE_値が示す集計レベル

sectgrp	itemgrp	2進数	_TYPE_値	集計レベル
0	0	"00"B	0	全体
0	1	"01"B	1	itemgrp別
1	0	"10"B	2	sectgrp別
1	1	"11"B	3	sectgrp別itemgrp別

なお、_FREQ_ 自動変数は CLASS 変数値で識別されるグループに該当する入力オブザベーション数を表します。統計量キーワード N が欠損値で無いオブザベーション数を表すのに対して、_FREQ_ は欠損値を含む件数です。

(プログラム 3.1-17)　NWAY オプションを指定して一番深い _TYPE_ レベルのみリクエスト

```
proc means data=trans noprint nway;
  class sectgrp itemgrp;
  var sales;
  output out=group_stat n= mean= sum=/autoname;
run;
proc print data=group_stat;run;
```

(アウトプット)

OBS	sectgrp	itemgrp	_TYPE_	_FREQ_	sales_N	sales_ Mean	sales_ Sum
1	KANAGAWA	A	3	2	2	84.000	168
2	TOKYO	A	3	1	1	256.000	256
3	TOKYO	B	3	3	3	656.667	1970

(プログラム 3.1-18)　特定の _TYPE_ レベルのみ集計するようリクエスト

```
proc means data=trans noprint;
  class sectgrp itemgrp;
  types () sectgrp sectgrp*itemgrp ;
  var sales;
  output out=group_stat n= mean= sum=/autoname;
run;
proc print data=group_stat;run;
```

(アウトプット)

OBS	sectgrp	itemgrp	_TYPE_	_FREQ_	sales_N	sales_ Mean	sales_ Sum
1			0	6	6	399.000	2394
2	KANAGAWA		2	2	2	84.000	168
3	TOKYO		2	4	4	556.500	2226
4	KANAGAWA	A	3	2	2	84.000	168
5	TOKYO	A	3	1	1	256.000	256
6	TOKYO	B	3	3	3	656.667	1970

では、MEANS プロシジャの指定方法をまとめておきます。

[MEANS プロシジャ]
PROC MEANS DATA=入力データセット オプション;
CLASS 変数 / オプション;
TYPES リクエスト;
WAYS リスト;
VAR 変数;
ID 変数;
FREQ 変数;
WEIGHT 変数;
OUTPUT OUT=出力データセット キーワード=変数;
RUN;

(PROC MEANS ステートメントの良く使うオプション)
・統計量キーワード
たくさんありますが、 N,MEAN,SUM,STD,VAR,MIN,MAX,MEDIAN などを良く使うでしょう。
・MISSING
CLASS ステートメントで指定した分類変数の欠損値を有効な 1 つのグループとして集計対象にします。
数値変数の特殊欠損値(.A~.Z および._)もそれぞれ別々の有効なグループとみなします。
注：FREQ プロシジャの場合は MISSING オプションを指定しなくても、OUT=出力データセットには TABLES ステートメントの変数に欠損があっても必ずそのカテゴリは出力されます。それに対して、**MEANS プロシジャの場合は MISSING オプションを指定しないと、出力データセットには CLASS ステートメントに指定した変数の欠損カテゴリは出力されません。**
・NOPRINT
アウトプット画面への結果表示を抑制します。
・NWAY
CLASS ステートメントに指定した全変数のカテゴリ組み合わせの集計結果のみデータセット出力します。
画面出力には無関係です。TYPES ステートメントまたは WAYS ステートメントが指定された場合このオプションは無効になります。
・VARDEF=DF|N|WDF|WGT
分散と標準偏差を計算するときの分母を指定します。DF(自由度)がデフォルト。N は非欠損値の数、WDF と WGT は WEIGHT ステートメントの指定がある場合の重み合計の自由度と非欠損件数。

(良く使うステートメント)

・CLASS 変数;
分析で使う分類変数を指定しておきます。
・TYPES リクエスト;
CLASS ステートメントで指定した変数の集計したい組み合わせ方を指定します。
例)
CLASS a b c;
TYPES () a b a*c; ()は全体(_TYPE_=0)のリクエストを表します。
全体と a と b の単純層別と a と c のクロス層別で分類した集計をリクエストしています。

・WAYS 数字;

CLASS ステートメントで指定した変数の組み合わせ数を指定します。
例)
CLASS a b c;
WAYS 2;　a,b,c の 3 変数から 2 変数を取る全組み合わせを層別とする集計をリクエストします。
TYPES ステートメントの指定とは OR 条件で働きます。

・ID,BY,WEIGHT,FREQ,VAR ステートメント　これらは多くのプロシジャで共通に使えるステートメントです。なお、WEIGHT ステートメントと FREQ ステートメントの違いは、WEIGHT ステートメントはオブザベーションの相対的重みを表し、少数点以下の値も有効で統計量の計算に使われるのに対して、FREQ ステートメントは実際にオブザベーションが変数値の数だけ重複して入力されたものとみなし、整数値に切り捨てされて用いられるということです。ただし、度数集計を行う FREQ プロシジャでは WEIGHT ステートメントが FREQ ステートメントの意味で働きます。

なお、MEANS プロシジャは SUMMARY プロシジャという別名を持っています。以上で MEANS プロシジャの説明は終わりです。

［データセットの転値］

Excelの「形式を選択して貼り付け」ダイアログにある「行列を入れ替える」を実行するSASのプロシジャがTRANSPOSEです。

(transデータセット)

OBS	ID	date	itemno	section	sales	ym	itemgrp	sectgrp	length
1	101	18536	A01	TOKYO_2	256	2010-10	A	TOKYO	5
2	102	18541	B02	TOKYO_1	600	2010-10	B	TOKYO	5
3	103	18568	A01	KANAGAWA_2	123	2010-11	A	KANAGAWA	8
4	102	18577	B03	TOKYO_2	850	2010-11	B	TOKYO	5
5	103	18591	A03	KANAGAWA_3	45	2010-11	A	KANAGAWA	8
6	101	18596	B03	TOKYO_3	520	2010-11	B	TOKYO	5

(プログラム3.1-19)　SASデータセットの転値(オブザベーションと変数の入れ替え)

```
proc transpose data=trans out=trans2;
run;
proc print data=trans2;run;
```

(ログ)

```
NOTE: データセット WORK.TRANS から 6 オブザベーションを読み込みました。
NOTE: データセット WORK.TRANS2 は 3 オブザベーション、 7 変数です。
```

(アウトプット)

OBS	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6
1	date	18536	18541	18568	18577	18591	18596
2	sales	256	600	123	850	45	520
3	length	5	5	8	5	8	5

数値タイプの3個の変数(date,sales,length)のみ選択され転値されました。元の変数名は 自動変数 _NAME_ の値となっています。元のデータセットの1番目のオブザベーションの値は 自動的に 変数 COL1 の値になり、2番目のオブザベーションは変数 COL2 、... 6番目のオブザベーションは変数 COL6の値に格納されました。

(プログラム3.1-20)　転値したい変数の指定

```
proc transpose data=trans out=trans2;
  var itemgrp sectgrp;
run;
proc print data=trans2;run;
```

(アウトプット)

OBS	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6
1	itemgrp	A	B	A	B	A	B
2	sectgrp	TOKYO	TOKYO	KANAGAWA	TOKYO	KANAGAWA	TOKYO

SASデータセットはデータセルごとに**数値型、文字型のタイプを混在させることはできません**ので、VARステートメントに指定できる変数タイプはすべて数値型か、すべて文字型を指定する必要があります。

(プログラム3.1-21)　BYグループ処理

```
proc freq data=trans;
  table sectgrp*itemgrp/noprint out=cross(drop=percent);
run;
proc print data=cross;run;
proc transpose data=cross out=cross2;
  by sectgrp;
  id itemgrp;
  var count;
run;
proc print data=cross2;run;
```

(アウトプット)

OBS	sectgrp	itemgrp	COUNT
1	KANAGAWA	A	2
2	TOKYO	A	1
3	TOKYO	B	3

OBS	sectgrp	_NAME_	_LABEL_	A	B
1	KANAGAWA	COUNT	度数	2	.
2	TOKYO	COUNT	度数	1	3

以上のようにTRANSPOSEプロシジャを使ってSASデータセットのオブザベーションと変数を変換することができます。
注意点としては、次の点が挙げられます。
・ID 変数値は BYステートメントがあればBYグループ内で、無ければ全オブザベーションで32文字以内でユニークで無ければなりません。またID変数値にSAS変数名の命名規則に違反する文字があったときは、すべて "_" アンダースコアに変換されます。なお、数値変数をID変数指定すると、変数名の頭に必ず"_"がつきます。

[TRANSPOSE プロシジャ]

PROC TRANSPOSE DATA=入力データセット　OUT=出力データセット　オプション;
VAR　変数;

BY 変数;
ID 変数;
RUN;

(PROC TRANSPOSEステートメント)
・OUT=指定を省略すると、_DATA_データセットが指定されたものとみなし、 DATAn (nは数字)の自動データセット名が作成されます。
・PREFIX=変数名のPREFIX
転値後の自動変数名 COL1,COL2,...,COLn のCOLの代わりにつけるPREFIXを指定します。IDステートメントが指定されたなら、各ID値の頭に指定したPREFIXが付いた変数名となります。
例)
prefix=VAR
VAR1,VAR2,... という名前の変数になります。

[SQL プロシジャ]

SAS の中で SQL の文法によるデータ検索、作成を行うプロシジャが SQL プロシジャです。
PROC SQL ステートメントで SQL プロシジャに入ると、SQL の世界の構文ルールに従います。RUN ステートメントは無意味で QUIT ステートメントで SQL プロシジャを終了します。SQL 句のセパレータはブランクでは無くカンマが使われます。PROC SORT でソートしていなくても ORDER BY 句を使うことができます。PROC PRINT を行わなくても SQL の中で SELECT ステートメントをサブミットすると検索結果がアウトプット画面に表示されます。 というように SAS の中から SQL の世界が開きます。

(プログラム 3.1-22) テーブル定義とデータ挿入

```
options nocenter nodate nonumber;
proc sql;
  create table sample
    (ID char(3),
     sex char(1),
     birth num informat=date9. format=yymmdds10.,
     height num);
  insert into sample
     values("001","M","01JAN1960"D,175)
     values("002","F","11DEC1985"D,160);
  title "PROC SQL SELECT Statement";
  select * from sample;
```

(ログ)

```
51   options nocenter nonumber nodate;
52   proc sql;
53     create table sample
54       (ID char(3),
55        sex char(1),
56        birth num informat=date9. format=yymmdds10.,
57        height num);
NOTE: テーブル WORK.SAMPLE (行数 0、列数 4) が作成されました。
58     insert into sample
59        values("001","M","01JAN1960"D,175)
60        values("002","F","11DEC1985"D,160);
NOTE: 2 行が WORK.SAMPLE に挿入されました。

61     title "PROC SQL SELECT Statement";
```

```
62     select * from sample;
```

(アウトプット)

```
PROC SQL SELECT Statement

ID   sex       birth    height
------------------------------
001  M    1960/01/01       175
002  F    1985/12/11       160
```

(プログラム 3.1-23)　既存のテーブル読込みによるテーブル作成

```
proc sql;
  create table sample2 as
  select ID, birth as birth2
  from sample;

proc print data=sample2;run;
```

(ログ)

```
64   proc sql;
65     create table sample2 as
66     select ID, birth as birth2
67     from sample;
NOTE: テーブル WORK.SAMPLE2 (行数 2、列数 2) が作成されました。
```

(アウトプット)

```
PROC SQL SELECT Statement

OBS    ID         birth2

 1     001    1960/01/01
 2     002    1985/12/11
```

(プログラム 3.1-24)　テーブル結合

```
data japan;
  input ID :$3. name :$10. sex :$1.;
cards;
001 fujita M
002 suzuki F
003 takahashi M
004 tanaka M
;
data us;
  input ID :$3. name :$10. age height weight;
cards;
101 browne 30 160 51
102 gibson 40 160 48
;
proc sql number;
  select id, name, sex from japan
  union
  select id, name, "", age, height, weight from us;
```

(アウトプット)

```
PROC SQL SELECT Statement
```

```
  Row  ID   name        sex       age    height    weight
-------------------------------------------------------
    1  001  fujita      M           .         .         .
    2  002  suzuki      F           .         .         .
    3  003  takahashi   M           .         .         .
    4  004  tanaka      M           .         .         .
    5  101  browne                 30       160        51
    6  102  gibson                  40       160        48
```

(プログラム 3.1-25)　VIEW の作成と集計

```
data trans;
  input ID :$3. date :yymmdd8. itemno :$3. section :$10. sales;
  ym=put(year(date),4.)||"-"||put(month(date),z2.);
  itemgrp=substr(itemno,1,1);
  sectgrp=scan(section,1,"_");
  length=length(sectgrp);
cards;
101 20101001 A01 TOKYO_2   256
102 20101006 B02 TOKYO_1   600
103 20101102 A01 KANAGAWA_2   123
102 20101111 B03 TOKYO_2   850
103 20101125 A03 KANAGAWA_3   45
101 20101130 B03 TOKYO_3   520
;
proc sql;
  create view view1 as
  select id, count(id) as number label="件数", avg(sales) as average_sales label="平均売上"
  from trans
  group by id
  having average_sales ge 100;

proc print data=view1;run;
```

(ログ)

```
199  proc sql;
200    create view view1 as
201    select id, count(id) as number label="件数", avg(sales) as average_sales label="平均売
上"
202    from trans
203    group by id
204    having average_sales ge 100;
NOTE: SQL ビューは WORK.VIEW1 定義されました。
```

(アウトプット)

```
PROC SQL SELECT Statement

                              average_
     OBS    ID      number       sales

       1    101        2          388
       2    102        2          725
```

というように際限がありませんので、このあたりで終わりにします。

SQL プロシジャは ANSI(米国規格協会) が発表している SQL ガイドラインにほとんど適合しています。したがって、SQL 言語が得意な人は簡単に使いこなすことができると思います。ただし、SAS の SQL プロシジャは SAS 環境で使うことを前提としているため、データベース操作関連の機能(Commit や Roll Back 機能など)は含まれていません。[14]
指定方法は膨大となるため、ここで記載できません。マニュアル等をご参照ください。

［データベースからデータ抽出のための SQL プロシジャ］

補足として、SAS ACCESS プロダクトを導入して、ORACLE, DB2, Teradata などのデータベーステーブルからデータを SAS データセットに抽出[15]する場合の SQL を例示します。

[SQL パススルー方式による DB2 データベースからのデータアクセス例]

```
proc sql;
  connect to DB2(user=username password=password datasrc=sample);
  create table A as
  select *
  from connection to DB2
  ( select *
   from STAFF
   where job='Mgr'
  );
  disconnect from DB2;
quit;
```

上記の例は DB2 データベース sample 内のテーブル STAFF の全項目を 列名 job の値が 'Mgr' に一致するレコードのみ SAS データセット WORK.A に読み取るプログラム例となっています。 () 内にデータベース用 SQL を記述します。

同じ結果は、以下の(A),(B)いずれの指定でも得られます。しかし、パススルー方式の方が高速です。
なお、(A)方式の場合はデータベース用 SQL では無く、SAS の SQL として有効でなければなりません。

(A)

```
libname in DB2 datasrc=sample;
proc sql;
  create table A as
  select *
  from in.STAFF
  where job='Mgr'
  ;
quit;
```

(B)

```
libname in DB2 datasrc=sample;
data A;
  set in.STAFF;
  if job='Mgr';
run;
```

14 データベースとのインターフェースをとる SAS ACCESS プロダクトを導入するとデータベース関連のコマンドも利用可能になります。

15 各データベースごとに SAS ACCESS プロダクトの導入が必要です。

3.2 レポーティングとマクロ処理
~ TABULATE プロシジャとマクロ処理

SAS 講習の最後は SAS のレポーティング関連で SAS ユーザの高い支持を受けていると思われる**TABULATE プロシジャ**を中心に学習します。TABULATE プロシジャは罫線入りの多重クロス表を作成するためのプロシジャですが、表のレイアウトやセルに入れる統計量や比率の指定が少ない指定で行えるという特長があります。明細データから度数や合計値や平均値などのクロス集計表を作成することもできますが、大容量データに対しては、一旦 FREQ や MEANS(SUMMARY)を用いて要約結果をデータセットに出力しておいて、そのデータセットの入力値から作表する方が、作りたい表の形式を試行錯誤するような場合は時間的に有利になります。このように FREQ や MEANS を組み合わせて TABULATE を使うこともあります。

簡易なレポートを作成するためには **REPORT プロシジャ**があります。しかし、このプロシジャが行えるのは 2 日目の最後に紹介した DATA ステップで作成するレポートと同程度のレポートであり、使うメリットがそれほど無いと思われますので、講習対象から外しました。

最後の**マクロ処理**は、SAS に慣れたユーザがさらに便利に SAS を使えるようにするための機能です。これまで見てきたように、SAS 言語のプログラミングとは、DATA ステップと PROC ステップという実行単位に分けて逐次的に実行するよう最初から最後のステップまで明示的に書いた SAS ステートメントの集合という形でした。しかし、SAS マクロ機能を使うと、**プログラムの途中で得られた計算結果によって、別のプロシジャを実行したり、別のデータ加工処理を行う**いった処理を**「事前に」**プログラミングしておくことができます。また、汎用的に使いたいプログラムを **SAS マクロ**として定義しておくことにより、部品として再利用しやすくなります。SAS マクロ機能は SAS マクロ言語という言語体系を持っており、SAS 言語と良く似たステートメントや関数、演算子などが使えますが、すぐに使いこなせるものではありません。ここではマクロ言語を使った簡単なプログラムの例を実行し、SAS マクロ機能を理解することを学習の目標とします。

[TABULATE プロシジャ]

これまで学習したように、FREQ プロシジャや MEAN プロシジャは、明細データから分類変数値の組み合わせごとに度数集計や数値変数の合計や平均を集計しますが、画面アウトプットはあまり見やすいものではありません。**FREQ プロシジャ**は、分類変数を重ねた度数集計を他のどのプロシジャより素早く実行する能力を持っていますが、**3 つ以上の分類変数の組み合わせを同じクロス集計表の中に表示する能力を持っていません**。また、**MEANS プロシジャ**は分類変数のすべての組み合わせ階層における連続変数の集計値を同時に計算する能力がありますが、**分類変数の値の組み合わせを識別変数として統計量を横一線に並べて表示する以外の表現方法を持っていません**。**TABULATE プロシジャ**はこのような分類変数値の組み合わせを、われわれが認識しやすいような**多重クロス表のイメージで表現**します。

(プログラム 3.1-1 で作成した trans データセット)

OBS	ID	date	itemno	section	sales	ym	itemgrp	sectgrp	length
1	101	18536	A01	TOKYO_2	256	2010-10	A	TOKYO	5
2	102	18541	B02	TOKYO_1	600	2010-10	B	TOKYO	5
3	103	18568	A01	KANAGAWA_2	123	2010-11	A	KANAGAWA	8
4	102	18577	B03	TOKYO_2	850	2010-11	B	TOKYO	5
5	103	18591	A03	KANAGAWA_3	45	2010-11	A	KANAGAWA	8

6	101	18596	B03	TOKYO_3	520	2010-11	B	TOKYO	5

(プログラム 3.2-1)　度数集計

```
options ls=132 ps=60;
proc tabulate data=trans;
  class itemgrp sectgrp;
  tables itemgrp, sectgrp;
run;
```

(アウトプット)

	sectgrp	
	KANAGAWA	TOKYO
	N	N
itemgrp		
A	2.00	1.00
B	.	3.00

CLASS ステートメントは MEANS プロシジャと同じくプロシジャで用いる分類変数を指定します。
TABLE ステートメントに登場する分類変数は必ず CLASS ステートメントで指定しておく必要があります。

TABLES ステートメントで作成したい表イメージをリクエストします。
・アウトプットは FREQ プロシジャの表示と良く似ています。
・TABLES ステートメントの中の **カンマ(,) は、作成する表の次元**を意味します。カンマは 0 個から 2 個まで指定でき、それぞれ 1 次元から 3 次元までの表の作成を意味します。
この例の場合、**カンマは 1 個ですので、2 次元の表をリクエスト**することになります。カンマの左側の指定は行方向(**行次元**)、右側の指定は列方向(**列次元**)に配置された形のクロス表イメージを作成します。
この例の行次元には指定した分類変数 itemgrp で変数名と値が各行に表示されています。また列次元には分類変数 sectgrp が指定され、変数名の行、値の行、さらに統計量(N)の 3 行にわたって表示されています。
・表の**セルの値**は次元のクロスで定義される**分類変数の組み合わせ**と**統計量**の種類に従って集計されます。この例では分類変数同士の組み合わせのときのデフォルトの統計量である度数(N)が自動的に用いられています。

なお、OPITONS ステートメントで指定している LS=,PS=はそれぞれアウトプット画面の幅(LineSize)と 1 ページの長さ(PageSize)を指定します。LS は 64 から 256 まで、PS は 15 から 32767 までの範囲で指定できます。

(プログラム 3.2-2)　次元の指定

```
proc tabulate data=trans;
  class itemgrp sectgrp ym;
  tables itemgrp sectgrp ym;
  tables itemgrp, sectgrp ym;
  tables itemgrp, sectgrp, ym;
run;
```

(アウトプット)

itemgrp		sectgrp		ym	
A	B	KANAGAWA	TOKYO	2010-10	2010-11
N	N	N	N	N	N
3.00	3.00	2.00	4.00	2.00	4.00

	sectgrp		ym	
	KANAGAWA	TOKYO	2010-10	2010-11
	N	N	N	N
itemgrp				
A	2.00	1.00	1.00	2.00
B	.	3.00	1.00	2.00

itemgrp A

	ym	
	2010-10	2010-11
	N	N
sectgrp		
KANAGAWA	.	2.00
TOKYO	1.00	.

itemgrp B

	ym	
	2010-10	2010-11
	N	N
sectgrp		
TOKYO	1.00	2.00

TABLES ステートメントにカンマなし、1 個、2 個をそれぞれ指定した場合のアウトプットを表示しました。

・1 次元指定は列方向への配置を意味します。

・2 次元指定は行と列に配置されます。

・3 次元指定は FREQ プロシジャの BY グループ処理と同じように、ページと行と列に配置されます。

(プログラム 3.2-3)　ブランクとアスタリスク演算子

```
proc tabulate data=trans;
  class itemgrp sectgrp ym;
  tables itemgrp sectgrp, ym;
  tables itemgrp*sectgrp, ym;
run;
```

(アウトプット)

	ym	
	2010-10	2010-11
	N	N
itemgrp		
A	1.00	2.00
B	1.00	2.00
sectgrp		
KANAGAWA	.	2.00
TOKYO	2.00	2.00

		ym	
		2010-10	2010-11
		N	N
itemgrp	sectgrp		
A	KANAGAWA	.	2.00
	TOKYO	1.00	.
B	TOKYO	1.00	2.00

・**ブランク( )**は 前後の表現を「**並列**」させる意味を持ちます。
・**アスタリスク(*)** は前後の表現を「**交差(クロス)**」させる意味を持ちます。
なお、カンマ(,)もアスタリスク(*)の意味を持ちます。

(セルの参照)
・最初の TABLES ステートメントは統計量 N の指定が省略されています。

tables itemgrp sectgrp, ym***N**;

この*N の*はクロスの意味ではなく統計量の指定を行うときの決まりです。***統計量キーワード** という書き方をします。(例 *mean *sum *std など)

TABULATE がこの TABLES ステートメントをどのように解釈して表を作成しているかをなぞってみましょう。

(1) カンマが 1 個あるので、行次元の指定 itemgrp sectgrp を解釈して、分類変数 itemgrp をまず配置し、(変数名と itemgrp の各値をそれぞれ 1 つの行に配置)、その下に分類変数 sectgrp を並列配置（変数名 と sectgrp の各値をそれぞれ 1 つの行に配置）します。一方、列次元の指定 ym*N を解釈して、分類変数名 ym を表示する行、ym の値を表示する行、そして統計量 N を表示する行の 3 行を作成します。これで表のイメージが完成しました。
(2) 次に各セルの値を計算します。行と列が交差するセルにはその行変数の値と列変数の値で定義されるグループについて指定された統計量を計算します。この指定の場合は、上の 4 つのセルは行次元分類変数 itemgrp と 列次元分類変数 ym との交差を表すセルとなっており、統計量は N です。また下の 4 つのセルは行次元分類変数 sectgrp と列次元分類変数 ym, との交差を表すセルです。

以上、TABULATE がどのように TABLES ステートメントを解釈し表のイメージとセルの値を計算するかを簡単に説明しました。

さて、カンマがアスタリスクと同じく、次元間の指定同士を交差させる働きがあることに注意すると、表のイメージは異なりますが、この指定は、次の指定と同じです。

tables itemgrp*ym*N sectgrp*ym*N;

これは最初の指定を展開した形となっており、表の形こそ異なりますが、生成されるセルの数と値は同じになります。TABULATE を使って、アスタリスク(交差)とブランク(並列)を組み合わせて複雑な表を作成しようとすると、すべてのセルが矛盾の無い分類定義と 1 つの統計量（さらに次で登場する 1 つのフォーマット）で定義されていないと作表不能となるため、最初はエラーの嵐に見舞われることになると思います。このとき、TABLES ステートメントの指定が正しいかどうかを調べるのに、1 次元に展開してみることをお勧めします。分類定義や統計量などの各要素の指定に重複や矛盾がないかどうかを確認しやすくなります。

2 番目の TABLES ステートメントの指定を 1 次元に展開して各セルの定義を説明してください。

```
tables itemgrp*sectgrp, ym;
カンマを展開すると？？？
```

(プログラム 3.2-4)　分析変数の指定と変数ラベル、統計量、フォーマットの指定

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp;
  var sales;
  tables itemgrp="アイテムＧ", sectgrp="セクションＧ"*sales="売上"
    *(n="件数" sum="合計"*f=comma8. mean="平均"*f=6.1)
    /rts=12;
run;
```

(アウトプット)

	セクションG					
	KANAGAWA			TOKYO		
	売上			売上		
	件数	合計	平均	件数	合計	平均
アイテムG						
A	2	168	84.0	1	256	256.0
B	.	.	.	3	1,970	656.7

VAR ステートメントは MEANS プロシジャと同じく、合計や平均を計算する数値タイプの変数(**分析変数**)を指定します。MEANS プロシジャでは省略可能でしたが、TABULATE プロシジャでは **TABLES ステートメントに登場する分析変数は必ず VAR ステートメントで指定しておく**必要があります。

TABLES ステートメントの中で以下の指定が登場しています。

変数名="ラベル"*f=フォーマット

="ラベル"

変数名や統計量の代わりに指定したラベルを使用するようにする指定です。
・列次元の分析変数名や統計量に対して**ブランクラベル（=”” または =” ”）**を指定した場合で、並列する列次元の統計量のラベル指定も同じく="" または =" "であった場合は、その行自体が削除されます。それ以外は表示がブランクになるだけで該当行または該当列は削除されません。
・変数名のラベルは**LABEL ステートメント**で指定しても表示できます。
・同様に統計量キーワードのラベルは**KEYLABEL ステートメント**でも指定できます。
・TABLES ステートメントにおいて指定すると、**同じ統計量キーワードに別々のラベルを与える**ことができます。

***f=フォーマット**
統計量キーワード、もしくは統計量キーワード="ラベル" 指定に続いて**列次元の統計量の表示フォーマット**を指定します。
f=w.d の形で、w は該当するセルの表示幅(縦罫線を除く表示列幅)、d は w の中での小数点の桁数指定です。**デフォルトのフォーマットは f=12.2** となっています。

・PROC TABULATE ステートメントの f=フォーマット オプション
PROC TABULATE ステートメントで f=フォーマットを指定すると、デフォルトフォーマットが指定したフォーマットに変わります。しかし、TABLES ステートメントに指定した各セルのフォーマット指定がその統計量のフォーマット出力として優先されます。

TABLES ステートメントのオプション
/ を指定してからオプションを指定します。

・RTS=行タイトルスペース
行次元の分類変数項目表示スペース幅をカラム数で指定します。この値は前後の"|"を含んだ幅の長さになります。デフォルトは LineSize の値の 1/4 に設定されています。(LS=132 の場合 RTS=33)

(プログラム 3.2-5)　列次元の分析変数と統計量の表示の削除

```
title "売上集計";
proc tabulate data=trans f=5.;
  class itemgrp sectgrp ym;
  var sales;
  tables itemgrp, (sectgrp ym)*sales=" "*sum=""*f=comma8./rts=12;
  label itemgrp="アイテムG" sectgrp="セクションG" ym="年月";
run;
```

(アウトプット)

売上集計

	セクションG		年月	
	KANAGAWA	TOKYO	2010-10	2010-11
アイテムG				
A	168	256	256	168
B	.	1,970	600	1,370

列次元の変数名に続く sales=""*sum="" の指定により、分析変数これらの行が削除されています。
列次元の (sectgrp ym) とカッコでくくることにより、2 つの分類変数を最上位で並列させる指定を簡潔

に表現しています。

(プログラム 3.2-6)　合計欄の追加

```
title;
proc tabulate data=trans f=5.;
  class itemgrp sectgrp ym;
  var sales length;
  tables itemgrp ALL, (ALL sectgrp ym)*sales=" "*sum=""*f=comma8./rts=12;
  label itemgrp="アイテムG" sectgrp="セクションG" ym="年月";
  keylabel all="全体";
run;
```

(アウトプット)

		セクションG		年月	
	全体	KANAGAWA	TOKYO	2010-10	2010-11
アイテムG					
A	424	168	256	256	168
B	1,970	.	1,970	600	1,370
全体	2,394	168	2,226	856	1,538

(プログラム 3.2-7)　小計と合計

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp ym;
  var sales length;
  tables itemgrp ALL, (ALL sectgrp)*(ym ALL="合計")*sales=" "
    *sum=""*f=comma8./rts=12;
  tables itemgrp ALL, (ALL*(ym ALL="合計") sectgrp*(ym ALL="小計"))*sales=" "
    *sum=""*f=comma8./rts=12;
  label itemgrp="アイテムG" sectgrp="セクションG" ym="年月";
  keylabel all="全体";
run;
```

(アウトプット)

				セクションG				
	全体			KANAGAWA		TOKYO		
	年月			年月		年月		
	2010-10	2010-11	合計	2010-11	合計	2010-10	2010-11	合計
アイテムG								
A	256	168	424	168	168	256	.	256
B	600	1,370	1,970	.	.	600	1,370	1,970
全体	856	1,538	2,394	168	168	856	1,370	2,226

	全体			セクションG				
				KANAGAWA		TOKYO		
	年月			年月		年月		
	2010-10	2010-11	合計	2010-11	小計	2010-10	2010-11	小計
アイテムG								
A	256	168	424	168	168	256	.	256
B	600	1,370	1,970	.	.	600	1,370	1,970
全体	856	1,538	2,394	168	168	856	1,370	2,226

(プログラム 3.2-8)　百分率の計算

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp;
  tables itemgrp ALL, (sectgrp ALL)*(N PCTN*f=5.2) /rts=12;
  tables itemgrp ALL, (sectgrp ALL)*(N PCTN<itemgrp*sectgrp itemgrp*ALL ALL*sectgrp
ALL*ALL>*f=6.2) /rts=20;
  label itemgrp="アイテムG" sectgrp="セクションG";
  keylabel all="全体" N="件数" PCTN="件数百分率";
run;
```

(アウトプット)

	セクションG					
	KANAGAWA		TOKYO		全体	
	件数	件数百分率	件数	件数百分率	件数	件数百分率
アイテムG						
A	2	33.33	1	16.67	3	50.00
B	.	.	3	50.00	3	50.00
全体	2	33.33	4	66.67	6	100.0

	セクションG					
	KANAGAWA		TOKYO		全体	
	件数	件数百分率	件数	件数百分率	件数	件数百分率
アイテムG						
A	2	33.33	1	16.67	3	50.00
B	.	.	3	50.00	3	50.00
全体	2	33.33	4	66.67	6	100.00

PCTN 統計量キーワードは以下のように指定します。

PCTN<分母>

<分母>は省略すると、**その分類組み合わせ定義のレベルの合計** という意味になります。

1 番目の TABLES ステートメントの場合は、PCTN の<分母>指定を省略しています。この TABLES 指定を展開すると、

itemgrp*sectgrp*PCTN
itemgrp*ALL*PCTN
ALL*sectgrp*PCTN
ALL*ALL*PCTN

の 4 つの部分に展開されます。それぞれの部分における<分母>は上記の*PCTN の左側になっているという意味です。
これらの分類組み合わせの合計件数は、いずれも全体件数の 6 となりますので、全体件数を分母とする件数百分率を計算することを意味しています。

2 番目の TABLES ステートメントの PCTN の指定は分布の指定を明示したものです。結果が同じになっていることを確認してください。

(プログラム 3.2-9)　行百分率の計算

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp;
  tables itemgrp ALL, (sectgrp ALL)*(N PCTN<sectgrp ALL sectgrp ALL*ALL>*f=6.2) /rts=20;
  label itemgrp="アイテムG" sectgrp="セクションG";
  keylabel all="全体" N="件数" PCTN="行百分率";
run;
```

(アウトプット)

	セクションG				全体	
	KANAGAWA		TOKYO			
	件数	行百分率	件数	行百分率	件数	行百分率
アイテムG						
A	2	66.67	1	33.33	3	100.00
B	.	.	3	100.00	3	100.00
全体	2	33.33	4	66.67	6	100.00

TABLES ステートメントの PCTN キーワードの<分母>指定を以下のように変えています。

itemgrp*sectgrp*PCTN　に対しては <sectgrp>　各 itemgrp 値ごとに分母は sectgrp の値を全部足した値
itemgrp*ALL*PCTN　に対しては <ALL>　各 itemgrp 値ごとに分母は sectgrp の ALL 値
ALL*sectgrp*PCTN　に対しては <sectgrp>　itemgrp の ALL 値の分母は sectgrp の値を全部足した値
ALL*ALL*PCTN　に対しては <ALL+ALL>　全体合計値の分母はその全体合計値

すべての PCTN のセルで行百分率を計算することになります。

(プログラム 3.2-10)　PCTSUM

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp ym;
  var sales length;
  tables itemgrp ALL, (ALL sectgrp ym)*sales=" "
    *(sum="売上高"*f=comma8.  PCTSUM*f=6.1)/rts=12;
```

```
  tables itemgrp ALL, (ALL sectgrp ym)*sales=" "
    *(sum="売上高"*f=comma8.
   PCTSUM<itemgrp*ALL itemgrp*sectgrp itemgrp*ym
          ALL*ALL ALL*sectgrp ALL*ym>
    *f=8.1)/rts=20;

  label itemgrp="アイテムG" sectgrp="セクションG" ym="年月";
  keylabel all="全体" PCTSUM="構成比率%";
run;
```

(アウトプット)

	全体		セクションG				年月			
			KANAGAWA		TOKYO		2010-10		2010-11	
	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%
アイテムG										
A	424	17.7	168	7.0	256	10.7	256	10.7	168	7.0
B	1,970	82.3	.	.	1,970	82.3	600	25.1	1,370	57.2
全体	2,394	100.0	168	7.0	2,226	93.0	856	35.8	1,538	64.2

	全体		セクションG				年月			
			KANAGAWA		TOKYO		2010-10		2010-11	
	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%
アイテムG										
A	424	17.7	168	7.0	256	10.7	256	10.7	168	7.0
B	1,970	82.3	.	.	1,970	82.3	600	25.1	1,370	57.2
全体	2,394	100.0	168	7.0	2,226	93.0	856	35.8	1,538	64.2

PCTN と同様、PCTSUM の<分母>のデフォルトは全体です。

TABLES 指定を展開してみましょう。

itemgrp*ALL*sales*PCTSUM
itemgrp*sectgrp*sales*PCTSUM
itemgrp*ym*sales*PCTSUM
ALL*ALL*sales*PCTSUM
ALL*sectgrp*sales*PCTSUM
ALL*ym*sales*PCTSUM

以上の 6 つの部分に分かれることがわかります。

では、上記アウトプットの構成比率をすべて列百分率の表示に変えるプログラムを書いてください。

```
  tables itemgrp ALL, (ALL sectgrp ym)*sales=" "
    *(sum="売上高"*f=comma8.
```

```
  PCTSUM<itemgrp*ALL itemgrp*sectgrp itemgrp*ym
         ALL*ALL ALL*sectgrp ALL*ym>
   *f=8.1)/rts=20;
```

の PCTSUM<>を書き換えてください。

(こんなアウトプット)

	全体		セクションG				年月			
			KANAGAWA		TOKYO		2010-10		2010-11	
	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%	売上高	構成比率%
アイテムG										
A	424	17.7	168	100.0	256	11.5	256	29.9	168	10.9
B	1,970	82.3	.	.	1,970	88.5	600	70.1	1,370	89.1
全体	2,394	100.0	168	100.0	2,226	100.0	856	100.0	1,538	100.0

[分母の定義を省略した行百分率、列百分率の指定]

統計量キーワード ROWPCTN, COLPCTN, ROWPCTSUM, COLPCTSUM を使用すると分母の指定が不要になります。これらを指定した場合は、行または列の総合計欄の値が、表示されているいないに関わらず、分母に指定されたものとみなされます。

(プログラム 3.2-9') ROWPCTN キーワードを用いた行百分率の計算

```
proc tabulate data=trans f=5.;
  class itemgrp sectgrp;
  tables itemgrp ALL, (sectgrp ALL)*(N ROWPCTN*f=6.2) /rts=20;
  label itemgrp="アイテムG" sectgrp="セクションG";
  keylabel all="全体" N="件数" ROWPCTN="行百分率";
run;
```

(アウトプット)

	セクションG					
	KANAGAWA		TOKYO		全体	
	件数	行百分率	件数	行百分率	件数	行百分率
アイテムG						
A	2	66.67	1	33.33	3	100.00
B	.	.	3	100.00	3	100.00
全体	2	33.33	4	66.67	6	100.00

[数値変数を分母の定義に使用した百分率の例]

通常、百分率を表示する場合は、CLASS ステートメントに指定した分類変数（の交差定義）を分母に指定しますが、百分率を表示したいセルに対する分類変数による交差定義が指定されていない場合は、百分率は計算できません。以下の例は、数値変数としてアンケート回答者 ID を VAR ステートメントに指定しておき、q2_1, q2_2, q2_2 の 3 つの多重回答項目について、合計件数の回答者人数に対する百分率を集計しています。ID を分母に指定することにより、顧客 ID の種類数の合計件数（この場合は 101～106 の

6 件）が分母に用いられます。なお、多肢選択項目 q1 は CLASS 変数に指定し、通常の集計を行っています。

(プログラム 3.2-11)　多重回答アンケートの集計

```
data mult;
  input ID q1 q2_1 q2_2 q2_3;
cards;
101 1 1 1 .
102 2 1 1 1
103 1 . . 1
104 3 . 1 1
105 1 . . 1
106 .A 1 1 1
;
proc tabulate data=mult missing;
  class q1;
  var q2_1 q2_2 q2_3 ID;
  tables q1 q2_1 q2_2 q2_3,(n="件数" pctn<q1 ID ID ID >="構成比率%"*f=5.1)/rts=6;
run;
```

(アウトプット)

	件数	構成比率%
q1		
A	1.00	16.7
1	3.00	50.0
2	1.00	16.7
3	1.00	16.7
q2_1	3.00	50.0
q2_2	4.00	66.7
q2_3	5.00	83.3

以上で TABULATE は終了です。

[SAS マクロ機能]

Base SAS プロダクトに含まれる機能で、プログラムのモジュール化やアプリケーション開発に威力を発揮します。**DATA ステップや PROC ステップの上位に位置する機能**で、通常の SAS ステートメントより前に解釈実行されます。マクロ機能はプログラミング言語仕様を持つマクロ言語とマクロ言語で書かれた SAS コード部分を解釈実行するマクロプロセサの 2 つの要素で構成されます。主なマクロ機能として、マクロ定義（マクロ保存、マクロ呼出しを含む）、グローバルマクロ変数、自動マクロ変数、DATA ステップとのインターフェースなどがあります。

マクロの利用により、以下のようなプログラムコーディング上のメリットが得られます。

-同じ処理を繰り返し行う場合のコーディングの簡素化
-再帰処理
-処理のモジュール化
-アプリケーション作成

(プログラム 3.2-12)　%LET ステートメントとマクロ変数

```
options symbolgen;
%let pgm1=%str(data a;a=1;run;proc print data=a;run;);
&pgm1;
options nosymbolgen;
```

(ログ)

```
554  options symbolgen;
555  %let pgm1=%str(data a;a=1;run;proc print data=a;run;);
556  &pgm1;
SYMBOLGEN:  マクロ変数 PGM1 を data a;a=1;run;proc print data=a;run; に展開します。
SYMBOLGEN: 値のマクロ引用した文字のうち、 プリントするために引用符を取り除いたものがあります。
```

(アウトプット)

```
OBS    a

 1     1
```

%LET ステートメントは**マクロ変数**に値を定義します。マクロ変数は作成した後、SAS セッションのどこでも参照できます。マクロ機能(%か&で始まるワード)は DATA ステップや PROC ステップの実行を管理する SAS プロセサより前に認識され実行されます。この場合、参照されたマクロ変数の値（テキスト）は SAS プログラムとして SAS プロセサに認識され、実行されます。

%LET マクロ変数名=値;

マクロ変数の値は常にテキストです。引用符("")はそのままテキストとして認識されます。

(プログラム 3.2-13)　マクロ定義

```
options mprint;
%macro a;
  data _null_;
    age=int(input(put(today(),yymmddn8.),8.)/10000-&birth/10000);
    put age=;
  run;
%mend a;

%let birth=20101019;
%a
```

(ログ)

```
578  options mprint;
579  %macro a;
580    data _null_;
581      age=int(input(put(today(),yymmddn8.),8.)/10000-&birth/10000);
582      put age=;
583    run;
584  %mend a;
585
586  %let birth=20101019;
587  %a
MPRINT(A):   data _null_;
SYMBOLGEN:  マクロ変数 BIRTH を 20101019 に展開します。
```

```
MPRINT(A):   age=int(input(put(today(),yymmddn8.),8.)/10000-20101019/10000);
MPRINT(A):   put age=;
MPRINT(A):   run;

age=0
```

%で始まるステートメントはマクロステートメントです。
%MACRO ステートメントは**マクロ定義**を開始するステートメントで、**%MEND ステートメント**で定義を終了します。

%MACRO マクロ名(パラメータ) / オプション;

マクロ名は SAS 名の命名規則に準拠します。
この例ではマクロ名は a で、**パラメータ**は定義されていません。
マクロ a の処理は、yyyymmdd のフォーマット形式の文字列値が入ったマクロ変数 &birth の値を外部から受け取り、今日現在の年齢を DATA ステップで計算し結果をログに書き出すというものです。

%LET ステートメントでマクロ変数 birth に 20101019 を代入しています。
そして %a という指定は、

%マクロ名

という文法で 指定した**マクロを呼び出す**ものです。
このステートメントは**最後のセミコロンが不要**です。もしも指定すると、それは%マクロ名の指定部分で定義されたマクロを呼び出し SAS 言語に展開されたステートメントが SAS プロセサに送り込まれた後に、さらに 1 個のセミコロンが送り込まれサブミットされることを意味します。
この例のように、大抵の場合はマクロ定義の最後は RUN;といったステートメントで終了しているので、もう 1 個セミコロンがあっても結果に影響ありません。

(プログラム 3.2-14)　マクロ定義のパラメータ

```
%macro a(birth,format=best12.);
  data _null_;
    age=int(input(put(today(),yymmddn8.),8.)/10000-&birth/10000);
    put age= &format;
  run;
%mend a;

%a(20101019,format=z8.)
```

(ログ)

```
604  %macro a(birth,format=best12.);
605    data _null_;
606      age=int(input(put(today(),yymmddn8.),8.)/10000-&birth/10000);
607      put age= &format;
608    run;
609  %mend a;
610
611  %a(20101019,format=z8.)
MPRINT(A):   data _null_;
SYMBOLGEN:  マクロ変数 BIRTH を 20101019 に展開します。
```

```
MPRINT(A):   age=int(input(put(today(),yymmddn8.),8.)/10000-20101019/10000);
SYMBOLGEN:  マクロ変数 FORMAT を z8. に展開します。
MPRINT(A):   put age= z8.;
MPRINT(A):   run;

age=00000000
```

マクロ定義でパラメータを指定しておくと、パラメータ値を指定してマクロ呼出を行えます。
パラメータの指定方法には、**定位置パラメータ**と**キーワードパラメータ**の 2 通りの指定ができます。

(定位置パラメータ)
そのマクロ定義で用いているマクロ変数名をカンマで区切って指定します。初期値を持つことができません。マクロを呼び出すときは、必ず、定義された順にカンマで区切ってパラメータ値を指定します。キーワードパラメータより先に定義しておかなくてはなりませんし、呼び出すときも先に値を指定しなければなりません。

この例では マクロ変数 birth は定位置パラメータとして定義しています。

(キーワードパラメータ)
マクロ変数名=初期値　の形式で指定します。このパラメータは初期値を与えていますので、呼出時に指定しなくてもかまいません。

この例では マクロ変数 format を 初期値 best12. で定義しています。

(プログラム 3.2-15)　処理の分岐

```
%macro a(birth);
  data _null_;
    age=int(input(put(today(),yymmddn8.),8.)/10000-&birth/10000);
    call symput("AGE",left(put(age,best12.)));
  run;
  %put AGE=&AGE;
  %if &AGE<0 %then %put %str(入力されたbirthの値は未来の年月日を表しています.);
  %else %put %str(あなたの年齢は %trim(&AGE) 歳です.);
%mend a;

options nomprint nosymbolgen;
%a(20001019)
%a(20201019)
```

(ログ)

```
770  %a(20001019)

NOTE: DATA ステートメント 処理 (合計処理時間):
      処理時間                0.00 秒
      CPU 時間                0.00 秒

AGE=10
あなたの年齢は 10 歳です.
771  %a(20201019)

NOTE: DATA ステートメント 処理 (合計処理時間):
```

```
      処理時間              0.00 秒
      CPU 時間              0.00 秒

AGE=-9
入力されたbirthの値は未来の年月日を表しています.
```

マクロ言語は、%IF ステートメントや%trim 関数など DATA ステップのステートメントや関数と同じようなステートメントや関数を備えています。

(プログラム 3.2-16)　プログラムのモジュール化

```
%macro NOBS(data);
  %global NOBS;
  data _null_;
    call symput("NOBS",compress(put(nobs,best12.)));
    stop;
    set &data nobs=nobs;
  run;
%mend NOBS;

%NOBS(trans)
%put &NOBS;
```

(ログ)

```
754   %NOBS(trans)
MACROGEN(NOBS):   data _null_;
MACROGEN(NOBS):   call symput("NOBS",compress(put(nobs,best12.)));
MACROGEN(NOBS):   stop;
MACROGEN(NOBS):   set trans nobs=nobs;
MACROGEN(NOBS):   run;

NOTE: DATA ステートメント 処理 (合計処理時間):
      処理時間              0.00 秒
      CPU 時間              0.00 秒

755   %put &NOBS;
6
```

(プログラム 3.2-17)　アプリケーション開発

```
%macro NOBS(data);
  %*global NOBS;
  data _null_;
    call symput("NOBS",compress(put(nobs,best12.)));
    stop;
    set &data nobs=nobs;
  run;
%mend NOBS;
%macro means1(DATA);
  %local NOBS;
  %let dsid=%sysfunc(open(&DATA));
  %if &dsid<=0 %then %do;
    %put %str(データセット &data が見つかりません.);
    %goto eee;
  %end;
  %let rc=%sysfunc(close(&dsid));
```

```
  %NOBS(&DATA)
  %if &NOBS=0 %then %do;
    %put %str(データセット%DATA にはオブザベーションがありません.);
    %goto eee;
  %end;
  proc means data=&DATA;run;
  %eee:
%mend means1;

%symdel NOBS;

%means1(sample)
%means1(trans)
```

(ログ)

```
807  %means1(sample)
データセット sample が見つかりません.
808  %means1(trans)
NOTE: DATA ステートメント 処理 (合計処理時間):
      処理時間                0.00 秒
      CPU 時間                0.00 秒

NOTE: データセット WORK.TRANS から 6 オブザベーションを読み込みました。
NOTE: PROCEDURE MEANS 処理 (合計処理時間):
      処理時間                0.01 秒
      CPU 時間                0.01 秒
```

(アウトプット)

変数	N	平均	標準偏差	最小値	最大値
date	6	18568.17	25.0871813	18536.00	18596.00
sales	6	399.0000000	309.9625784	45.0000000	850.0000000
length	6	6.0000000	1.5491933	5.0000000	8.0000000

(終了)

[別表 1] DATA ステップで使えるステートメント一覧

ステートメント名	役割
ABORT	DATAステップの実行を中断する
ARRAY	変数配列の宣言
asign（割り当て）	変数名=式;の形で指定する。 等号の左辺の変数に右辺の式の値を割り当てる
ATTRIB	1つの変数の属性（タイプ・長さ・フォーマット・インフォーマット・ラベル）をまとめて宣言する
BY	指定の変数のソート順にオブザベーションが並んでいることを示す。 BYグループ処理を行う場合に必須となる
CALL	CALLルーティン（複数の戻り値を許す関数）の呼び出し
CARDS	これ以降にカードイメージデータが記述されていることを示す
CARDS4	同　セミコロンや2バイト文字を含むデータを正しく読み取る場合にCARDSに代えて用いる
CONTINUE	DOループ（DOグループ）処理の中で用い、ENDステートメントまで強制移動させてDOループ処理にとどまることを指示する
DATA	DATAステップの開始とこのステップで作成する出力データセット名を宣言する
DATALINES	CARDSステートメントの別名
DATALINES4	CARDS4ステートメントの別名
DELETE	現在処理中のオブザベーションの処理を中断（出力データセットに書き込まない）して、次のオブザベーションの処理に移るためにDATAステップのはじめに戻る
DO	ENDステートメントと対で用い、条件式に合致した場合の実行範囲をDO～ENDで囲んで指定する。 囲まれた範囲をDOループまたはDOグループと呼ぶ
DO, iterative（繰り返しDO）	繰り返しDOステートメントの1つで、iterativeの部分には　変数名=開始値　TO　終了値　BY　増分値というDOループ処理の実行条件指定が入る
DO UNTIL	同　UNTIL(条件式)に指定した条件を満たさない範囲でDOループ処理を実行する
DO WHILE	同　WHILE（条件式）に指定した条件を満たしている範囲でDOループ処理を実行する
DROP	出力データセットに含めない変数を指定する
ELSE	IF～THENステートメントと共に用い、IF条件に合致しない場合の処理を記述する
END	DOステートメントと対で用い、DOループ処理範囲を指定する
ERROR	強制的にエラーを発生させる
FILE	データ値の出力先（外部ファイル名、リスティング、ログなど）を指定する
FORMAT	指定の変数に出力フォーマットを指定する
GO TO	指定のラベル名が書かれたプログラム位置に次の処理を強制移動させる
IF, Subsetting（サブセットIF）	指定の条件に合致するオブザベーションのみこれ以降の処理に進むことを許可する
IF～THEN	指定の条件に合致する場合の処理を記述する。 条件に合致しない場合の処理は続くELSEステートメントで記述する
INFILE	外部入力ファイル名を指定する
INFORMAT	指定の変数に入力フォーマットを指定する
INPUT	外部ファイルから指定の変数名の値を指定の入力形式で読み取る
KEEP	出力データセットに含める変数を指定する
LABEL	指定の変数に変数ラベルを定義する
Labels, Statement	ラベル名：（コロン）の指定により、GO TOやLINKステートメントにより強制移動させるプログラム位置を示す
LEAVE	DOループ（DOグループ）処理の中で用い、ENDステートメントの次のステートメントまで（ラベルを指定していた場合はラベル位置まで）強制移動させてDOループ処理を抜けることを指示する
LENGTH	作成する変数のタイプと長さを定義する。
LINK	指定のラベル名が書かれたプログラム位置からRETURNステートメントまでの範囲に記述されたサブルーティンに処理を強制移動させた後、移動前の位置に戻るよう指示する
LIST	変数の値をログに書き出す
MERGE	複数のデータセットを横に結合した形でオープンする
OUTPUT	指定の出力データセットに現在処理中のオブザベーションを書き出す
PAGE	改ページを指示する。 DATAステッププログラミングによるレポート作成用ステートメント

PUT	外部ファイルやリスティング出力やログへ指定の変数名の値を指定の出力形式で書き出す
PUTLOG	FILEステートメントの指定する書き出し先に無関係に、ログにメッセージを書き出す
RENAME	変数名を変更する
RETAIN	指定の変数値の現在値を次のオブザベーション処理に変わっても初期化せずに保持するよう宣言する
RETURN	最初のDATAステートメントに処理を戻す。 LINKステートメントからの分岐の場合はLINKの次のステートメントに処理を戻す
RUN	DATAステップの記述の終了を明示的に指定する
SELECT	条件選択のために条件を指定する
SET	データセットを入力のためにオープンする。 複数のデータセットを指定した場合MERGEと異なり縦に結合したイメージでオープンする
SKIP	ブランク行を書き出す。 DATAステッププログラミングによるレポート作成用ステートメント
STOP	DATAステップの処理を中止する
Sum	変数名+式の形で指定する。DATAステップのループ処理中の変数値は右辺の式の値の累積値を値として持つ
UPDATE	UPDATE Master Transact;の形の指定となり必ずBYステートメントと共に指定する。 Masterデータセットの値をTransactデータセットの値で更新する場合に用いる

[別表 2] PROC ステップの種類

プロシジャ名	役割
PROC APPEND	データセット最後のオブザベーションの後に他のデータセットのオブザベーションを追加する
PROC COMPARE	2つのデータセットの内容を比較する
PROC CONTENTS	データセットのコンテンツ情報（オブザベーション数などの一般属性と変数名や変数タイプなどの変数属性）を表示したりデータセットに出力する
PROC COPY	データセットをコピーする
PROC CORR	相関係数を計算する
PROC DATASETS	特定のライブラリに格納されているデータセット名のリストを表示したり、個々のデータセットの名前の変更・削除などを行う。 また、個々のデータセットに関する属性の表示や編集を行う
PROC DELETE	上記DATASETSプロシジャの機能の一部であるデータセットの削除を行う
PROC EXPORT	データセットを外部ファイル形式に変換する。 IMPORTプロシジャの逆の操作を行う
PROC FORMAT	ユーザー定義フォーマットを作成する
PROC FREQ	度数集計を行う。 n次元クロス集計も可能
PROC IMPORT	特定の形式（CSV形式など）の外部ファイルからデータを読み取りデータセットに変換する
PROC MEANS	変数ごとの基本統計（平均値、標準偏差など）を計算する
PROC OPTIONS	オプション設定を変更する
PROC PRINT	データセットの値をリスト表示する
PROC PRINTTO	リスティング出力の出力先をファイルに変更する
PROC RANK	変数値の順序を計算する
PROC SCORE	スコア係数とデータ値の積和によるスコアを計算する
PROC SORT	オブザベーションを指定の変数値の順に並び替える
PROC SQL	SQL言語によるデータ検索・加工を行う
PROC SUMMARY	MEANSと同じく変数ごとの基本統計（平均値、標準偏差など）を計算する
PROC TABULATE	度数・平均・百分率を含む多重クロス集計表を作成する
PROC TRANSPOSE	データセットを転置（行と列を交換）する
PROC UNIVARIATE	変数ごとの基本統計（平均値、標準偏差など）を計算する。 MEANS,SUMMARYより詳細

[別表 3] グローバルステートメント一覧

ステートメント名	役割
Comment（コメント）	*（アスタリスク記号）で始まるステートメント。 任意のコメントを記述できる
ENDSAS	WPSセッションを終了する
FILENAME	外部ファイル参照名を定義する

FILENAME_DDE	DDE（動的データ交換）機能によるアプリケーションとWPS間のデータ交換を行う
FILENAME_EMAIL	WPSからEmailを送る
FOOTNOTE	フットノートテキストを定義する
%INC	%INCステートメントの省略形
%INCLUDE	外部ファイルに書かれたソースコードを読み込み実行する
LIBNAME	データセットライブラリ参照名を定義する
MISSING	数値タイプ変数の入力値に書かれた欠損を表す文字を指定する。
ODS EXCLUDE	ODS機能による選択リストの中から除外項目を選ぶ
ODS HTML	HTML出力を管理する。 また、どのオブジェクトをHTML出力するかを制御する
ODS LISTING	特定項目のリスティング出力を管理する
ODS OUTPUT	特定項目のデータセット出力を管理する
ODS SELECT	ODS機能による選択リストの中から選択項目を選ぶ
ODS SHOW	ODS選択リストの表示
ODS TRACE	ODS出力に関するメッセージをログに書き出すかどうかを切り替える
OPTIONS	各種オプションの設定を変更する
RUN	DATAステップ、PROCステップのステートメントの指定を終了しステップを実行する
TITLE	タイトルテキストを定義する
PAGE	ログを新しいページに切り替える
SKIP	ログにブランク行を1行書いて改行する
X	コマンドプロンプトを呼び出す

[別表 4] 演算子一覧

分類	シンボル	別表記	意味	例
算術	+		足し算	c=a+b
	−		引き算	d=10−x
	*		掛け算	y=2*x
	/		割り算	z=x/y
	**		累乗計算	value=2**(x+1)
論理	&	AND	かつ	if (a=1) & (b=10)
	|	OR	または	if (a=1) | (b=10)
	^	NOT	否定	if ^(z=1)
比較	=	EQ	等しい	if a=b
	^=	NE	等しくない	if a NE b
	<	LT	より小さい（未満）	if a<b
	<=	LE	等しいかより小さい（以下）	if a LE 5
	>	GT	より大きい（超）	if b GT 15*x
	>=	GE	等しいかより大きい（以上）	if b>=16*x
符号	+		プラス（正の数）	y=+1
	−		マイナス（負の数）	y=−1
最小	><		小さい方の値	z=(x><y)
最大	<>		大きい方の値	H=(x<>y<>z)
文字列検索		IN	いずれかの文字列に一致する	if name in (″abc″ ″de″)
文字列連結	||		左右の文字列をつなぐ	z=″ABC″||″DEFG″||″HI″
文字列比較	:		比較演算子と共に用いる。 左右の文字列の長さを短い方に揃えてから比較する	a=″12345″; b=″123″; if a=:b （真）

[別表 5] 関数一覧

分類	関数名	意味	例
三角	ARCOS	アークコサイン	y=arcos(x);
	ARSIN	アークサイン	y=arsin(x);

	ATAN	アークタンジェント	y=atan(x);
	COS	コサイン	y=cos(x);
	COSH	ハイパボリックコサイン	y=cosh(x);
	SIN	サイン	y=sin(x);
	SINH	ハイパボリックサイン	y=sinh(x);
	TAN	タンジェント	y=tan(x);
	TANH	ハイパボリックタンジェント	y=tanh(x);
数学	ABS	絶対値	y=abs(x);
	EXP	指数	y=exp(x);
	LOG	自然対数（底=e）	y=log(x);
	LOG10	常用対数（底=10）	y=log10(x);
	LOG2	2を底とする対数	y=log2(x);
	MOD	割り算の余りを返す	y=mod(x,100);
	POW	累乗。**演算子と同じ	x=pow(100,2);
	SIGN	符号を返す	s=sign(-156);
	SQRT	平方根	y=sqrt(x);
数値丸め	CEIL	整数値に切り上げ	y=ceil(x);
	FLOOR	整数値に切り捨て	y=floor(x);
	FUZZ	最も近い整数値との差が1E-12以内であればその整数値を返す	x=fuzz(x);
	INT	整数部分を取り出す	x_int=int(x);
	ROUND	四捨五入して指定の桁位置に丸める	x=round(125.321,0.1);
	ROUNDZ	四捨五入して指定の桁位置に丸める、fuzzing処理を行わない	x=roundz(125.321,0.1);
統計	CSS	修正済平方和	x=css(5,10,20,16,0,5);
	CV	変動係数（%表示）	x=cv(5,10,20,16,0,5);
	KURTOSIS	尖度	x=kurtosis(5,10,20,16,0,5);
	MAX	最大値	x=max(5,10,20,16,0,5);
	MEAN	平均値	x=mean(5,10,20,16,0,5);
	MIN	最小値	x=min(5,10,20,16,0,5);
	N	非欠損値の数を返す	n=n(1,3,.,5,10);
	NMISS	欠損値の数を返す	nmiss=nmiss(1,3,.,5,10);
	RANGE	範囲	x=range(5,10,20,16,0,5);
	SKEWNESS	歪度	x=skewness(5,10,20,16,0,5);
	STD	標準偏差	x=std(5,10,20,16,0,5);
	SUM	合計	x=std(5,10,20,16,0,5);
	USS	修正前平方和	x=uss(5,10,20,16,0,5);
	VAR	不偏分散	x=var(5,10,20,16,0,5);
配列	DIM	配列の要素数を返す	do i=1 to dim(z);
	HBOUND	定義された配列の最後の要素の参照番号を返す	do i=lbound(arrayname) to hbound(arrayname);
	LBOUND	定義された配列の最初の要素の参照番号を返す	do i=lbound(arrayname) to hbound(arrayname);
日付と時間	DATE	今日の日付を返す（1960年1月1日を起点とした経過日数）、TODAYと同じ	today=date();
	DATEJUL	ジュリアン日付値表示から標準の日付値に変換	d=datejul(2008366);
	DATEPART	日時値から日付値部分を取り出す	date=datepart("01JAN2008:12:10:00"DT);
	DATETIME	現在の日時値を返す（1960年1月1日を起点とした経過秒数）	now=datetime();
	DAY	日付値または日時値から日部分を取り出す	day=day("10AUG2008"D);
	DHMS	日付、時、分、秒から日時値を作成	val=dhms("10AUG2008"D,12,10,30);
	HMS	時、分、秒から時間値を作成	time=hms(12,0,0);
	HOUR	時間値または日時値から時間部分を取り出す	h=hour("01JAN2008:12:10:00"DT);
	INTCK	開始時点から終了時点までの経過	keika_month=intck("month","10JAN2008"D,"25AUG2008"D);

		時間をさまざまな時間単位で計算	
	INTNX	指定の時間経過後の時点を返す	after=intnx("month","10JAN2008"D,3,"END");
	JULDATE	日付値を5桁のジュリアン日付値形式(yyddd)に変換	juldate=juldate("01JAN2008"D);
	JULDATE7	日付値を7桁のジュリアン日付値形式(yyyyddd)に変換	juldate=juldate7("01JAN2008"D);
	MDY	月、日、年から日付値を作成	date1=mdy(12,31,2007);
	MINUTE	時間値または日時値から分部分を取り出す	m=minute("01JAN2008:12:10:00"DT);
	MONTH	日付値または日時値から分部分を取り出す	month=month("01JAN2008:12:10:00"DT);
	QTR	日付値または日時値から四半期部分を取り出す	qtr=qtr("10AUG2008"D);
	SECOND	時間値または日時値から秒部分を取り出す	s=second("01JAN2008:12:10:00"DT);
	TIME	現在の時間値を返す	now=time();
	TIMEPART	日時値から時間部分を取り出す	time=timepart("01JAN2008:12:10:00"DT);
	TODAY	今日の日付を返す(1960年1月1日を起点とした経過日数)、DATEの別名	today=today();
	WEEKDAY	日付値または日時値から曜日を取り出す	week=weekday("10AUG2008"D);
	YEAR	日付値または日時値から年部分を取り出す	year=year("10AUG2008"D);
	YYQ	年、四半期から日付値を作成	yyq=yyq(2008,1);
ビット演算	BAND	ビットのAND	x=band(9Fx,11x);
	BLSHIFT	ビットを左シフトする	x=blshift(01x,1);
	BNOT	ビットのNOT	x=bnot(01x);
	BOR	ビットのOR	x=bor(9fx,90x);
	BRSHIFT	ビットを右シフトする	x=brshift(01x,31);
	BXOR	ビットのXOR	bxor(01x,55x);
マクロ	CALL EXECUTE	DATAステップの中から実行ルーティンを呼び出す	if x=1 then call execute("proc print;run;");
	CALL SYMDEL	グローバルマクロ変数を削除	(動かない)
	CALL SYMPUT	DATAステップ変数値をマクロ変数値に割り当てる	call symput("mvar",char);
	SYMGET	マクロ変数値をDATAステップ変数値に変換	c=symget("c");
文字	BYTE	ASCII文字を返す、RANKの逆	char=byte(40x);
	CAT	文字列を連結する	List=cat(a,b):
	CATS	前後のブランクを詰めてから文字列を連結する	List_space=cats(a,b);
	CATT	後ろのブランクを詰めてから文字列を連結する	List_trim=catt(a,b);
	COMPBL	連続するブランクを1個に圧縮する	char=compbl(a||" "||b);
	COMPRESS	指定の文字(デフォルトはブランク)を除外する	char=compress(a||b);
	CONTAINS	指定の部分文字列の有無をチェックする	check=contains(c,"abc");
	INDEX	文字値から指定の文字列の開始位置を返す	position=index("abcabdefgh","bde");
	INDEXC	文字値から指定のいずれかの文字の開始位置を返す	position=indexc("abcabdefgh","bde");
	INDEXW	文字値から指定のワードの開始位置を返す	position=indexw("abc,abde,fgh","abde",",");
	LEFT	文字値を左詰する	char=left(" abc ");
	LENGTH	文字値の長さを返す	len=length(compress(x));
	LIKE	正規表現のあいまい検索	if like(c1,"_ABC%") then put "OK";
	LOWCASE	小文字変換	low=lowcase("ABC");

	MAXC	ブランクを除く最大の文字値	maxchar=maxc("Z1","ABC");
	MINC	ブランクを除く最小の文字値	minchar=minc("Z1","ABC");
	PROPCASE	特殊文字をデリミタとして語単位に先頭は大文字化、残りの文字は小文字化する	c=propcase("new/software@world","/@ ");
	RANK	1文字値のシーケンス番号を返す、BYTEの逆	seq=rank("z");
	REPEAT	1文字の繰り返し文字列を作成	char=repeat("z",10);
	REVERSE	文字値を逆順に並べ替える	rev_c=reverse(c);
	RIGHT	文字値を右詰する	char=right("　　abc　");
	SCAN	区切り文字で区切られたn番目の文字列を抽出	c=scan("new/software@world",2,"/@ ");
	SUBSTR	部分文字列の抽出	c=substr("abcdefg",3,2);
	TRANSLATE	特定の文字を別の文字に変換する	new=translate("abcdefgfcded","150","ceg");
	TRANWRD	特定の文字列を別の文字列に変換する	new=tranwrd("abcdefgfcded","cd","99");
	TRIM	文字値の後ろ側のブランクを削除する	c=trim(a)||trim(b);
	TRIMN	欠損値に対して長さ0の文字値を返す以外はTRIMと同じ	c=trimn(a);
	UPCASE	大文字変換	up=upcase("Abc");
	VERIFY	文字値が指定の文字のみ含むかどうかをチェック	chk=verify("abcdefgfcded","abcdefg");
DBCS関数	KCOMPRESS	指定の文字（デフォルトはブランク）を除外する	char=kcompress(a||b);
	KINDEX	文字値から指定の文字列の開始位置を返す	position=kindex("abcあいうefgh","うe");
	KINDEXC	文字値から指定のいずれかの文字の開始位置を返す	position=kindexc("abcあいうefgh","bうe");
	KLEFT	前にある全角と半角のブランク文字を削除して文字列を左詰する	char=kleft("　　abc　");
	KLENGTH	文字値の長さを返す	len=klength(kcompress(x));
	KLOWCASE	小文字変換	low=klowcase(a);
	KREVERSE	文字値を逆順に並べ替える	rev_c=kreverse(c);
	KRIGHT	後にある全角と半角のブランク文字を削除して文字値を右詰する	char=kright("　　abc　");
	KSCAN	区切り文字で区切られたn番目の文字列を抽出	c=kscan("new/ソフト@world",2,"/@ ");
	KSUBSTR	部分文字列の抽出	c=ksubstr("abcあいうdefg",4,3);
	KTRANSLATE	特定の文字を別の文字に変換する	new=ktranslate("abcあいうgfcded","150","あいう");
	KTRIM	文字値の後ろ側の全角および半角のブランクを削除する	c=ktrim(a)||trim(b);
	KUPCASE	大文字変換	up=kupcase(a);
	KVERIFY	文字値が指定の文字のみ含むかどうかをチェック	chk=kverify("abcdeabあいうfgfcded","abcdefあいう");
乱数	CALL RANCAU	コーシー乱数（シードの詳細制御可能）	call rancau(seed,x);
	CALL RANNOR	正規乱数（シードの詳細制御可能）	call rannor(seed,x);
	CALL RANUNI	一様乱数（シードの詳細制御可能）	call ranuni(seed,x);
	RANCAU	コーシー乱数	x=rancau(seed);
	RANNOR	正規乱数	x=rannor(seed);
	RANUNI	一様乱数、UNIFORMと同じ	x=ranuni(seed);
	UNIFORM	一様乱数、RANUNIの別名	x=uniform(seed);
データセット操作	ATTRC	文字型属性の値をとる	dslabel=attrc(dsid,"label"):
	ATTRN	数値型属性の値をとる	nobs=attrn(dsid,"nobs");
	CLOSE	データセットをクローズ	dsid=close("work.a");
	EXIST	データセットやカタログが存在するかどうかをチェック	rc=exist(work.a,data);

	FETCH	オープンしたデータセットのオブザベーション読み取りポインタを次のオブザベーションに移動する	rc=fetch(dsid);
	FETCHOBS	オープンしたデータセットのオブザベーション読み取りポインタを指定のオブザベーションに移動する	rc=fetchobs(dsid,5,abs);
	GETVARC	FETCHされているオブザベーションの文字変数値を読み取る	cval=getvarc(dsid,varnum(dsid,"varc");
	GETVARN	FETCHされているオブザベーションの数値変数値を読み取る	xval=getvarn(dsid,varnum(dsid,"varx");
	LIBREF	ライブラリ参照名の存在をチェック(値0が返ると存在を意味する)	rc=libref("work");
	OPEN	データセットをオープン	dsid=open("work.a");
	PATHNAME	データライブラリ参照名またはファイル参照名の物理パスを返す	path1=pathname("work");
	SYSMSG	ファイルアクセス時のエラーメッセージまたは警告メッセージを獲得	msg=sysmsg();
	VARFMT	変数に定義されているフォーマット名を返す	fmt=varfmt(dsid,varnum(dsid,"a"));
	VARINFMT	変数に定義されているインフォーマット名を返す	infmt=varinfmt(dsid,varnum(dsid,"a"));
	VARLABEL	変数に定義されているラベル名を返す	label=varlabel(dsid,varnum(dsid,"a"));
	VARLEN	変数に定義されている長さを返す	len=varlen(dsid,varnum(dsid,"a"));
	VARNAME	変数に定義されている変数名を返す	name1=varname(dsid,1);
	VARNUM	変数名の定義されている番号を返す	num=varnum(dsid,"a"));
	VARTYPE	変数に定義されているタイプを返す	type=vartype(dsid,varnum(dsid,"a"));
その他	CALL SYSTEM	OSコマンドを呼び出す	call system("dir c:¥");
	CHOOSEC	文字列リストから指定の番号の文字列を抽出する	a=choosec(2,"abc","de","fgh");
	CHOOSEN	数値リストから指定の番号の数値を抽出する	x=choosen(5,120,35,11,16,280);
	DIF	nオブザベーション前の値との差をとる	d2=dif2(x);
	GETOPTION	オプション設定値を返す	ls=getoption("linesize");
	INPUT	インフォーマットを用いて値を変換	num=input("100",3.);
	LAG	nオブザベーション前の値を返す	l2=lag2(x);
	MISSING	欠損値かどうかをチェック	if missing(x) then put "MISSING VALUE";
	PUT	フォーマットを用いて値を変換	char=put(100,3.);
	SLEEP	実行を休止する	sleep=sleep(10,1);
	SOUNDEX	英語のみ関係する文字列分類アルゴリズム	a=soundex("hello");
	SOUNDSLIKE	2つの文字列を英語の発音で比較	r=soundslike("hello","helow");
	SPEDIS	2つの文字列のレーベンシュタイン距離を返す	distance=spedis("test","twist");
	SYSPARM	SYSPARM=オプション指定値を返す	sysparm=sysparm();
	SYSPROD	そのプロダクトのライセンス有無をチェックする	prod_chk=sysprod("WPS");
	SYSTEM	OSコマンドを呼び出しシステムリターンコードを返す	rc=system("dir c:¥");

[別表 6] フォーマット一覧

分類	フォーマット名	意味	w と d の範囲（デフォルト）	例	結果
数値コード変	BINARYw.	数値をバイナリコード(0 or 1)で書き出す	1-64 (8)	x=256;put x binary10.;	0100000000

換					
	HEXw.	数値を 16 進数で書き出す	1-16 (8)	x=512;put x hex8.;	00000200
	IBw.d	数値を整数バイナリ形式で書き出す	1-8 (4)	x=put(12345,ib8.);put x $hex16.;	3930000000000000
	IBRw.d	数値をOS環境下依存の整数バイナリ形式で書き出す	1-8 (4)	x=put(12345,ibr8.);put x $hex16.;	3930000000000000
	IEEEw.d	数値を IEEE 浮動小数点で書き出す	1-8 (8)	x=put(1,ieee8.);put x $hex16.;	3FF0000000000000
	OCTALw.	数値を 8 進数表記で書き出す	1-24 (3)	x=123456789012345;put x octal24.;	000000003404420603357571
	PDw.d	数値をパック 10 進数形式で書き出す	1-16 (1)	x=put(999,pd4.);put x hex8.;	00000999
	PDJULGw.	数値を z/OS 上のパックジュリアン日付値 yyyydddF の形式で書き出す	3-16 (4)	x=put("31DEC2008"D,pdjulg8.);put x $hex16.;	000000002008366F
	PDJULIw.	数値を z/OS 上の 16 進パックジュリアン日付値 ccyydddF の形式で書き出す	3-16 (4)	x=put("31DEC2008"D,pdjuli8.);put x $hex16.;	000000000108366F
	PIBw.d	数値をOS環境下依存の正の整数バイナリ形式で書き出す	1-8 (1)	x=put(2,pib1.);put x $hex2.;	02
	PIBRw.d	数値を Window OS 環境の正の整数バイナリ形式で書き出す	1-8 (1)	x=put(2,pib1.);put x $hex2.;	02
	PKw.d	数値を符号なしパック 10 進数形式で書き出す。	1-16 (1)	x=put(2,pk1.);put x $hex2.;	02
	RBw.d	数値をOSに依存する実数バイナリ形式で書き出す	2-8 (4)	x=put(2,rb2.);put x $hex4.;	0040
	S370FFw.d	数値を z/OS 上の EBCDIC 文字形式で書き出す	1-32 (12)	x=put(1234,s370ff4.);put x $hex8.;	F1F2F3F4
	S370FIBw.d	数値を z/OS 上の整数バイナリ形式で書き出す	1-8 (4)	x=put(1234,s370fib4.);put x $hex8.;	000004D2
	S370FIBUw.d	数値を z/OS 上の符号なしバイナリ形式で書き出す	1-8 (4)	x=put(1234,s370fib4.);put x $hex8.;	000004D2
	S370FPDw.d	数値を z/OS 上のパック 10 進数形式で書き出す	1-16 (1)	x=put(1234,s370fpd4.);put x $hex8.;	0001234C
	S370FPDUw.d	数値を z/OS 上の符号なしパック 10 進数形式で書き出す	1-16 (1)	x=put(1234,s370fpdu4.);put x $hex8.;	0001234F
	S370FPIBw.d	数値を z/OS 上の正の整数バイナリ形式で書き出す	1-8 (4)	x=put(1234,s370fpib4.);put x $hex8.;	000004D2
	S370FRBw.d	数値を z/OS 上の実数バイナリ形式で書き出す	2-8 (4)	x=put(2,s370frb2.);put x $hex4.;	4120
	S370FZDw.d	数値を z/OS 上のゾーン 10 進数形式で書き出す	1-32 (8)	x=put(1234,s370fzd4.);put x $hex8.;	F1F2F3C4
	S370FZDLw.d	数値を z/OS 上の符号つきゾーン 10 進数形式で書き出す	1-32 (8)	x=put(1234,s370fzdl6.);put x $hex16.;	C0F0F1F2F3F4
	S370FZDSw.d	数値を z/OS 上の符号を分離したゾーン 10 進数形式で書き出す	2-32 (8)	x=put(1234,s370fzds6.);put x $hex16.;	4EF0F1F2F3F4

	S370FZDTw.d	数値を z/OS 上の符号を分離して後ろにつけたゾーン 10 進数形式で書き出す	2-32 (8)	x=put(1234,s370fzdt6.);put x $hex16.;	F0F1F2F3F44E
	S370FZDUw.d	数値を z/OS 上の符号なしゾーン 10 進数形式で書き出す	1-32 (8)	x=put(1234,s370fzdu6.);put x $hex16.;	F0F0F1F2F3F4
	ZDw.d	数値を OS 環境下依存のゾーン 10 進数形式で書き出す	1-16 (8)	x=put(12.5,zd5.1);put x $hex32.;	3030313245
数値編集	BESTw.d	数値を指定の長さで可能な限り詳細なフォーマットで書き出す	1-32 (12)	x=11111111111111111;put x best12.;	1.1111111E17
	COMMAw.d	整数部分は 3 桁おきにカンマを追加し、指定の小数点桁数まで書き出す	1-32 (6)	x=1000000;put x comma12.2;	1,000,000.00
	COMMAXw.d	整数部分は 3 桁おきにピリオドを追加し、小数点はカンマ表示し、指定の小数点桁数まで書き出す	1-32 (6)	x=1000000;put x commax12.2;	1.000.000,00
	Dw.d	変動範囲の大きな数値に対してなるだけ小数点位置を揃えて書き出す	1-32 (12)	x=12.518;put x d6.1;	12.52
	DOLLARw.d	数値を先頭に$、3 桁おきにカンマを付加して書き出す	2-32 (6)	x=1250.50;put x dollar12.2;	$1,250.50
	DOLLARXw.d	数値を先頭に$、3 桁おきにピリオドを付加し、小数点にカンマを使って書き出す	2-32 (6)	x=1250.50;put x dollarx12.4;	$1.250,5000
	Ew.	数値を科学(浮動)小数点法で書き出す	7-32 (12)	x=123.456789;put x e12.;	1.23457E+02
	FLOATw.d	数値を単精度浮動小数点で書き出す。	4-4 (4)	x=put(123.456789,float4.);put x $hex8.;	E0E9F642
	FRACTw.	数値を分数表示で書き出す	4-32 (10)	x=0.3;put x fract10.;	3/10
	NEGPARENw.d	数値をカンマ編集し、かつ、負の値の場合はカッコで囲む	1-32 (6)	x=-12567.8;put x negparen8.0.;	(12,568)
	NUMXw.d	数値を小数点記号としてカンマを用いて書き出す	1-32 (12)	x=-12567.8;put x numx8.1;	-12567,8
	PERCENTw.d	数値を%記号をつけたパーセンテージ表現で書き出す	4-32 (6)	x=0.0512;put x percent6.1;	5.1%
	PVALUEw.d	数値を p 値(帰無仮説を誤って棄却してしまう確率)表示形式で書き出す	3-32 (6)	p=0.0000000001;put p pvalue6.4;	<.0001
	ROMANw.	数値をローマ数字で書き出す	2-32 (6)	x=2008;put x roman12.;	MMVIII
	SSNw.	数値を社会保障番号(Social Security Number)形式に書き出す	11-11 (11)		
	w.d	数値を全体で w 文字、少数点以下 d 桁の形式で書き出す	1-32 (1)	x=10.091;put x 5.1;	10.1
	WORDFw.	数値を英語の数の読み方で分数表現つきで書き出す	5-32767 (10)	x=12.25;put x wordf64.;	twelve and 25/100
	WORDSw.	数値を英語の数の読み方で書き出す	5-32767 (10)	x=12.25;put x words64.;	twelve and twenty-five hundredths

	Zw.d	数値を全体で w 文字、少数点以下 d 桁の形式で書き出す。 先頭の 0 を書き出す点で w.d と異なる	1-32 (1)	x=10.091;put x z5.1;	010.1
日付・時間	DATEw.	日付値を ddmmmyy または ddmmmyyyy の形式で書き出す	5-9 (7)	x=0;put x date9.;	01JAN1960
	DATEAMPMw.d	日時値を ddmmmyy:hh:mm:ss.ss AM または ddmmmyy:hh:mm:ss.ss PM の形式で書き出す	7-40 (19)	x=0;put x dateampm19.;	01JAN60:12:00:00 AM
	DATETIMEw.d	日時値を ddmmmyy:hh:mm:ss.ss の形式で書き出す	7-40 (19)	x=0;put x datetime19.;	01JAN1960:00:00:00
	DAYw.	日付値から日付部分を取り出し dd の形式で書き出す	2-32 (2)	x=0;put x day2.;	1
	DDMMYYw.	日付値を ddmmyy または ddmmyyyy または dd/mm/yy または dd/mm/yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyy10.;	01/01/1960
	DDMMYYBw.	日付値を dd mm yy または dd mm yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyyb10.;	01 01 1960
	DDMMYYCw.	日付値を dd:mm:yy または dd:mm:yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyyc10.;	01:01:1960
	DDMMYYDw.	日付値を dd-mm-yy または dd-mm-yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyyd10.;	01-01-1960
	DDMMYYNw.	日付値を ddmmyy または ddmmyyyy の形式で書き出す	2-8 (8)	x=0;put x ddmmyyn8.;	01011960
	DDMMYYPw.	日付値を dd.mm.yy または dd.mm.yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyyp10.;	01.01.1960
	DDMMYYSw.	日付値を dd/mm/yy または dd/mm/yyyy の形式で書き出す	2-10 (8)	x=0;put x ddmmyys10.;	01/01/1960
	DOWNAMEw.	日付値から曜日を書き出す	1-32 (9)	date="01jan2008"d;put date downame9.;	Tuesday
	DTDATEw.	日時値を ddmmmyy または ddmmmyyyy の形式で書き出す	5-9 (7)	x="01JAN1960:00:00:00"DT;put x dtdate9.;	01JAN1960
	DTMONYYw.	日時値を mmmyy または mmmyyyy の形式で書き出す	5-7 (7)	x="01JAN1960:00:00:00"DT;put x dtmonyy7.;	JAN1960
	DTWKDATXw.	日時値を day-of week, dd name-of month yy または day-of week, dd name-of month yyyy の形式で書き出す	3-37 (29)	x="01JAN1960:00:00:00"DT;put x dtwkdatx29.;	Friday, 1 January 1960
	DTYEARw.	日時値を yy または yyyy の形式で書き出す	2-4 (4)	x="01JAN1960:00:00:00"DT;put x dtyear4.;	1960
	DTYYQCw.	日時値を yy:q または yyyy:q の形式で書き出す	4-6 (4)	x="01JAN1960:00:00:00"DT;put x dtyyqc4.;	60:1
	HHMMw.d	時間値を hh:mm.mm の形式で書き出す	2-20 (5)	x=60;put x hhmm.;	0:01

HOURw.d	時間値を hh.hh の形式で書き出す	2-20 (2)	x=3600;put x hour2.;	1
JULDAYw.	日付値からジュリアン日付値の日付部分 ddd を書き出す	3-32 (3)	x="31DEC2008"D;put x julday3.;	366
JULIANw.	日付値をジュリアン日付値 yyddd または yyyyddd の形式で書き出す	5-7 (5)	x="31DEC2008"D;put x julian5.;	08366
MMDDYYw.	日付値を mmddyy , mmddyyyy , mm/dd/yy または mm/dd/yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyy10.;	12/31/2008
MMDDYYBw.	日付値を mm dd yy または mm dd yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyyb10.;	12 31 2008
MMDDYYCw.	日付値を mm:dd:yy または mm:dd:yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyyc10.;	12:31:2008
MMDDYYDw.	日付値を mm-dd-yy または mm-dd-yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyyd10.;	12-31-2008
MMDDYYNw.	日付値を mmddyy または mmddyyyy の形式で書き出す	2-8 (8)	x="31DEC2008"D;put x mmddyyn8.;	12312008
MMDDYYPw.	日付値を mm.dd.yy または mm.dd.yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyyp10.;	12.31.2008
MMDDYYSw.	日付値を mm/dd/yy または mm/dd/yyyy の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x mmddyys10.;	12/31/2008
MMSSw.d	時間値を mm:ss.ss の形式で書き出す	2-20 (5)	x="01:10:30"T;put x mmss8.;	70:30
MMYYw.	日付値を mmMyy または mmMyyyy の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x mmyy7.;	12M2008
MMYYCw.	日付値を mm:yy または mm:yyyy の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x mmyyc7.;	12:2008
MMYYDw.	日付値を mm-yy または mm-yyyy の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x mmyyd7.;	12-2008
MMYYNw.	日付値を mmyy または mmyyyy の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x mmyyn6.;	122008
MMYYPw.	日付値を mm.yy または mm.yyyy の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x mmyyp7.;	12.2008
MMYYSw.	日付値を mm/yy または mm/yyyy の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x mmyys7.;	12/2008
MONNAMEw.	日付値から月名を書き出す	1-32 (9)	x="31DEC2008"D;put x monname9.;	December
MONTHw.	日付値から月部分を取り出し mm の形式で書き出す	1-32 (9)	x="31DEC2008"D;put x month2.;	12
MONYYw.	日付値を mmmyy または mmmyyyy の形式で書き出す	5-7 (5)	x="31DEC2008"D;put x monyy5.;	DEC08
QTRw.	日付値を四半期 q 形式で書き出す	1-32 (1)	x="31DEC2008"D;put x qtr1.;	4
QTRRw.	日付値を四半期 qr 形式で書き出す	3-32 (3)	x="31DEC2008"D;put x qtrr3.;	IV

TIMEw.d	時間値または日時値を hh:mm:ss.ss の形式で書き出す	2-20 (8)	x="01JAN1960:00:00:00"DT;put x time8.;	0:00:00
TIMEAMPMw.d	時間値または日時値を hh:mm:ss.ss AM または、hh:mm:ss.ss PM の形式で書き出す	2-20 (11)	x="01JAN1960:00:00:00"DT;put x timeampm11.;	12:00:00 AM
TODw.	日時値から hh:mm:ss.ss の形式で時間部分のみを書き出す	2-20 (8)	x="01JAN1960:00:00:00"DT;put x tod8.;	00:00:00
WEEKDATEw.	日付値を 曜日, 月名 dd, yy または 曜日, 月名 dd, yyyy の形式で書き出す	3-37 (29)	x="31DEC2008"D;put x weekdate3.;	Wed
WEEKDATXw.	日付値を 曜日, dd 月名 yy または 曜日, dd 月名 yyyy の形式で書き出す	3-37 (29)	x="31DEC2008"D;put x weekdatx29.;	Wednesday, 31 December 2008
WEEKDAYw.	日付値を 曜日を表す整数で書き出す	1-32 (1)	x="31DEC2008"D;put x weekday1.;	4
WORDDATEw.	日付値を 月名 dd, yy または 月名 dd, yyyy の形式で書き出す	3-32 (18)	x="31DEC2008"D;put x worddate18.;	December 31, 2008
WORDDATXw.	日付値を dd 月名 yy または dd 月名 yyyy の形式で書き出す	3-32 (18)	x="31DEC2008"D;put x worddatx18.;	31 December 2008
YEARw.	日付値から年部分を取り出し yy または yyyy の形式で書き出す	2-32 (4)	x="31DEC2008"D;put x year4.;	2008
YYMMw.	日付値を yyMmm または yyyyMmm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymm7.;	2008M12
YYMMCw.	日付値を yy:mm または yyyy:mm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymmc7.;	2008:12
YYMMDw.	日付値を yy-mm または yyyy-mm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymmd7.;	2008-12
YYMMNw.	日付値を yymm または yyyymm の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yymmn6.;	200812
YYMMPw.	日付値を yy.mm または yyyy.mm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymmp7.;	2008.12
YYMMSw.	日付値を yy/mm または yyyy/mm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymms7.;	2008/12
YYMMDDw.	日付値を yymmdd , yyyymmdd , yy/mm/dd または yyyy/mm/dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmdd10.;	2008-12-31
YYMMDDBw.	日付値を yy mm dd または yyyy mm dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmddb10.;	2008 12 31
YYMMDDCw.	日付値を yy:mm:dd または yyyy:mm:dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmddc10.;	2008:12:31
YYMMDDDw.	日付値を yy-mm-dd または yyyy-mm-dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmddd10.;	2008-12-31
YYMMDDNw.	日付値を yymmdd または yyyymmdd の形式で書き出す	2-8 (8)	x="31DEC2008"D;put x yymmddn8.;	20081231

	YYMMDDPw.	日付値を yy.mm.dd または yyyy.mm.dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmddp10.;	2008.12.31
	YYMMDDSw.	日付値を yy/mm/dd または yyyy/mm/dd の形式で書き出す	2-10 (8)	x="31DEC2008"D;put x yymmdds10.;	2008/12/31
	YYMONw.	日付値を yymmm または yyyymmm の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yymon7.;	2008DEC
	YYQw.	日付値を yyQq または yyyyQq の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yyq6.;	2008Q4
	YYQCw.	日付値を yy:q または yyyy:q の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yyqc6.;	2008:4
	YYQDw.	日付値を yy-q または yyyy-q の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yyqd6.;	2008-4
	YYQNw.	日付値を yyq または yyyyq の形式で書き出す	3-32 (5)	x="31DEC2008"D;put x yyqn6.;	20084
	YYQPw.	日付値を yy.q または yyyy.q の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yyqp6.;	2008.4
	YYQSw.	日付値を yy/q または yyyy/q の形式で書き出す	4-32 (6)	x="31DEC2008"D;put x yyqs6.;	2008/4
	YYQRw.	日付値を yyQqr または yyyyQqr の形式で書き出す	6-32 (8)	x="31DEC2008"D;put x yyqr6.;	08QIV
	YYQRCw.	日付値を yy:qr または yyyy:qr の形式で書き出す	6-32 (8)	x="31DEC2008"D;put x yyqrc6.;	08:IV
	YYQRDw.	日付値を yy-qr または yyyy-qr の形式で書き出す	6-32 (8)	x="31DEC2008"D;put x yyqrd6.;	08-IV
	YYQRNw.	日付値を yyqr または yyyyqr の形式で書き出す	5-32 (7)	x="31DEC2008"D;put x yyqrn.;	2008IV
	YYQRPw.	日付値を yy.qr または yyyy.qr の形式で書き出す	6-32 (8)	x="31DEC2008"D;put x yyqrp6.;	08.IV
	YYQRSw.	日付値を yy/qr または yyyy/qr の形式で書き出す	6-32 (8)	x="31DEC2008"D;put x yyqrs6.;	08/IV
文字編集	$CHARw.	文字列を先頭のブランクは詰めずにそのまま書き出す	1-32767 (8)	c="|"||put(" ABC ",$char7.)||"|";put c;		ABC	
	$Fw.	文字列を文字列として書き出す。 $w.と同じ	1-32767 (1)	c="ABC";put c $F3.;	ABC
	$QUOTEw.	文字列をダブルクオテーションで囲んで書き出す	2-32767 (8)	c="ABC";put c $quote5.;	"ABC"
	$REVERJw.	文字列を逆順に書き出す。先頭および後ろのブランクは取り除かない	1-32767 (1)	c=put(" ABC ",$char7.);put "|" c $reverj7. "|";		CBA	
	$REVERSw.	文字列を逆順に書き出す。先頭のブランクは取り除かないが後ろは除く	1-32767 (1)	c=put(" ABC ",$char7.);put "|" c $revers7. "|";	|CBA	
	$UPCASEw.	文字列を大文字化して書き出す	1-32767 (1)	c="abc";put c $upcase3.;	ABC
	$w.	文字列を文字列として書き出す。 $Fw.と同じ	1-32767 (1)	c="ABC";put c $3.;	ABC

文字コード変換	$ASCIIw.	文字列を ASCII コード文字で書き出す。ただし PC 環境では$CHARw.と同じ	1-32767 (1)	c=put("ABC",$ascii3.);put c $hex6.;	414243
	$BINARYw.	文字列をバイナリコード(0 or 1)で書き出す	1-32767 (1)	c="ABC";put c $binary24.;	010000010100001001000011
	$EBCDICw.	文字列を EBCDIC コード文字で書き出す	1-32767 (1)	c=put("ABC",$ebcdic3.);put c $hex6.;	C1C2C3
	$HEXw.	文字列を 16 進コードで書き出す	1-32767 (1)	c="ABC";put c $hex6.;	414243
	$OCTALw.	文字列を 8 進コードで書き出す	1-24 (3)	c="ABC";put c $octal6.;	101102

注:数値フォーマットに共通: w.d の d を指定した場合、小数点以下 d 桁を表示する。

[別表 7] インフォーマット一覧

分類	フォーマット名	意味	w と d の範囲 (デフォルト)	例	結果
数値	BESTw.d	数値を読み込む。1個のピリオドは欠損値とみなす	1-32 (1)	x=input("1.1111111E17",best12.);put x;	1.1111111E17
	BITSw.d	文字列をビット列として読み込み対応する数値に変換する	1-64 (1)	x=input("A",bits8.);put x;	65
	COMMAw.d	数字とピリオド、そして先頭のプラス記号とマイナス記号以外のカンマ、ブランク、ダッシュ、ドル記号、%記号、右カッコを除去して読み込む。先頭の左カッコはマイナス記号に変換する	1-32 (1)	x=input("-$1,234.5",comma12.);put x;	-1234.5
	COMMAXw.d	数字とカンマ、そして先頭のプラス記号とマイナス記号以外のピリオド、ブランク、ダッシュ、ドル記号、%記号、右カッコを除去して読み込む。先頭の左カッコはマイナス記号に変換する	1-32 (1)	x=input("-$1.234,5",commax12.);put x;	-1234.5
	DOLLARw.d	数字とピリオド、そして先頭のプラス記号とマイナス記号以外のカンマ、ブランク、ダッシュ、ドル記号、%記号、右カッコを除去して読み込む。先頭の左カッコはマイナス記号に変換する	1-32 (1)	x=input("$1,250.50",dollar12.);put x;	1250.5
	DOLLARXw.d	数字とカンマ、そして先頭のプラス記号とマイナス記号以外のピリオド、ブランク、ダッシュ、ドル記号、%記号、右カッコを除去して読み込む。先頭の左カッコはマイナス記号に変換する	1-32 (1)	x=input("$1.250,50",dollarx12.);put x;	1250.5
	Ew.	科学(浮動)小数点法で書かれた数値を読み込む	1-32 (1)	x=input("1.23457E+02",e12.);put x;	123.457
	FLOATw.d	4 バイトのバイナリ浮動小数点形式で書かれた数値を読み込む	4-4 (4)	x=input("E0E9F642"X,float4.);put x;	123.45678711
	HEXw.	16 進数を表す文字列を数値として読み込む	1-16 (8)	x=input("FFFF",hex4.);put x;	65535
	IBw.d	Window 環境下の整数バイナリ形式で書かれた値を読み込む	1-8 (4)	x=input("39300000"X,ibr8.);put x;	12345

	IBRw.d	OS環境下依存の整数バイナリ形式で書かれた値を読み込む	1-8 (4)	x=input("39300000"X,ibr8.);put x;	12345
	PERCENTw.	数値をパーセントとして読み込む。ピリオド以外のカンマ、ブランク、ダッシュ、パーセント記号と右カッコを無視し、先頭の左カッコはマイナス記号に変換する。パーセント記号は除去され値は 100 で割る	1-32 (6)	x=input("(5.12%",percent6.);put x;	-0.0512
	PIBw.d	OS環境依存の正の整数バイナリ形式の数値を読み込む	1-8 (1)	x=input("02"X,pib2.);put x;	2
	PIBRw.d	Window OS 環境下の正の整数バイナリ形式の数値を読み込む	1-8 (1)	x=input("02"X,pib2.);put x;	2
	PKw.d	符号なしパック 10 進数形式の数値を読み込む	1-16 (1)	x=input("02"X,pk2.);put x;	2
	RBw.d	OS環境に依存する実数バイナリ形式の数値を読み込む	2-8 (4)	x=input("0040"X,rb4.);put x;	2
	w.d	全体で w 文字、少数点以下 d 桁の形式で書かれた数値を読み込む。ピリオド1個(.)は欠損値として読み込む	1-32 (1)	x=input("10.091",6.3);put x;	10.091
	PDw.d	パック 10 進数形式で書かれた値を読み込む	1-16 (1)	x=input("00000999"X,pd8.);put x;	999
日付・時間	DATEw.	ddmmmyy または ddmmmyyyy の形式で書かれた値を日付値として読み込む	7-32 (7)	x=input("01JAN1960",date9.);put x;	0
	DATETIMEw	ddmmmyy:hh:mm:ss.ss または ddmmmyyyy:hh:mm:ss.ss の形式で書かれた値を日時値として読み込む	13-40 (18)	x=input("01JAN1960:00:01:00",datetime18.);put x;	60
	DDMMYYw.	ddmmyy , ddmmyyyy の形式、または dd/mm/yy , dd/mm/yyyy などの区切り文字付きの形式で書かれた値を日付値として読み込む	6-32 (6)	x=input("10/01/1960",ddmmyy10.);put x;	9
	JULIANw.	ジュリアン日付値 yyddd または yyyyddd の形式で書かれた値を日付値として読み込む	5-32 (5)	x=input("08366",julian5.);put x;	17897
	MMDDYYw.	mmddyy , mmddyyyy の形式、または mm/dd/yy , mm/dd/yyyy などの区切り文字付き形式で書かれた文字列を日付値として読み込む	2-10 (8)	x=input("12/31/2008",mmddyy10.);put x;	17897
	MONYYw.	mmmyy , mmmyyyy の形式、または mmm/yy , mmn/yyyy などの区切り文字付き形式で書かれた文字列を日付値として読み込む	5-32 (5)	x=input("DEC08",monyy5.);put x;	17867
	TIMEw.	hh:mm:ss.ss の形式で書かれた数値を時間値として読み込む	5-32 (8)	x=input("12:10:30",time8.);put x;	43830
	YYMMDDw.	yymmdd , yyyymmdd または yycmmcdd, yyyycmmcdd(c は /などの区切り文字) の形式で書かれた値を日付値として読み込む	6-32 (6)	x=input("2008-12-31",yymmdd10.);put x;	17897
	YYMMNw.	yymm または yyyymm の形式で書かれた値を日付値とし	4-6 (6)	x=input("200812",yymmn6.);put x;	17867

		て読み込む			
文字	$CHARw.	文字列を先頭のブランクは詰めずにそのまま読み込む。また、空白付き空白付きでないにかかわらず、ピリオド1個は欠損値とみなさない	1-32767 (8)	c=input(" ABC ",$char10.);put c $char.;	ABC
	$CHARZBw.	文字列をバイナリゼロはブランクに変換して読み込む。その他は$CHARw. と同じ	1-32767 (8)	c=input("2041420043"X,$charzb10.);put c $char.;	AB C
	$UPCASEw.	文字列を大文字化して読み込む	1-32767 (8)	c=input("abc",$upcase3.);put c;	ABC
	$w.	先頭のブランクを無視して文字列を読み込む。 また、空白付き空白付きでないにかかわらず、ピリオド1個は欠損値とみなす	1-32767 (1)	c=input(" ABC",$6.);put c $char6.;	ABC
	$ASCIIw.	ASCII コード文字で書かれた文字列を読み込む。ただし PC 環境では$CHARw.と同じ	1-32767 (1)	c=input("414243"X,$ascii3.);put c;	ABC
	$BINARYw.	バイナリコード(0 or 1)で書かれた文字列を読み込む	1-32767 (1)	c=input("010000010100001001000011",$binary24.);put c ;	ABC
	$EBCDICw.	EBCDIC コード文字列を読み込む	1-32767 (1)	c=input("C1C2C3"X,$ebcdic6.);put c $char.;	ABC
	$HEXw.	16 進コードで書かれた文字列を読み込む	1-32767 (2)	c=input("414243",$hex6.);put c $char.;	ABC
	$PHEXw.	パック 16 進コードで書かれた文字列を読み込む	1-32767 (2)	c=input("414243",$phex6.);put c $char.;	343134
z/OS 数値	S370FFw.d	z/OS 環境下の EBCDIC 文字形式で書かれた数値を読み込む	1-32 (12)	x=input("F1F2F3F4"X,s370ff8.);put x;	1234
	S370FIBw.d	z/OS 環境下の整数バイナリ形式で書かれた数値を読み込む	1-8 (4)	x=input("000004D2"X,s370fib8.);put x;	1234
	S370FIBUw.d	z/OS 環境下の符号なしバイナリ形式で書かれた数値を読み込む	1-8 (4)	x=input("000004D2"X,s370fib8.);put x;	1234
	S370FPDw.d	z/OS 環境下のパック 10 進数形式で書かれた数値を読み込む	1-16 (1)	x=input("0001234C"X,s370fpd8.);put x;	1234
	S370FPDUw.d	z/OS 環境下の符号なしパック 10 進数形式で書かれた数値を読み込む	1-16 (1)	x=input("0001234F"X,s370fpdu8.);put x;	1234
	S370FPIBw.d	z/OS 環境下の正の整数バイナリ形式で書かれた数値を読み込む	1-8 (4)	x=input("000004D2"X,s370fpib8.);put x;	1234
	S370FRBw.d	z/OS 環境下の実数バイナリ形式で書かれた数値を読み込む	2-8 (6)	x=input("4120"X,s370frb4.);put x;	2
	S370FZDw.d	z/OS 環境下のゾーン 10 進数形式で書かれた数値を読み込む	1-32 (8)	x=input("F1F2F3C4"X,s370fzd8.);put x;	1234
	S370FZDBw.d	z/OS 環境下の 0 はブランクで表現された符号つきゾーン 10 進数形式で書かれた数値を読み込む。	1-32 (8)	x=input("C0F0F1F2F3F4"X,s370fzdl12.);put x;	1234
	S370FZDLw.d	z/OS 環境下の符号つきゾーン 10 進数形式で書かれた数値を読み込む	1-32 (8)	x=input("C0F0F1F2F3F4"X,s370fzdl12.);put x;	1234
	S370FZDSw.d	z/OS 環境下の符号を分離したゾーン 10 進数形式で書かれた数値を読み込む	2-32 (8)	x=input("4EF0F1F2F3F4"X,s370fzds12.);put x;	1234

	S370FZDTw.d	z/OS 環境下の符号を分離して後ろにつけたゾーン 10 進数形式で書かれた数値を読み込む	2-32 (8)	x=input("F0F1F2F3F44E"X,s370fzdt12.);put x;	1234
	S370FZDUw.d	z/OS 環境下の符号なしゾーン 10 進数形式で書かれた数値を読み込む	1-32 (8)	x=input("F0F0F1F2F3F4"X,s370fzdu12.);put x;	1234
	ZDw.d	OS 環境依存のゾーン 10 進数形式(最後のバイトに符号つきデジタル)で書かれた数値を読み込む	1-32 (1)		
	ZDBw.d	OS 環境依存の0がブランクで表現されているゾーン 10 進数形式で書かれた数値を読み込む	1-32 (1)		
z/OS 日付・時間	MSECw.	z/OS 環境下の 8 バイトマイクロ秒値を時間値として読み込む	1-8 (8)		
	PDJULGw.	z/OS 環境下のパックジュリアン日付値 yyyydddF の形式で書かれた値を日付値として読み込む	4-4 (4)	x=input(put("31DEC2008"D,pdjulg4.),pdjulg4.);put x;	17897
	PDJULIw.	z/OS 環境下の 16 進パックジュリアン日付値 ccyydddF の形式で書かれた値を日付値として読み込む	4-4 (4)	x=input(put("31DEC2008"D,pdjuli4.),pdjuli4.);put x;	17897
	PDTIMEw.	z_OS 環境下の RMF や SMF レコードに見られる 0hhmmssF 形式の 16 進パック時間値を読み込む	4-4 (4)		
	RMFDURw.	z_OS 環境下の mmsstttF 16 進形式の RMF レコードの時間値を読み込む	4-4 (4)		
	RMFSTAMPw.	z_OS 環境下の 0hhmmssFccyydddF 16 進形式の RMF レコードの時間値を読み込む	8-8 (8)		
	SMFSTAMPw.d	z_OS 環境下の hhhhhhhhccyydddF 16 進形式の SMF レコードの日時値を読み込む	8-8 (8)		
	TODSTAMPw.	z_OS 環境下の 8 バイトバイナリ整数形式の時間値(Time Of Day)を数値として読み込む	1-8 (8)		
	TUw.	z_OS 環境下の 4 バイトバイナリ整数形式の時間単位(Timer Unit)を数値として読み込む	4-4 (4)		

[別表 8] merge ステートメントによる 2 つのデータセットのマージ例

(例示データ) 1 対 n のマージ例になっていることに注意

	A	
obs	name	value1
1	A	11
2	B	21
3	C	31

	B	
obs	name	value2
1	A	111
2	A	112
3	B	221

4	B	222
5	B	223
6	D	441
7	D	442

(1) 基本的な merge ステートメント+ by 変数の指定 (いずれかのデータセットに含まれる by 変数値がすべて抽出され、個々の by 変数値のオブザベーション数は左右いずれか多い方に合わせて出力されます)

```
data merge1;
   merge A B;
   by name;
run;
```

	merge1		
Obs	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223
6	C	31	.
7	D	.	441
8	D	.	442

(SQL の完全外部結合を意味します)

```
proc sql;
   reset print number;
   select coalesce(A.name, B.name) as name,
      value1, value2 from A full join B
      on A.name = B.name;
quit;
```

Row	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	223
5	B	21	222
6	C	31	.
7	D	.	442
8	D	.	441

(2) 左側の in 変数のみ if 条件で使用 (左側に含まれる by 変数値のみ抽出され、個々の by 変数値のオブザベーション数は左右いずれか多い方に合わせて出力されます)

```
data merge2;
   merge A(in=A) B;
   by name;
   if A=1;
run;
```

	merge2		
Obs	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223
6	C	31	.

(SQL の左外部結合を意味します)

```
proc sql;
   reset print number;
   select coalesce(A.name, B.name) as name,
      value1, value2 from A left join B
      on A.name = B.name;
quit;
```

Row	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	223
5	B	21	222
6	C	31	.

(3) 右側の in 変数のみ if 条件で使用 (右側に含まれる by 変数値のみ抽出され、個々の by 変数値のオブザベーション数は左右いずれか多い方に合わせて出力されます)

```
data merge3;
   merge A B(in=B);
   by name;
   if B=1;
run;
```

	merge3		
Obs	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223
6	D	.	441
7	D	.	442

(SQL の右外部結合を意味します)

```
proc sql;
   reset print number;
   select coalesce(A.name, B.name) as name,
      value1, value2 from A right join B
      on A.name = B.name;
quit;
```

Row	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	223
5	B	21	222
6	D	.	442
7	D	.	441

(4) 左右の in 変数を AND 条件で結んで使用 (左右両方に共通に含まれる by 変数値のみ抽出され、個々の by 変数値のオブザベーション数は左右いずれか多い方に合わせて出力されます)

```
data merge4;
   merge A(in=A) B(in=B);
```

	merge4		
Obs	name	value1	value2

```
   by name;
   if A=1 and B=1;
run;
```

1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223

(SQL の内部結合を意味します)

```
proc sql;
   reset print number;
   select coalesce(A.name, B.name) as name,
      value1, value2 from A, B
      where A.name = B.name;
quit;
```

Row	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223

(5) 左側の in 変数のみ使用し、冒頭で in 変数をリセット (左側に含まれる by 変数値のみ抽出され、個々の by 変数値のオブザベーション数は左側のオブザベーション数に合わせて出力されます)

```
data merge5;
   A=0;
   merge A(in=A) B;
   by name;
   if A=1;
run;
```

	merge5		
Obs	name	value1	value2
1	A	11	111
2	B	21	221
3	C	31	.

(SQL で書くと、左外部結合した後、元の左側の個々の Row と結合した一番最初の Row のみ抽出することを意味します)

```
proc sql;
   reset print number;
   create table A1 as
   select *, monotonic() as s1 from A;
   select *, monotonic() as s from
   (select coalesce(A1.name, B.name) as name,
      value1, value2, s1 from A1 left join B
      on A1.name = B.name)
   group by s1 having min(s)=s;
quit;
```

Row	name	value1	value2	s1	s
1	A	11	111	1	1
2	B	21	221	2	3
3	C	31	.	3	6

(6) in 変数の使用 (右側のみ) し、冒頭で in 変数をリセット (右側に含まれる by 変数値のみ抽出され、

個々の by 変数値のオブザベーション数は右側のオブザベーション数に合わせて出力されます）

```
data merge6;
   B=0;
   merge A B(in=B);
   by name;
   if B=1;
run;
```

	merge6		
Obs	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223
6	D	.	441
7	D	.	442

（右外部結合した後、元の右側の個々の Row と結合した一番最初の Row のみ抽出する処理を意味します）

```
proc sql;
   reset print number;
   create table B1 as
   select *, monotonic() as s2 from B;
   select *, monotonic() as s from
   (select coalesce(A.name, B1.name) as name,
      value1, value2, s2 from A right join B1
      on A.name = B1.name)
   group by s2 having min(s)=s;
quit;
```

Row	name	value1	value2	s2	s
1	A	11	111	1	1
2	A	11	112	2	2
3	B	21	221	3	3
4	B	21	222	4	5
5	B	21	223	5	4
6	D	.	441	6	7
7	D	.	442	7	6

（注：この例では右外部結合結果と同じになります）

(7) 左右の in 変数を AND 条件で結んで使用し、冒頭で両方の in 変数をリセット（左右両方に共通に含まれる by 変数値のみ抽出され、個々の by 変数値のオブザベーション数は左右いずれか短い方に合わせて出力されます）

```
data m7;
   A=0;B=0;
   merge A(in=A) B(in=B);
   by name;
   if A=1 and B=1;
run;
```

	merge7		
Obs	name	value1	value2
1	A	11	111
2	B	21	221

（内部結合した後、同じ name 値の中の両方のデータでの最初の Row のみ抽出する処理を意味します）

```
proc sql;
   reset print number;
   create table A1 as
   select *, monotonic() as s1 from A;
```

```
  create table B1 as
  select *, monotonic() as s2 from B;
  select *, monotonic() as t from
    (select *, monotonic() as s from
      (select coalesce(A1.name, B1.name) as name,
       value1, value2, s1, s2 from A1 inner join B1
       on A1.name = B1.name)
     group by s2 having min(s)=s)
  group by s1 having min(t)=s;
quit;
```

Row	name	value1	value2	s1	s2	s	t
1	A	11	111	1	1	1	1
2	B	21	221	2	3	3	3

注意：

・上記 SQL は 1 対 n のマージ（by 変数(name)の個々の値がデータセット A ではすべて 1 つずつしか存在せず、データセット B では複数存在する可能性がある場合のマージ）の場合に DATA ステップの結果と一致する SQL を例として示しています。

以下の例のように、n 対mのマージ（個々の by 変数値が両方のデータセットに複数存在する可能性がある場合のマージ）の場合には、例示した SQL の結果は DATA ステップの結果と異なることに注意して下さい。

(例示データ) n 対 m のマージ例になっていることに注意

obs	A	
	name	value1
1	A	11
2	B	21
3	B	22
4	C	31

obs	B	
	name	value2
1	A	111
2	A	112
3	B	221
4	B	222
5	B	223
6	D	441
7	D	442

基本的な merge ステートメント+ by 変数の指定（いずれかのデータセットに含まれる by 変数値がすべて抽出され、個々の by 変数値のオブザベーション数は左右いずれか多い方に合わせて出力されます）

```
data merge1;
   merge A B;
   by name;
run;
```

Obs	merge1		
	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	22	222
5	B	22	223
6	C	31	.
7	D	.	441
8	D	.	442

(1 対 n で表示した同じ SQL)(SQL 結果は name 値に関して 2*3=6 行の Row を出力します。)

```
proc sql;
   reset print number;
   select coalesce(A.name, B.name) as name,
      value1, value2 from A full join B
      on A.name = B.name;
quit;
```

Row	name	value1	value2
1	A	11	111
2	A	11	112
3	B	22	221
4	B	22	223
5	B	22	222
6	B	21	221
7	B	21	223
8	B	21	222
9	C	31	.
10	D	.	442
11	D	.	441

・ (1)のマージは、以下の DATA ステップのマージ結果と同じです。

(8) merge ステートメント+ by 変数の指定+両方の in 変数を OR 条件で結合して指定

```
data merge8;
   A=0;B=0;  /* 記述しなくてもかまわない */
   merge A(in=A) B(in=B);
   by name;
   if A=1 or B=1;
run;
```

	merge8		
Obs	name	value1	value2
1	A	11	111
2	A	11	112
3	B	21	221
4	B	21	222
5	B	21	223
6	C	31	.
7	D	.	441
8	D	.	442

お問合せ先

本資料は 3 日間の Base SAS オンサイト講習会用資料として作成したものです。本資料に関するご質問、その他講習実施等に関するお問合せは以下の宛先までお願いします。

データマインテック株式会社
本社 〒201-0004 東京都狛江市岩戸北 3-3-6-405
info@dataminetech.co.jp

なお、本資料は予告なく改訂される場合があります。下記のホームページで公開する最新の資料をご参照ください。

http://www.dataminetech.co.jp/