MWSUG 2025 – Paper PL-032

Last Observation Carried Forward (LOCF) in Longitudinal Clinical Studies: Adopting a Functional Approach to Imputing Missing Values Using PROC FCMP, the SAS® Function Compiler

Troy Martin Hughes

ABSTRACT

Last observation carried forward (LOCF) is a ubiquitous method of imputing missing values in longitudinal studies, and is commonly implemented when a subject (i.e., a patient or participant) misses a scheduled visit, and data cannot be collected (or generated). In general, the last "valid" value from a previous visit is retained for the later visit on which the data could not be obtained, and this conservative estimation succeeds in cases where the actual value would have been little changed. Nuanced criteria may stipulate which prior values count as "valid" (e.g., after the start of treatment) as well as for how long (e.g., how many days, visit weeks, consecutive missed visits) a value can be used to impute other values. Given these complexities, LOCF solutions implemented in SAS® historically adopt a *procedural* approach, and often require multiple DATA steps and/or procedures to impute data both across observations and within subjects. Conceptually, however, a *functional* approach can be envisioned in which LOCF could be calculated using a function call—that is, delivering the same functionality through a single line of code while hiding (abstracting) the complexity of the calculation inside the function's definition. The FCMP procedure can deliver this functionality, enabling SAS practitioners to build user-defined functions—even those that perform inter-observation calculations—and this text demonstrates a user-defined subroutine that dynamically calculates LOCF while relying on CDISC and ADaM standards, data structures, and nomenclature. The software design concepts herein are adapted from the renowned SAS Press book: *PROC FCMP User-Defined Functions: An Introduction to the SAS® Function Compiler*. (Hughes, 2024)

LAST CREATININE CARRIED FORWARD

Creatinine ($C_4H_7N_3O$) is one of the primary compounds measured in blood or urine samples to detect kidney dysfunction, with higher levels of creatinine—*on average, and with numerous caveats such as the influence of muscle mass*—indicating poorer kidney performance. Consider a clinical trial that evaluates, among other metrics, the creatinine level of its subjects through blood draws during each visit. In a perfect study, no subjects would drop out and all subjects would faithfully attend all scheduled visits (at protocol-specified times). But the realities and complexities of life afford that subjects *do* leave studies earlier, and even those subjects who complete study protocols may unavoidably miss some scheduled visits. In other cases, a subject may not miss the visit, but a scheduled sample may not be taken, or it may be insufficient or invalid for a host of reasons.

Thus, regardless of why study data are omitted, this common occurrence requires that study design plan for and respond to missing data through published, standardized methods, and imputation—whether using last observation carried forward (LOCF) or other methods—provides data for those missed visits. By imputing a value from a prior visit, a "conservative" estimate replaces the missing value, with the understanding that this assumes the subject's data have not appreciably changed since that visit.

Consider a study that commenced on January 6, 2025, and in which subjects have visits scheduled for each Monday for a set number of weeks, only eight weeks of which are demonstrated in this scenario:

```
WK1 = 06JAN2025 (pre-treatment)
WK2 = 13JAN2025 (pre-treatment)
WK3 = 20JAN2025 (first exposure to study drug)
WK4 = 27JAN2025
WK5 = 03FEB2025
WK6 = 10FEB2025
WK7 = 17FEB2025
WK8 = 24FEB2025 (study completion)
```

An extremely simplified Basic Data Structure (BDS) is created, which records only the creatinine levels (recorded as mg/dL and coded as CREAT) for subjects. The following DATA step builds the BDS_RAW data set, which records the weekly creatinine levels for Subjects 0001 and 0002:

```
data bds_raw;
   length SUBJID $11 PARAMCD $8 ADT 8 AVISIT $19 AVAL 8;
   format SUBJID $11. PARAMCD $8. ADT yymmdd10. AVISIT $19. AVAL 8.1;
   input @1 SUBJID $11. @12 PARAMCD $8. @20 ADT yymmdd10. @30 AVISIT $19. @49 AVAL 8.1;
   datalines;
0001       CREAT   2025-01-06Week 1             61.2
0001       CREAT   2025-01-13Week 2             63.5
0001       CREAT   2025-01-20Week 3             64.9
0001       CREAT   2025-01-27Week 4             .
0001       CREAT   2025-02-03Week 5             65.7
0001       CREAT   2025-02-10Week 6             .
0001       CREAT   2025-02-17Week 7             .
0001       CREAT   2025-02-24Week 8             65.0
0002       CREAT   2025-01-06Week 1             .
0002       CREAT   2025-01-13Week 2             .
0002       CREAT   2025-01-20Week 3             82.1
0002       CREAT   2025-01-27Week 4             85.6
0002       CREAT   2025-02-03Week 5             .
0002       CREAT   2025-02-10Week 6             .
0002       CREAT   2025-02-17Week 7             .
0002       CREAT   2025-02-24Week 8             80.9
;
```

The DATA step creates the BDS_RAW data set.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL
1	0001	CREAT	2025-01-06	Week 1	61.2
2	0001	CREAT	2025-01-13	Week 2	63.5
3	0001	CREAT	2025-01-20	Week 3	64.9
4	0001	CREAT	2025-01-27	Week 4	.
5	0001	CREAT	2025-02-03	Week 5	65.7
6	0001	CREAT	2025-02-10	Week 6	.
7	0001	CREAT	2025-02-17	Week 7	.
8	0001	CREAT	2025-02-24	Week 8	65.0
9	0002	CREAT	2025-01-06	Week 1	.
10	0002	CREAT	2025-01-13	Week 2	.
11	0002	CREAT	2025-01-20	Week 3	82.1
12	0002	CREAT	2025-01-27	Week 4	85.6
13	0002	CREAT	2025-02-03	Week 5	.
14	0002	CREAT	2025-02-10	Week 6	.
15	0002	CREAT	2025-02-17	Week 7	.
16	0002	CREAT	2025-02-24	Week 8	80.9

Note that Subject 0001 did not have the scheduled assessment during weeks 4, 6, and 7, and that Subject 0002 did not have the scheduled assessment during weeks 1, 2, 5, 6, and 7, as indicated by the missing values in the AVAL variable. These missing values could have resulted from missed visits, invalid or insufficient blood or urine draws, or a host of other reasons.

Studies maintain various business rules that specify which observations can be carried forward—and which cannot. One common factor is the length of time from actual data collection (i.e., attended visit) to the imputation (i.e., missed visit), and cut-off criteria are often established. For example, a value might not be able to be imputed if the last valid measurement was collected more than four weeks ago. Another common factor is the number of consecutive visits that are missed. For example, in this scenario, a study protocol can be specified that: *Two consecutive visits can be imputed, but not a third consecutively missed visit*.

Both of these imputation selection criteria (i.e., with and without the three-visit rule) are demonstrated in this text—first procedurally (using the DATA step) and then functionally (using the FCMP procedure).

In this scenario, Subject 0001 only misses two consecutive assessments, so all missed values can be imputed from previous visits. However, as Subject 0002 missed three consecutive assessments, Week 7 cannot be imputed.

The following DATA step builds the BDS_VALIDATE data set, which contains both the imputed values (in the AVAL variable) and the indication of which observations were imputed (demonstrated by "LOCF" in the DTYPE variable); this data set is only used to validate that subsequent procedural and functional methods (of imputing LOCF) are producing accurate results:

```
*/
NOTE that DTYPE must be added, and indicates when imputation occurs
NOTE that SORTEDBY simulates a sorted data set (required because BY is later used)
*/
data bds_validate (sortedby=SUBJID ADT);
   length SUBJID $11 PARAMCD $8 ADT 8 AVISIT $19 AVAL 8 DTYPE $8;
   format SUBJID $11. PARAMCD $8. ADT yymmdd10. AVISIT $19. AVAL 8.1 DTYPE $8.;
   input @1 SUBJID $11. @12 PARAMCD $8. @20 ADT yymmdd10. @30 AVISIT $19. @49 AVAL 8.1
      @57 DTYPE $8.;
   datalines;
0001       CREAT   2025-01-06Week 1             61.2
0001       CREAT   2025-01-13Week 2             63.5
0001       CREAT   2025-01-20Week 3             64.9
0001       CREAT   2025-01-27Week 4             64.9    LOCF
0001       CREAT   2025-02-03Week 5             65.7
0001       CREAT   2025-02-10Week 6             65.7    LOCF
0001       CREAT   2025-02-17Week 7             65.7    LOCF
0001       CREAT   2025-02-24Week 8             65.0
0002       CREAT   2025-01-06Week 1             .
0002       CREAT   2025-01-13Week 2             .
0002       CREAT   2025-01-20Week 3             82.1
0002       CREAT   2025-01-27Week 4             85.6
0002       CREAT   2025-02-03Week 5             85.6    LOCF
0002       CREAT   2025-02-10Week 6             85.6    LOCF
0002       CREAT   2025-02-17Week 7             .
0002       CREAT   2025-02-24Week 8             80.9
;
```

The DATA step creates the BDS_VALIDATE data set.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL	DTYPE
1	0001	CREAT	2025-01-06	Week 1	61.2	
2	0001	CREAT	2025-01-13	Week 2	63.5	
3	0001	CREAT	2025-01-20	Week 3	64.9	
4	0001	CREAT	2025-01-27	Week 4	64.9	LOCF
5	0001	CREAT	2025-02-03	Week 5	65.7	
6	0001	CREAT	2025-02-10	Week 6	65.7	LOCF
7	0001	CREAT	2025-02-17	Week 7	65.7	LOCF
8	0001	CREAT	2025-02-24	Week 8	65.0	
9	0002	CREAT	2025-01-06	Week 1	.	
10	0002	CREAT	2025-01-13	Week 2	.	
11	0002	CREAT	2025-01-20	Week 3	82.1	
12	0002	CREAT	2025-01-27	Week 4	85.6	
13	0002	CREAT	2025-02-03	Week 5	85.6	LOCF
14	0002	CREAT	2025-02-10	Week 6	85.6	LOCF
15	0002	CREAT	2025-02-17	Week 7	.	
16	0002	CREAT	2025-02-24	Week 8	80.9	

The DTYPE variable signifies a special value that does not appear in the SDTM data set. Here, DTYPE conveys that a value has been imputed, as opposed to measured, and defines the method of imputation—LOCF. Note that weeks 1 and 2 for Subject 0002 cannot be imputed because no prior assessments occurred from which subsequent values could be imputed.

PROCEDURALLY IMPUTING LOCF

Fundamentally, LOCF always requires an inter-observation calculation because a past value is carried forward to impute a subsequent missing value. And perhaps this is why the myriad published approaches to calculating LOCF always adopt a procedural approach that combines multiple DATA step statements, and often multiple DATA STEPs, procedures, or SAS macros. Although this undue complexity can be avoided through functional approaches (demonstrated subsequently), two procedural solutions are first shown for comparison.

Within DATA step approaches to calculating LOCF, the RETAIN statement is commonly employed to retain a prior value for possible later imputation.

For example, the following DATA step imputes all missing AVAL values; however, note that in this first version, no accommodation is made for imputation protocol rules, such as the requirement that only two consecutive assessments can be imputed:

```
data bds_retain_test_v1 (drop=_aval_last);
   set bds_raw;
   by SUBJID ADT;
   length DTYPE $8 _aval_last 8 _aval_last 8.1;
   if missing(AVAL) and ^first.SUBJID and ^missing(_aval_last) then do;
      AVAL = _aval_last;
      DTYPE = 'LOCF';
      end;
   _aval_last = AVAL;
   retain _aval_last;
run;
```

The DATA step creates the BDS_RETAIN_TEST_V1 data set, which differs from BDS_VALIDATE because week 7 for Subject 0002 has been imputed—despite it being the third consecutively missed assessment.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL	DTYPE
1	0001	CREAT	2025-01-06	Week 1	61.2	
2	0001	CREAT	2025-01-13	Week 2	63.5	
3	0001	CREAT	2025-01-20	Week 3	64.9	
4	0001	CREAT	2025-01-27	Week 4	64.9	LOCF
5	0001	CREAT	2025-02-03	Week 5	65.7	
6	0001	CREAT	2025-02-10	Week 6	65.7	LOCF
7	0001	CREAT	2025-02-17	Week 7	65.7	LOCF
8	0001	CREAT	2025-02-24	Week 8	65.0	
9	0002	CREAT	2025-01-06	Week 1	.	
10	0002	CREAT	2025-01-13	Week 2	.	
11	0002	CREAT	2025-01-20	Week 3	82.1	
12	0002	CREAT	2025-01-27	Week 4	85.6	
13	0002	CREAT	2025-02-03	Week 5	85.6	LOCF
14	0002	CREAT	2025-02-10	Week 6	85.6	LOCF
15	0002	CREAT	2025-02-17	Week 7	85.6	LOCF
16	0002	CREAT	2025-02-24	Week 8	80.9	

To include the study protocol requirement that no more than two consecutively missed visits can be imputed, a counter variable (_CNT_LOCF) can be incremented each time a consecutively missed

assessment is encountered, as well as reset to zero each time a new subject or a non-missed assessment is encountered. The following DATA step again leverages RETAIN to retain values across observations:

```
data bds_retain_test_v2 (drop=_aval_last _dtype_last _cnt_locf);
   set bds_raw;
   by SUBJID ADT;
   length DTYPE _dtype_last $8 _aval_last _cnt_locf 8;
   format _aval_last 8.1;
   if first.SUBJID then do;
      _cnt_locf = 0;
      _dtype_last = '';
      end;
   if missing(AVAL) and ^first.SUBJID and ^missing(_aval_last) then do;
      if _cnt_locf < 2 then do;
         AVAL = _aval_last;
         DTYPE = 'LOCF';
         if (_cnt_locf = 0) or (_dtype_last = 'LOCF') then _cnt_locf +1;
         end;
      end;
   if DTYPE = '' then _cnt_locf = 0;
   _aval_last = AVAL;
   _dtype_last = DTYPE;
   retain _aval_last _dtype_last _cnt_locf;
run;
```

The DATA step creates the BDS_RETAIN_TEST_V2 data set, which matches the BDS_VALIDATE data set that was previously created. However, note the added complexity that was forced upon the DATA step because study rules were modified—and all of this clutter only exists to impute a single variable! Thus, imagine more realistic scenarios in which multiple values are being imputed, and possibly through a host of imputation methods in addition to LOCF. It is far better to place all of these imputation rules inside of user-defined functions or subroutines than to expose the DATA step to this complexity.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL	DTYPE
1	0001	CREAT	2025-01-06	Week 1	61.2	
2	0001	CREAT	2025-01-13	Week 2	63.5	
3	0001	CREAT	2025-01-20	Week 3	64.9	
4	0001	CREAT	2025-01-27	Week 4	64.9	LOCF
5	0001	CREAT	2025-02-03	Week 5	65.7	
6	0001	CREAT	2025-02-10	Week 6	65.7	LOCF
7	0001	CREAT	2025-02-17	Week 7	65.7	LOCF
8	0001	CREAT	2025-02-24	Week 8	65.0	
9	0002	CREAT	2025-01-06	Week 1	.	
10	0002	CREAT	2025-01-13	Week 2	.	
11	0002	CREAT	2025-01-20	Week 3	82.1	
12	0002	CREAT	2025-01-27	Week 4	85.6	
13	0002	CREAT	2025-02-03	Week 5	85.6	LOCF
14	0002	CREAT	2025-02-10	Week 6	85.6	LOCF
15	0002	CREAT	2025-02-17	Week 7	.	
16	0002	CREAT	2025-02-24	Week 8	80.9	

Numerous functionally equivalent methods could be engineered to deliver this functionality procedurally, as well as to expand the business rules for LOCF imputation to incorporate additional study requirements. However, despite the functionality of this and other methods, the common detractor of all procedural approaches is the complexity of the LOCF calculation itself, which increasingly obfuscates code readability as business rules themselves increase in complexity. A *functional*—as opposed to *procedural*—approach overcomes this arguable design flaw, as introduced and demonstrated in the next section.

FUNCTIONALLY IMPUTING LOCF

A *functional* software design approach does not speak to software that functions (i.e., that the software works…although yes, your software also should work!) but rather that calculations or transformations are performed within a single statement—that is, conveyed through a function as opposed to multiple lines of code. The FCMP procedure (aka, the SAS Function Compiler) empowers SAS practitioners to design user-defined functions (and subroutines) in Base SAS, to compile and save these functions, and to call them subsequently and repeatedly. This design approach facilitates software reusability, in which a user-defined function can be built, tested, and documented only once, but used thereafter in perpetuity.

The following FCMP procedure defines the LOCF_V1 subroutine, which is functionally equivalent to the first procedural method previously demonstrated—that is, it ignores the study protocol that only two consecutively missed visits can be imputed:

```
proc fcmp outlib=work.funcs.locf;
   subroutine locf_v1(sub_id $, val_num, locf_method $);
      outargs val_num, locf_method;
      length _sub_last $11 _aval_last 8;
      static _sub_last '';
      static _aval_last .;
      if missing(val_num) and (sub_id = _sub_last) and ^missing(_aval_last) then do;
         val_num = _aval_last;
         locf_method = 'LOCF';
         end;
      _sub_last = sub_id;
      _aval_last = val_num;
      endsub;
quit;
```

The SUBROUTINE statement declares and names the subroutine, and declares the parameters to which arguments will be *bound* when the subroutine is called from a DATA step. *Binding* a value to a parameter effectively transfers data from the DATA step to the subroutine (or function) when it is called. Character parameters must have a trailing $ whereas numeric parameters do not.

The OUTARGS statement enumerates parameters that will be *passed by reference* rather than *passed by value*, the latter of which occurs by default for all parameters not listed in the OUTARGS statement. That is, the subroutine will have the ability to modify two arguments (corresponding to the VAL_NUM and LOCF_METHOD parameters) in the *calling program*, the DATA step that calls the subroutine. OUTARGS is required in this instance because the subroutine must set both the imputed value and the “LOCF” marker in the DTYPE variable. Had only one value needed to be modified, a function could have been designed to return the single value using the RETURN statement.

The STATIC statement causes the subroutine to retain the specified variables across subroutine (or function) calls, and can optionally initialize the variable to a specified value the first time the subroutine is called. For example, the first time LOCF_V1 is called in a DATA step, _SUB_LAST will be initialized to a missing character value, which will persist—even across subsequent subroutine calls in the same DATA step—until the value is updated inside the subroutine. In this sense, just as the RETAIN statement enables a value to be retained from one observation to the next inside the DATA step, the STATIC statement enables a value to be retained across subsequent function calls made from inside the same DATA step.

The remaining logic inside the LOCF_V1 subroutine is identical to that of the corresponding DATA step that was previously demonstrated, and imputes an unlimited number of consecutively missed assessments.

The following DATA step (prefaced by the CMPLIB option, which identifies where the LOCF_V1 subroutine has been compiled and saved) builds the BDS_FCMP_TEST_V1 data set:

```
options cmplib=work.funcs;

data bds_fcmp_test_v1;
   set bds_raw;
   by SUBJID ADT;
   length DTYPE $8;
   call missing(DTYPE);
```

```
   call locf_v1(SUBJID, AVAL, DTYPE);
run;
```

The MISSING subroutine (aka, CALL MISSING) is optional, and eliminates the pesky “NOTE: Variable DTYPE is uninitialized” message from appearing in the SAS log.

User-defined subroutines must be called using the CALL statement, whereas functions do not require CALL. For this reason, and because user-defined functions return a value, whereas subroutines cannot, functions can be used in direct assignment (to initialize a variable in the DATA step), whereas subroutines cannot. That is, the OUTARGS statement is the only method for a subroutine to communicate back to the DATA step that called it, whereas a function can communicate either through OUTARGS or the RETURN statement.

Thus, the LOCF_V1 subroutine passes the value of SUBJID to the subroutine, where it binds to the SUB_ID parameter. Arguments are passed positionally (as opposed to by name) and must be passed in the order in which they are declared (as parameters) inside the function or subroutine definition. Thus, the value of AVAL is bound to the VAL_NUM parameter and the value of DTYPE (which will always be missing when the subroutine is called) is bound to the LOCF_METHOD parameter. Passing these three arguments to the subroutine enables it to *use* their values inside the subroutine and, in the case of parameters specified in the optional OUTARGS statement, to *modify* the values (in the calling DATA step).

Using a functional approach, the DATA step creates the BDS_FCMP_TEST_V1 data set, which is identical to the BDS_RETAIN_TEST_V1 data set previously produced procedurally.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL	DTYPE
1	0001	CREAT	2025-01-06	Week 1	61.2	
2	0001	CREAT	2025-01-13	Week 2	63.5	
3	0001	CREAT	2025-01-20	Week 3	64.9	
4	0001	CREAT	2025-01-27	Week 4	64.9	LOCF
5	0001	CREAT	2025-02-03	Week 5	65.7	
6	0001	CREAT	2025-02-10	Week 6	65.7	LOCF
7	0001	CREAT	2025-02-17	Week 7	65.7	LOCF
8	0001	CREAT	2025-02-24	Week 8	65.0	
9	0002	CREAT	2025-01-06	Week 1	.	
10	0002	CREAT	2025-01-13	Week 2	.	
11	0002	CREAT	2025-01-20	Week 3	82.1	
12	0002	CREAT	2025-01-27	Week 4	85.6	
13	0002	CREAT	2025-02-03	Week 5	85.6	LOCF
14	0002	CREAT	2025-02-10	Week 6	85.6	LOCF
15	0002	CREAT	2025-02-17	Week 7	85.6	LOCF
16	0002	CREAT	2025-02-24	Week 8	80.9	

However, note the significantly improved readability in the DATA step adopting the functional approach. After DTYPE is declared, a single line of code is required to call the subroutine and generate both the AVAL and DTYPE values.

Next, a second subroutine (LOCF_V2) is designed, which now incorporates the study protocol requirements that only two consecutively missed assessments can have their missing values imputed:

```
proc fcmp outlib=work.funcs.locf;
   subroutine locf_v2(sub_id $, val_num, locf_method $);
      outargs val_num, locf_method;
      length _sub_last $11 _aval_last 8 _dtype_last $8 _cnt_locf 8;
      static _sub_last '';
      static _aval_last .;
      static _dtype_last '';
      static _cnt_locf 0;
      if missing(val_num) and (sub_id = _sub_last) and ^missing(_aval_last) then do;
```

```
         if _cnt_locf <2 then do;
            val_num = _aval_last;
            locf_method = 'LOCF';
            if (_cnt_locf = 0) or (_dtype_last = 'LOCF') then _cnt_locf +1;
            end;
         end;
      if locf_method = '' then _cnt_locf = 0;
      _sub_last = sub_id;
      _aval_last = val_num;
      _dtype_last = locf_method;
      endsub;
quit;
```

As before, the logic of the procedural approach is identical to the logic implemented in the functional approach that defines the LOCF_V2 subroutine; only subtle syntactical changes are required to execute this logic within the FCMP procedure. For example, OUTARGS ensures that the VAL_NUM and LOCF_METHOD variables are available in the calling DATA step. The STATIC statements retain the previous subject ID, the previous AVAL value, the previous DTYPE value, and the running count of consecutive LOCF imputations—similar to the reliance on the RETAIN statement in the comparable procedural approach implemented in the DATA step.

The following DATA step calls the LOCF_V2 subroutine to calculate AVAL and DTYPE for those observations for which AVAL is missing, and now incorporates the study requirement that no more than two consecutively missed assessments can be imputed:

```
data bds_fcmp_test_v2;
   set bds_raw;
   by SUBJID ADT;
   length DTYPE $8;
   call missing(DTYPE);
   call locf_v2(SUBJID, AVAL, DTYPE);
run;
```

The DATA step creates the BDS_FCMP_TEST_V2 data set, which is identical to the BDS_RETAIN_V2 data set and, most importantly, to the BDS_VALIDATE data set that is used to demonstrate that the correct solution has been achieved.

	SUBJID	PARAMCD	ADT	AVISIT	AVAL	DTYPE
1	0001	CREAT	2025-01-06	Week 1	61.2	
2	0001	CREAT	2025-01-13	Week 2	63.5	
3	0001	CREAT	2025-01-20	Week 3	64.9	
4	0001	CREAT	2025-01-27	Week 4	64.9	LOCF
5	0001	CREAT	2025-02-03	Week 5	65.7	
6	0001	CREAT	2025-02-10	Week 6	65.7	LOCF
7	0001	CREAT	2025-02-17	Week 7	65.7	LOCF
8	0001	CREAT	2025-02-24	Week 8	65.0	
9	0002	CREAT	2025-01-06	Week 1	.	
10	0002	CREAT	2025-01-13	Week 2	.	
11	0002	CREAT	2025-01-20	Week 3	82.1	
12	0002	CREAT	2025-01-27	Week 4	85.6	
13	0002	CREAT	2025-02-03	Week 5	85.6	LOCF
14	0002	CREAT	2025-02-10	Week 6	85.6	LOCF
15	0002	CREAT	2025-02-17	Week 7	.	
16	0002	CREAT	2025-02-24	Week 8	80.9	

Note the simplicity of functional software design—that a 7-line DATA step is now exceling and generating identical results to the prior 21-line DATA step, which unnecessarily used a procedural approach! At this point, several clear advantages of adopting a functional approach to LOCF should be salient, and this

software design approach can be generalized to other imputation methods and, in fact, to other calculations and transformations in general.

First, the readability of the DATA step in which LOCF is being calculated is not diminished by the business rules (i.e., study protocol requirements) that specify how LOCF must be calculated. That is, through procedural abstraction, all of this complexity is hidden from SAS practitioners calling the subroutine, and instead maintained in the subroutine definition itself—inside the FCMP procedure.

Second, as the complexity of business rules increases, such as adding the requirement that no more than two consecutively missed visits can have their values imputed, the corresponding complexity of the DATA step calling the subroutine does not increase. That is, the DATA step that called LOCF_V2 is no more complex than the DATA step that had previously called LOCF_V1, and procedural abstraction is again to thank for this. And to be clear, the author advocates for *procedural abstraction* in *functional software design* to replace *procedural software design.*

Third, note that despite the added complexity of temporary variables needing to be declared, retained, and maintained to perform the necessary calculation of LOCF, all of these variables exist solely within the subroutine—they neither convolute the DATA step logic nor need to be explicitly dropped using DROP.

Fourth, and finally, as study requirements may change over time, these changes often can be made solely inside the function or subroutine definition, without having to burden the DATA step with modifications. Thus, SAS practitioners can continue to use and call core user-defined functions and subroutines without needing to change the code that calls them—even as the functionality of those functions or subroutines is expanded or as the performance thereof is increased.

CONCLUSION

Last observation carried forward (LOCF) is a ubiquitous method of imputing data and is commonly employed in clinical trials and other longitudinal studies in which subjects might miss visits or assessments, resulting in missing data. All published approaches to calculating LOCF using SAS software—insofar as the author could identify—unfortunately adopt a procedural approach in which the *how* (granular details) of LOCF calculation unnecessarily clutters the *what* (inputs and outputs) of the calculation. In adopting functional software design, the solutions in this text instead rely on procedural abstraction to hide these complexities inside the subroutine (or function) definitions, which are built and compiled utilizing the FCMP procedure, the SAS Function Compiler. This design paradigm engenders far more readable, maintainable, reusable SAS software—static software quality attributes that should be valued both in software requirements and the resultant SAS programs that are built from them. (Hughes, Data Analytic Development: Dimensions of Software Quality, 2016) And, with an appreciation for this enhanced software quality, both the efficiency and effectiveness of software—and the software development environment—are improved.

ACKNOWLEDGEMENTS

The author wishes to thank the informative, inspirational, indefatigable, indispensable Richann Jean Watson for her assistance and insight in review of this manuscript.

REFERENCES

Hughes, T. M. (2016). *Data Analytic Development: Dimensions of Software Quality.* Hoboken, NJ: John Wiley and Sons.

Hughes, T. M. (2024). *PROC FCMP User-Defined Functions: An Introduction to the SAS® Function Compiler.* Cary, NC: SAS Press.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com