Paper HOW96

Applications Design and Development, a case study of the EDA Summarize-Each-Variable Suite

Ronald J. Fehd Senior Maverick
Theoretical Programmer Stakana Analytics

Abstract

Description : This presentation shows the programming steps in the development of an Exploratory Data Analysis (EDA) application.

Purpose : The case study consists of six files which summarize each variable in any specified data set. The discipline shows how to write programs and conduct both unit and integration tests. The emphasis is on writing small, easy to read programs, which may be converted to macros at a later time. Global macro variables are used as parameters. This unit is the first of four of a planned day-long seminar on applications development for programmers.

Audience : beginning programmers and intermediate to advanced users

In this paper

Quote

Information is *the* difference
that makes *a* difference

Gregory Bateson, 1904–1980
Steps to an Ecology of Mind, 1972

Checklist of Ideas

• cardinality ratio	n-levels, n-obs
• categories of variables	few discrete many continuous row-id unique
• guarantee	quality, readable, tested
• list contains items, tasks, or lists	Polish notation
• mnemonics	DJD, HIPO, LATCH
• requirements, design, 80/20, specifications	change notice
• style guide	naming conventions
• data structure	less than 1% of statements 99% of success
• global macro variables	parameters
• hierarchy of program types	module, routine, or subroutine
• know thy options	prepare for testing
• small is beautiful	easy to test
• use SAS naming conventions	avoid quirks!

---

time budget

This table shows the time spent during applications development.

Phase	Time	Action	Time
design	1/2	understand problem: education and research	1/3
		development coding	1/6
testing	1/2	component or unit test	1/4
		systems or integration test	1/4

The Mythical Man-Month, by Fred Brooks, on the development of the IBM 360

EDA requirements

for each variable in a data set, perform frequency

module	routine	subroutine
01-data-class		
proc-1-freq		make-list-vars
	proc-list-vars	proc-freq

Compare to Change Notice requirements on page 14.

---

discipline of applications development

• hard-coded		absolute
• soft-coded	use macro variables as parameters	relative
• split into		*subroutine.sas* *subroutine-test.sas*
• testing		unit integration

Research, Contents Procedure

contents sashelp.class

```
PROC contents data = sashelp.class
```

```
Data Set Name  SASHELP.CLASS                  Observations   19
Member Type    DATA                           Variables       5

Created        Tuesday, May 24, 2011 01:52:28 PM   [granularity]
Label          Student Data

Folder         ...\core\sashelp\                   [libref]
Filename       class.sas7bdat                      [data set name]
```

Location Alphabet Time Category Hierarchy

5 Ways of Organizing Information

```
Alphabetic List of Variables and Attributes

Var
Num  Variable    Type    Len

3    Age         Num       8
4    Height      Num       8
1    Name        Char      8
2    Sex         Char      1
5    Weight      Num       8
```

Location Alphabet Time Category Hierarchy

desired program

The objective of list processing is to have SAS write statements like this program.

```
%let in_data = sashelp.class;

proc freq data = &in_data ...
          tables Age      ...
          tables Height   ...
          tables Name     ...
          tables Sex      ...
          tables Weight   ...

proc display data = out_frequencies;
```

slide 10

desired output

The output data sets are concatenated and look like this.

```
       valu    valu
name   char    num   COUNT  PERCENT
-----  ------  ----  -----  -------
Name   Alfred    .       1    5.263
       Alice     .       1    5.263
...
Sex    F         .       9   47.368
       M         .      10   52.631
Age           11.0       2   10.526
              12.0       5   26.315
...
Height        51.3       1    5.263
              56.3       1    5.263
```

slide 11

Bottom Up, Frequency Procedure

step.1 hard-coded

This program is the first step in applications development, getting an example program in SAS. Concatenation requires that the output data sets have a consistent data structure, i.e., same variable names. This program shows the names of variables in an output data set from the frequency procedure.

```
proc freq data = sashelp.class;
          tables sex / out = out_freq_sex;
          tables age / out = out_freq_age;
proc sql; describe table     out_freq_sex;
          describe table     out_freq_age;
          quit;
```

r11-freq.log

```
create table out_freq_sex
   Sex     char(1),
   count   num label = Frequency Count,
   percent num label = Percent of Total Frequency

create table out_freq_age
   Age     num,
```

slide 12

Notes: 1. Discipline step.1: hard-coded statements. 2. `describe table` writes to the log. 3. file-naming conventions: r1?-freq for frequency procedure.

step.2 soft-coded

This program shows the allocation of two macro variables, `in_data` and `var`, as the parameters for a sub-program.

```
%let in_data   = sashelp.class;
%let var       = sex;

proc freq data = &in_data;
          tables &var / out = out_freq;

proc sql; describe table &syslast;
          quit;
```

slide 13

Notes: 1. Discipline step.2: soft-coded statements, using global macro variables as parameters. 2. `syslast` is a global automatic macro variable, which contains the two-level name of the data set created in the previous step.

---

step.3.1 split into called

This sub-program shows the references of the two macro variables, `in_data` and `var`, as the parameters for this called sub-routine.

```
 1 proc freq   data    =  &in_data;
 2             tables     &var    / noprint
 3                out  =   out_freq
 4            (rename  = (&var = valu_&type ));
 5
 6 proc sql;   describe  table &syslast;
 7             quit;
 8
 9 proc append data    =       &syslast
10             base    =  out_data;
```

slide 14

Notes: 1. Discipline step.3: split soft-coded program into two programs. 2. Lines 5-6, see program `describe`, page 11. 3. Style guide: use white space to group keywords and arguments in vertical columns.

---

step.3.2 . . . and caller

This program shows the allocation of two macro variables, `in_data` and `var`, and the call of the sub-routine.

```
1  %let in_data = sashelp.class;
2
3  %let var  = sex;
4  %let type = c;
5  %include    ’r13-freq.sas’/source2;
6
7  %let var  = age;
8  %let type = n;
9  %include    ’r13-freq.sas’/source2;
10
11 proc print data = &syslast;
12 run;
```

slide 15

Notes: 1. lines 5, 9; option `source2` is a global option, shown in next program, and option of the `%include` statement; it echoes the statements in the file to the log. 2. The object is described first and then the verb. This is an example of a task written in Reverse Polish Notation, i.e., the verb after the object comes. Why Yoda famous was. 3. Polish notation: 2+2=4 :: infix notation; + 2 2 :: prefix notation; 2 2 + reverse notation.

---

debugging

```
NOTE: Appending WORK.OUT_FREQ to WORK.OUT_DATA.
WARNING: Variable valu_N was not found on BASE file.
                       ^
WARNING: Variable valu_C was not found on DATA file.
                       ^
ERROR: No appending done because of anomalies listed above.
NOTE: Statements not processed because of errors noted above.
```

slide 16

Data Structure

polishing data structure

Data sets to be appended must have same variable names. The variable is renamed in the frequency procedure output data set and the data step then adds the other variable name.

```
PROC freq data   = &in_data;
          tables   &var        / noprint
             out = out_freq
         (rename =(&var = valu_&type) );

DATA freq_values;
     attrib valu_c length = $32
            valu_n length =   8;
set  &syslast;

PROC append data = &syslast
            base = out_data;
run;
```

slide 17

Notes: 1. Data structure may be as small as <1% of program statements. 2. Style guide, the naming convention for variables `valu-c` and `valu-n` depends on the parameter `type`.

polishing test program

This program shows the principle of allocating a macro variable – `proc_name` — to replace any set of two or more occurences of the same string.

```
options source2;
%let in_data   = sashelp.class;
%let proc_name = 'r14-freq.sas';

%let var  = sex;
%let type = c;
%include    &proc_name;

%let var  = age;
%let type = n;
%include    &proc_name;

proc print data = &syslast;
run;
```

slide 18

Notes: 1. Option `source2` is a global option; it has been moved from the `%include` statement. 2. Macro variable `proc-name` reduces typing and the need for find-and-replace. Compare to previous program, `r13-freq-test`.

output, first draft

```
Obs    valu_c    valu_n    COUNT    PERCENT

 1       F          .         9     47.3684
 2       M          .        10     52.6316
 3                 11         2     10.5263
 4                 12         5     26.3158
...
 7                 15         4     21.0526
 8                 16         1      5.2632
```

slide 19

Notes: Each row is from which variable?

---

polishing data structure

Add a variable name 'var' and `retain` the name of the variable.

```
PROC freq data   = &in_data;
          tables   &var        / noprint
             out = out_freq
         (rename =(&var = valu_&type) );

DATA freq_values;
     attrib var    length = $32
            valu_c length = $32
            valu_n length =   8;
     retain var    "&var";
set &syslast;

PROC append data = &syslast
            base = &out_data;
run;
```

slide 20

Notes: 1. The name of the variable, `var`, is placed first in the data structure. 2. Style guide: always end routine or subroutine with a `run;` statement.

---

polishing test program

```
options source2;
%let in_data   = sashelp.class;
%let out_data  =    work.out_data;
%let proc_name = 'r15-freq.sas';

%let var  = sex;
%let type = c;
%include    &proc_name;

%let var  = age;
%let type = n;
%include    &proc_name;

proc print data = &out_data;
run;
```

slide 21

---

output, 2nd rough draft

```
Obs   var   valu_c   valu_n   COUNT   PERCENT

 1    sex     F         .        9    47.3684
 2    sex     M         .       10    52.6316
 3    age              11        2    10.5263
 4    age              12        5    26.3158
 5    age              13        3    15.7895
 6    age              14        4    21.0526
 7    age              15        4    21.0526
 8    age              16        1     5.2632
```

slide 22

---

output, polished

```
PROC print data = &out_data;
           title3 &out_data;
           title4 &in_data ;
           by     name notsorted;
           id     name;
run;
```

slide 23

Notes: Consider using ODS for other destinations.

```
var   valu_c   valu_n   COUNT   PERCENT

sex     F         .        9    47.3684
        M         .       10    52.6316
age              11        2    10.5263
                 12        5    26.3158
                 13        3    15.7895
                 14        4    21.0526
                 15        4    21.0526
                 16        1     5.2632
```

slide 24

Notes: Information: double-digit percents indicate variable is discrete.

Top Down, Making List of Variable Names

---

echo data structure

This sql program lists the data structure in the log.

```
PROC sql; describe table sashelp.class;
          describe table dictionary.columns;
          quit;
```

r21-sql-describe.log

```
create table SASHELP.CLASS( label='Student Data')
   Name   char(8),
   Sex    char(1),
   Age    num,
   Height num,
   Weight num
```

Notes: We want a data set with this information.

---

sql dictionary columns

This is the data structure of the sql dictionary columns.

```
NOTE: SQL table Dictionary.Columns was created like:
   libname char(8)   label='Library Name',
   memname char(32)  label='Member Name',
   memtype char(8)   label='Member Type',
   name    char(32)  label='Column Name',
   type    char(4)   label='Column Type',
   length  num       label='Column Length',
   varnum  num       label='Column Number in Table',
```

Notes: Compare to `describe view sashelp.vcolumns;`

---

making list of variable names

This sql program makes a list of the variable names in a data set.

```
PROC sql; create table list_variables as
          select libname, memname, name, type
          from   dictionary.columns
          where  libname eq "%upcase(&libname)"
            and  memname eq "%upcase(&memname)";
          quit;
```

Notes: 1. Save this program as `make-list-vars-sql.sas`.

2. Notice the use of SAS naming conventions to promote readability and reuse.

---

subroutine for testing

This sql program echos the data structure to the log and prints the data set.

```
PROC sql;  describe table &syslast;
           quit;
PROC print data = &syslast;
           title3 &syslast;
run;
```

slide 28

Notes: 1. Programmers want to read the notes in the log produced by sql; users want to examine the output from the print procedure. 2. This is from `r12-freq` on page 5.

testing subroutine make-list

```
%let libname = sashelp;
%let memname = class;
%include 'r22-sql-make-list.sas';
%include 'describe.sas';
```

slide 29

a control data set

```
Obs    libname    memname    name      type

 1     SASHELP     CLASS     Name      char
 2     SASHELP     CLASS     Sex       char
 3     SASHELP     CLASS     Age       num
 4     SASHELP     CLASS     Height    num
 5     SASHELP     CLASS     Weight    num
```

slide 30

Processing the List

demo.1 of sql text

Notes: The first few lines of program `r31-sql-proc-text-demo.sas` are not shown on the slide.

```
%let libname = sashelp;
%let memname = class;
%include 'make-list-vars-sql.sas';

PROC sql; select 'prefix', name, 'etc', type, 'suffix'
          from   list_variables;
          quit;
```

r31-sql-proc-text-demo.lst

```
        Name             Type
----------------------------------------
prefix  Name         etc  char    suffix
prefix  Sex          etc  char    suffix
prefix  Age          etc  num     suffix
prefix  Height       etc  num     suffix
prefix  Weight       etc  num     suffix
```

slide 31

demo.2 of sql text

```
PROC sql; select ’%let name=’, name, ’;’
                ,’%let type=’, type, ’;’
          from   list_variables;
          quit;
```

r32-sql-proc-text-let.lst

```
            Name                     Type
--------------------------------------------
%let name=  Name        ;%let type=  char  ;
%let name=  Sex         ;%let type=  char  ;
%let name=  Age         ;%let type=  num   ;
%let name=  Height      ;%let type=  num   ;
%let name=  Weight      ;%let type=  num   ;
```

slide 32

Notes: 1. Test program, `r33-sql-proc-list-test`, is not shown in the slides.
2. Program `r22-sql-make-list` on page 11 is the predecessor of `make-list-vars-sql`.

calling routine

```
%let libname = sashelp;
%let memname = class;
%include ’make-list-vars-sql.sas’;

%let proc_name = ’put-name-type.sas’;
%include ’r33-sql-proc-list.sas’;
```

Notes: This test program, `put-name-type.sas`, is not shown in the slides.

```
%put note: &=name &=type;
```

called subroutine

```
PROC sql; select catt(’%let name=’, name, ’;’
                     ,’%let type=’, type, ’;’
                     ,"%include &proc_name ;")
          into  :proc_list   separated by ’ ’
          from   list_variables;
          quit;
&proc_list
```

r33-sql-proc-list.lst

```
%let name=Name  ;%let type=char;%include ’put-name-type.sas’;
%let name=Sex   ;%let type=char;%include ’put-name-type.sas’;
%let name=Age   ;%let type=num ;%include ’put-name-type.sas’;
%let name=Height;%let type=num ;%include ’put-name-type.sas’;
%let name=Weight;%let type=num ;%include ’put-name-type.sas’;
```

slide 33

Notes: Save `r33-sql-proc-list-test.sas` as `proc-list-vars.sas`, as shown on page 18.

using SAS names

This program has four parameters, `libname`, `memname`, `name` and `type`.

```
PROC freq data   = &libname..&memname;
          tables   &name       / noprint
             out = out_freq
         (rename =(&name = valu_&type) );

DATA freq_values;
     attrib name      length = $32
            valu_char length = $32
            valu_num  length =   8;
     retain name      "&name";
```

Notes: Naming conventions: programs for the frequency procedure have the prefix `r1`.

---

first review

```
%let in_data  = sashelp.class;
%let libname  = %scan(&in_data, 1,.);
%let memname  = %scan(&in_data,-1,.);
%let out_data = work.out_data;

%include ’make-list-vars-sql.sas’;

%let     proc_name  = ’r16-freq.sas’;
%include ’r33-sql-proc-list.sas’;

%include ’proc-print-out-data.sas’;
```

Notes: Compare to `proc-1-freq.sas`, shown on page 18.

```
          valu_
name      char       valu_num    COUNT    PERCENT

Name      Alfred         .          1      5.2632
          Alice          .          1      5.2632
...
          Thomas         .          1      5.2632
          William        .          1      5.2632

Sex       F              .          9     47.3684
          M              .         10     52.6316
```

Notes: For variable *Name* count equals 1 for all values; this means it is unique, the row-identifier.

```
Age                     11.0        2     10.5263
                        12.0        5     26.3158
...
                        15.0        4     21.0526
                        16.0        1      5.2632

Height                  51.3        1      5.2632
...
                        62.5        2     10.5263
...
                        66.5        2     10.5263
                        72.0        1      5.2632
```

Notes: 1. For classification variable Age, n-levels = 6, this means it is discrete.
2. For fact variable Height, many instances of count=1, this means it is continuous.

Change Notice, Adjustments for Summary Procedure

requirements, v2

from: for each variable in a data set, perform frequency
to: for any data set

1. character variables, frequency
2. numeric variables, summary

module	routine	subroutine
02-data-*		
proc-2-freq-smry		make-list-vars-sql
	proc-list-vars-subset	
	proc-freq	data-structure
	proc-summary	

Notes: Compare to first requirements on page 2.

proc summary

r51-smry.sas is not shown in the slides.

```
PROC summary data   =  sashelp.class;
             var       height;
             output
                out =   out_summary;
%include 'describe.sas';
```

Notes: Style guide: align keywords, equal sign, and arguments. Readability promotes reuse.

step.2 soft-coded

```
%let libname = sashelp;
%let memname = class;
%let name    = height;

PROC summary data   =  &libname..&memname;
             var       &name;
             output
                out =   out_summary;
%include 'describe.sas';
```

slide 39

data structure

```
create table work.out_summary
            (label='Summary Statistics')
   _TYPE_ num,
   _FREQ_ num,
   _STAT_ char(8),
   Height num
```

r52-smry-test.lst

```
Obs  _TYPE_  _FREQ_  _STAT_   Height

 1      0      19     N      19.0000
 2      0      19     MIN    51.3000
 3      0      19     MAX    72.0000
 4      0      19     MEAN   62.3368
 5      0      19     STD     5.1271
```

slide 40

Notes: r53-smry.sas is not shown in the slides.

```
PROC summary data   =  &libname..&memname;
             var       &name;
             output
                out =   out_summary
            (  drop =  _type_  _freq_
             rename = (_stat_ = valu_char
                       &name  = valu_num));
%include ’describe.sas’;
```

Note: Style, naming conventions: the suffix of the variables named valu-* is in (’char’,’num’), which is the value of the variable `type` in the control data set.

---

proc summary

```
PROC summary data   =  &libname..&memname;
             var       &name;
             output
                out =   out_summary
            (  drop =  _type_  _freq_
             rename = (_stat_ = valu_char
                       &name  = valu_num));
%include ’data-structure.sas’;
```

slide 41

Notes: Save `r54-smry.sas` as `proc-summary.sas;` see listing on page 20.

---

common subroutine

```
DATA standardized_data_structure
    (label = "freq/smry of &libname &memname");
     attrib name      length = $32
            valu_char length = $32 %* fragile! ;
            valu_num  length =   8
            count     length =   8
            percent   length =   8;
     retain name "&name";
set &syslast;

PROC append data = &syslast
            base = &out_data;
run;
```

slide 42

Notes: See `r16-freq.sas` on page 13. See listing on page 20.

---

proc freq, updated

```
PROC freq data   = &libname..&memname;
          tables   &name    / noprint
             out = out_freq
         (rename =(&name = valu_&type));

%include ’data-structure.sas’;
```

slide 43

Notes: Save `r17-freq.sas` as `proc-freq.sas,` see listing on page 20.

Process the Subset

processing a subset

```
PROC sql noprint;
         select catt('%let name=', name, ';'
                    ,'%let type=', type, ';'
                    ,"%include &proc_name ;")
         into  :proc_list separated by ' '
         from   list_variables
         where &proc_where;     *  <---<<<  ;
         quit;
&proc_list
```

slide 44

Notes: Save `r34-sql-proc-list-where.sas` as `proc-list-vars-subset.sas`, see listing on page 21.

main module

```
%let      proc_name  = 'proc-freq.sas';
%let      proc_where = type eq 'char';
%include 'proc-list-vars-subset.sas';

%let      proc_name  = 'proc-summary.sas';
%let      proc_where = type eq 'num';
%include 'proc-list-vars-subset.sas';

%include 'proc-print-out-data.sas';
```

slide 45

Notes: Save `r42-proc-all-both.sas` as `proc-2-freq-smry.sas`, see listing on page 19.

Next-to-Last Review

proc either

```
          valu_
name      char       valu_num    count    percent

Name      Alfred         .          1     5.2632
...
          William        .          1     5.2632

Sex       F              .          9    47.3684
          M              .         10    52.6316

Age       N           19.000       .       .
          MIN         11.000       .       .
          MAX         16.000       .       .
          MEAN        13.316       .       .
          STD          1.493       .       .

Height    N           19.000       .       .
          MIN         51.300       .       .
          MAX         72.000       .       .
          MEAN        62.337       .       .
          STD          5.127       .       .
...
```

slide 46

future

list	make list with proc contents process list with call execute	
character	add accumulating count, percent extract pattern, calculate max length	
numeric	add percentile, calculate n-std examine extremes	Grubbs Dixon, Tukey
variables	use cardinality ratio to identify continuous, discrete, unique	

slide 47

Notes: Daily Job Diary (DJD) is a record of tasks, accomplishments, bells and whistles to add, and other notes-to-self.

Summary

Conclusion

Dividing programs into three types: modules, routines and subroutines, makes developing and testing applications easier and faster.

The next step is to convert subroutines into macros, which is the topic of the next chapter.

Recommended Reading

batch :	Walsh [16]
design :	Brooks Jr. [2], Mythical Man-Month; Brooks Jr. [3], Design of Design; Fehd [9], Macro Design Ideas; Knuth [12], Literate Programming; Lidwell et al. [13], Principles of Design
DJD :	google: DJD Daily Job Diary and see the pdf from *nickleelectrical.com/wp-content/.../09/DAILY-JOB-DIARY-WEEK.pdf*
HIPO :	IBM [11], Hierarchical Input Process Output
list processing :	Fehd [8], Using Sql Select Into; Fehd and Carpenter [10], List Processing Basics
naming conventions :	Martin [14], Clean Code; McConnell [15], Code Complete
SmryEachVar :	Fehd [6], Data Review Macro FreqAll; Fehd [5], FreqLibname: A Data Review Routine; Fehd [7], SmryEachVar: A Data Review Routine
testing :	Fehd [4], Writing testing-aware programs; Bentley [1], Software Testing Fundamentals

Contact Information:

Ronald J. Fehd
mailto:Ron.Fehd.macro.maven@gmail.com
http://www.sascommunity.org/wiki/Ronald_J._Fehd

About the author:

education:	B.S. Computer Science, U/Hawaii,	1986
	SAS User Group conference attendee since	1989
experience:	programmer: 30+ years	
	author: 40+ SUG papers	
	sas.community.org wiki: 400+ pages	
SAS-L:	author: 7,000+ messages to SAS-L since	1997

Programs: **http://www.sascommunity.org/wiki/**

SmryEachVar ApDev Introduction

Trademarks

Program Listings, Summarize-Each-Variable Suite

Suite.1 Frequency of All Variables

module	routine	subroutine	lines
01-data-*			5
proc-1-freq			16
		make-list-vars-sql	6
	proc-list-vars		10
	proc-print-out-data		7
	proc-freq		9

`01-data-sashelp-class.sas`

```
%let  in_data = sashelp.class;
%let out_data = work.summary_each_var_class;

%include ’proc-1-freq.sas’;
```

`proc-1-freq.sas`

```
******** echo included statements to log?;
options source2;

**** split two-level name: in_data :: libref.data;
%let libname = %scan(&in_data, 1,.);
%let memname = %scan(&in_data,-1,.);
%put note: &=in_data &=libname &=memname;

******** create input list: control data set;
%include ’make-list-vars-sql.sas’;

******** call item processing routine;
%let     proc_name  = ’r16-freq.sas’;
%include ’proc-list-vars.sas’;

%include ’proc-print-out-data.sas’;
```

`make-list-vars-sql.sas`

```
PROC sql; create table list_variables as
          select libname, memname, name, type
          from   dictionary.columns
          where  libname eq "%upcase(&libname)"
            and  memname eq "%upcase(&memname)";
          quit;
```

`proc-list-vars.sas`

```
%put note: &=proc_name;
*fragile: max length sql cat* values=$char200.;
*fragile: max length mvar = 2**16 = 64K;

PROC sql noprint;
         select catt(’%let name=’, name, ’;’
                    ,’%let type=’, type, ’;’
                    ,"%include &proc_name ;"  )
         into  :process_list  separated by ’ ’
         from   list_variables;
         quit;
&process_list
```

Note: `proc-freq` for suite.1 is `r16-freq`, on page 13.

proc-print-out-data.sas

```
PROC print data = &out_data;
           title3 &out_data;
           title4 &in_data ;
           by     name notsorted;
           id     name;
run;
```

Suite.2 Frequency or Summary

module	routine	subroutine	lines
01-data-*			5
proc-2-freq-smry			22
		make-list-vars-sql	6
	proc-list-vars-subset		11
	proc-print-out-data		7
		data-structure	13
	proc-freq		9
	proc-summary		11

02-data-sashelp-heart.sas

```
%let  in_data = sashelp.heart;
%let out_data = work.summary_each_var_heart;

%include ’proc-2-freq-smry.sas’;
```

proc-2-freq-smry.sas

```
******** echo included statements to log?;
*options source2;

**** split two-level in-data :: libref.data;
%let libname = %scan(&in_data, 1,.);
%let memname = %scan(&in_data,-1,.);
%put note: &=in_data &=libname &=memname;

******** create input list: control data set;
%include ’make-list-vars-sql.sas’;

******** call item processing routines;
%let     proc_name  = ’proc-freq.sas’;
%let     proc_where = 1; * all? or only char?;
%let     proc_where = type eq ’char’;
%include ’proc-list-vars-subset.sas’;

%let     proc_name  = ’proc-summary.sas’;
%let     proc_where = type eq ’num’;
%include ’proc-list-vars-subset.sas’;

%include ’proc-print-out-data.sas’;
```

proc-list-vars-subset.sas

```
%put note: &=proc_name &=proc_where;
PROC sql noprint;
         select catt(’%let name=’, name, ’;’
                    ,’%let type=’, type, ’;’
                    ,"%include &proc_name ;")
         into  :_process_list  separated by ’ ’
         from   list_variables
         where &proc_where;       *  <---<<<  ;
         quit;
&_process_list
```

data-structure.sas

```
DATA standardized_data_structure
    (label = "freq/smry of &libname &memname");
     attrib name      length = $32
            valu_char length = $32 %* fragile! ;
            valu_num  length =   8
            count     length =   8
            percent   length =   8;
     retain name "&name";
set &syslast;

PROC append data = &syslast
            base = &out_data;
run;
```

proc-freq.sas

```
%put note: &=libname &=memname &=name &=type;

PROC freq data   = &libname .&memname;
          tables   &name    / noprint
          out    =  out_freq
         (rename =(&name = valu_&type));

%include ’data-structure.sas’;
```

proc-summary.sas

```
%put note: &=libname &=memname &=name &=type;

PROC summary data   =  &libname..&memname;
             var       &name;
             output
                out =   out_summary
            (  drop =  _type_  _freq_
             rename = (_stat_ = valu_char
                       &name  = valu_num));

%include ’data-structure.sas’;
```

Program Listings, testing programs

routine	subroutine	lines
	make-list-vars-sql-test	6
proc-list-vars-subset-test		14
proc-freq-test		9
proc-summary-test		6
	describe	7

make-list-vars-sql-test.sas

```
%let libname = sashelp;
%let memname = class;
%include ’make-list-vars-sql.sas’;
%include ’describe.sas’;
```

proc-freq-test.sas

```
%let libname  = sashelp;
%let memname  = class;
%let out_data = out_data_&memname;

%let name     = sex;
%let type     = char;
%include ’proc-freq.sas’;
%include ’describe.sas’;

%let name     = age;
%let type     = num;
%include ’proc-freq.sas’;
%include ’describe.sas’;
```

proc-list-vars-subset-test.sas

```
%let libname  = sashelp;
%let memname  = class;
%let out_data = work.summary_each_var_class;

%include ’make-list-vars-sql.sas’;
%let proc_name  = ’put-name-type.sas’;
%let proc_where = 1;
%include ’proc-list-vars-subset.sas’;
```

proc-summary-test.sas

```
%let libname  = sashelp;
%let memname  = class;
%let out_data = out_data_&memname;

%let name     = height;
%let type     = num;
%include ’proc-summary.sas’;
%include ’describe.sas’;

%let name     = weight;
%include ’proc-summary.sas’;
%include ’describe.sas’;
```

Bibliography

[1] John E. Bentley. Software testing fundamental — concepts, roles, and terminology. In *Proceedings of the 30th Annual SAS® Users Group International Conference*, 2005. URL `http://www2.sas.com/proceedings/sugi30/141-30.pdf`. 12 pp.; theory of testing.

[2] Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Addison-Wesley, 1995. URL `https://en.wikipedia.org/wiki/The_Mythical_Man-Month`. 322 pp., 19 chap., index: 14 pp.; theory and story based on development of IBM System/360 operating system.

[3] Frederick P. Brooks Jr. *The Design of Design*. Addison-Wesley, 2010. URL `http://www.pearsonhighered.com/educator/product/Design-of-Design-The-Essays-from-a-Computer-Scientist/9780201362985.page`. 421 pp., 28 chap., index: 20 pp.; history of the design process in many disciplines, besides software.

[4] Ronald J. Fehd. Writing testing-aware programs that self-report when testing options are true. In *NorthEast SAS Users Group Annual Conference Proceedings*, 2007. URL `http://www.nesug.org/Proceedings/nesug07/cc/cc12.pdf`. Coders' Corner, 20 pp.; topics: options used while testing: echoauto, mprint, source2, verbose; variable testing in data step or macros; call execute; references.

[5] Ronald J. Fehd. FreqLibname: a data review routine for all memnames in a libname. In *NorthEast SAS Users Group Annual Conference Proceedings*, 2007. URL `http://www.nesug.org/proceedings/nesug07/cc/cc11.pdf`. Coders' Corner, 22 pp.; topics: comments, documentation, quality, theory, replacing macros with call execute of parameterized include files, saving procs freq and summary output data set; info: complete test suite of modules, routines, and subroutines, getting mode from proc freq.

[6] Ronald J. Fehd. Journeymen's tools: Data review macro FreqAll — using Proc SQL list processing with Dictionary.Columns to eliminate macro do loops. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL `http://www2.sas.com/proceedings/forum2007/028-2007.pdf`. Coder's Corner, 10 pp.; attributes, dictionary.columns, metadata, proc append, proc freq, proc sql, program header; bibliography.

[7] Ronald J. Fehd. SmryEachVar: A data-review routine for all data sets in a libref. In *SAS Global Forum Annual Conference Proceedings*, 2008. URL `http://www2.sas.com/proceedings/forum2008/003-2008.pdf`. Applications Development, 24 pp.; call execute, data review, data structure, dynamic programming, list processing, parameterized includes, utilities (writattr, writvalu) to repair missing elements in data structure; best contributed paper in ApDev.

[8] Ronald J. Fehd. How to use proc SQL select into for list processing. In *South Central SAS Users Group Annual Conference Proceedings*, 2010. URL `http://analytics.ncsu.edu/sesug/2010/HOW06.Fehd.pdf`. Hands On Workshop, 40 pp.; topics: writing constant text, and macro calls, using macro %do loops; references.

[9] Ronald J. Fehd. Macro design ideas: Theory, template, practice. In *MidWest SAS Users Group Annual Conference Proceedings*, 2013. URL `http://www.mwsug.org/proceedings/2013/00/MWSUG-2013-0002.pdf`. 21 pp.; topics: logic, quality assurance, testing, style guide, documentation, bibliography.

[10] Ronald J. Fehd and Art Carpenter. List processing basics: Creating and using lists of macro variables. In *SAS Global Forum Annual Conference Proceedings*, 2007. URL `http://www2.sas.com/proceedings/forum2007/113-2007.pdf`. Hands On Workshop, 20 pp.; comparison of methods: making and iterating macro arrays, scanning macro variable, writing calls to macro variable, write to file then include, call execute; using macro function nrstr in call execute argument; 11 examples, bibliography.

[11] IBM, editor. *HIPO — A Design Aid and Documentation Technique*. IBM Corporation, White Plains, NY, 1974. URL `http://en.wikipedia.org/wiki/HIPO`. Publication Number GC20-1851.

[12] Donald E. Knuth. *Literate Programming*. CSLI, Stanford, CA, USA, 1992. URL `http://en.wikipedia.org/wiki/Literate_programming`. 368 pp., 12 chap., index: 10 pp.

[13] William Lidwell, Kritina Holden, and Jill Butler. *Universal Principles of Design*. Rockport Publishers, Inc, 2010. URL `http://stuffcreators.com/upod/`. 272 pp., 125 chap., index: 5 pp.; each chapter is two pages, each explains core building blocks of design theory.

[14] Robert C. Martin. *Clean Code*. Pearson Education, Boston, MA, USA, 2009. URL `http://www.pearsonhighered.com/educator/product/Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/9780132350884.page`. 431 pp., 17 chap., index: 19 pp.; subtitle: handbook of Agile software craftsmanship.

[15] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction, 2e*. Microsoft Press, Redmond, WA, USA, 2004. URL `http://cc2e.com/`. 960 pp., 35 chap., index: 30 pp.

[16] Irina Walsh. Pros and cons of interactive SAS(R) mode vs. batch mode. In *Proceedings of the Western Users of SAS Software Annual Conference*, 2010. URL `http://www.lexjansen.com/wuss/2010/coders/2937_4_0_COD_Walsh.pdf`. Coders' Corner, 9 pp.; autoexec, naming conventions: folders, ODS rtf.