

Getting “Func”y with SAS 9.2



Michael Matthews
Insurance Australia Group
SNUG (SAS NSW Users Group) Q4 2008

Overview

SAS Software contains a powerful programming language, however until version 9.2 it has been lacking in some of the functionality provided by many other languages.

The ability to define discrete functions and subroutines is available in most programming languages, yet the techniques for implementing functions in the Data Step have traditionally been workarounds with inherent dangers.

This paper looks at two new SAS procedures FCMP and PROTO along with the traditional techniques, creating various implementations of the Factorial function as a basic example.

SAS Macro

The SAS Macro language has many uses, and is often used to simulate the concept of a function.

In the following macro the data step variable *i* is used for internal looping within the macro, while the Data Step requests the calculation of factorial 1 through 5.

```
%macro fact2(n,out);
  &out=1;
  do i=2 to &n;
    &out=&out*i;
  end;
%mend;

data test;
  do j=1 to 5;
    %fact2(j,out);
  output;
end;
run;
```

The macro above produces the correct output (see below), however the variable *i*, which does not appear to exist in the Data Step code, appears in the output. This could be removed with a DROP statement, however the interactions can be more serious.

VIEWTABLE: WORK.TEST				
	j	out	i	
1	1	1	1	2
2	2	2	2	3
3	3	6	4	4
4	4	24	6	5
5	5	120	6	6

If the Data Step loop also uses a variable *i* then there will be unexpected interactions producing incorrect results. This can be difficult to locate, and changing the macro may create new issues.

```
%macro fact2(n,out);
  &out=1;
  do i=2 to &n;
    &out=&out*i;
  end;
%mend;
data test;
  do i=1 to 5;
    %fact2(i,out);
  output;
end;
run;
```

Note that there are less observations in the dataset with incorrect values, due to the interaction of the variable *i* in the macro and Data Step.

VIEWTABLE: WORK.TEST		
	i	out
1	2	1
2	4	6
3	6	120

Various defensive measures of selecting unique variables names and using DROP statements can overcome the majority of problems; however no solution using this method can be perfect.

Link and Return Statements

Another technique in the Data Step is to use Link and Return statements. These also have the same potential interaction issues, and are not as re-usable as the macro language as the fact label is only available within the current Data Step.

```
data test;
  do j=1 to 5;
    link fact;
  output;
end;
return;
fact:
  out=1;
  do i=2 to j;
    out=out*i;
  end;
return;
run;
```

This produces the same results as the initial SAS Macro version.

PROC FCMP

This procedure was first available in SAS 9.1, however only with SAS 9.2 are the functions created able to be used in the Data Step.

Functions are defined within the `function` statement (which defines the name and parameters), and the `endsub` statement. The code contained within this block is similar to the Data Step language; however it has a number of documented differences.

```
proc fcmp outlib=work.myfuncs.math;
  function fact2(x);
    result=1;
    do i=2 to x;
      result=result*i;
    end;
    return(result);
  endsub;
run;
quit;
```

After PROC FCMP has been used to define functions, SAS requires a new global option to be set to locate the new functions. The `CMPLIB` option is set below to point to the dataset created by PROC FCMP above.

```
options cmplib=work.myfuncs;
```

The Data Step is then able to call the new function, which shows that the variable `i` in the new function is completely independent of the Data Step variable.

```
data test;
  do i=1 to 5;
    out=fact2(i);
    output;
  end;
run;
```

VIEWTABLE: WORK.TEST		
	i	out
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120

Recursive code is also possible with functions created by PROC FCMP, as shown in the example below. It is possible to produce stack overflows if the recursion level becomes too deep, so recursion should be used with care. The code performance may also be impacted using this method.

```
proc fcmp outlib=work.myfuncs.math;
  function fact3(x);
    if x=1 then return(1);
    else return(x*fact3(x-1));
  endsub;
run;
quit;
data test;
  do i=1 to 5;
    out=fact3(i);
    output;
  end;
run;
```

PROC FCMP allows subroutines to be created, which can update multiple variables in a single call.

Subroutine definitions begin with `subroutine` and end with `endsub`. The variables to be updated are specified in the `outargs` statement, as shown in the example below..

```
proc fcmp outlib=work.myfuncs.math;
  subroutine fact4(x,outvar);
    outargs outvar;
    outvar=1;
    do i=2 to x;
      outvar=outvar*i;
    end;
  endsub;
run;
quit;
```

Subroutines can then be accessed with the SAS `CALL` statement.

```
data test;
  do i=1 to 5;
    call fact4(i,out);
    output;
  end;
run;
```

All three options above (using functions, recursive functions and subroutines) return identical results.

Other PROCs

The functions created in PROC FCMP are available to be used in other SAS Procedures, such as CALIS, GENMOD, MODEL, NLP, SQL etc, which can expand the functionality of these existing SAS procedures.

PROC PROTO

PROC PROTO provides the ability to describe functions in system DLLs. PROC FCMP can then be used to make new functions which can access these external functions.

The C function below performs the same Factorial calculation, and was compiled into a DLL on the Microsoft Windows platform. The use of `int` and `double` types provides an example that PROC PROTO can support both.

```
double WINAPI proto_fact(int n)
{
  double result=1.0;
  int nMax=n+1;
  for(int i=2;i<nMax;i++)
    result=result*i;
  return result;
}
```

The DLL can then be accessed with PROC PROTO to define the function to the SAS System, followed by PROC FCMP to create a function accessible to the Data Step.

```
proc proto package=work.proto.test
  stdcall;
  link "D:\VS\Proto\Release\Proto.dll";
  double proto_fact(int x "Fact Input")
  kind="MyMath";
run;

proc fcmp library=work.proto
  outlib=work.myfuncs.math;
  function fact4(x);
    return (proto_fact(x));
  endsub;
run;
quit;
```

Note that the `stdcall` option is important in the Microsoft Windows environment.

SASCBTBL / MODULEx

Previous releases of SAS included the ability to access external DLLs in a similar way to PROC PROTO; however the technology was not as easy to use. Accessing the same DLL through SASCBTBL / MODULEN function has shown the overhead was significantly higher for the older technology.

Conclusion

The BASE SAS language has been significantly enhanced in release 9.2 of the SAS System. PROC FCMP now allows the development of modular code with functions and subroutines.

Recursive functions can be defined with PROC FCMP; however system resources and performance can be impacted using this method.

Existing applications accessing external executables (DLLs) will benefit from the ease of use and increased performance that PROC PROTO provides.

References

SAS Institute Inc. 2008. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc

SAS Institute Inc. 2008. *What's New in SAS® 9.2*. Cary, NC: SAS Institute Inc.