GESStabs

Makros

Gesellschaft für Software
in der Sozialforschung mbH

Waterloohain 6 - 8
22769 Hamburg
Tel.: 040 - 853 753 - 0
Fax: 040 - 853 753 - 33
www.gessgroup.de

Howto: Macros

In GESStabs kann man viele Aufgaben mit weniger Code erledigen, wenn man Macros zu Hilfe nimmt. Mit einem Macro kann man beliebige Script-Schnipsel vereinbaren, die man öfter benötigt. An den Stellen seines Aufrufs wird ein Macro immer den <macroinhalt> hinterlassen, d.h. der Compiler wird anstelle des Macroaufrufs den Macroinhalt interpretieren. Der Macroinhalt wird vorher daraufhin untersucht, an welcher Stelle ein Parameter aus der Parameterliste gefunden wird, wenn dies der Fall ist, wird dieser Text durch den Parameterinhalt ersetzt. Es ist manchmal ganz hilfreich, sich die Verwendung eines Macros als eine Übersetzung vorzustellen, die vor der eigentlichen Interpretation des Scripts durch den Compiler stattfindet. Man spricht an dieser Stelle auch von einer „Expansion" des Macros.
(Die „Urform" eines Macros könnte man als das ebenfalls einsetzbare Expand bezeichnen. Hier ist über das Schlüsselzeichen #NAME ein beliebiger Text später im Script über den Aufruf #NAME wieder aufrufbar. Diese Möglichkeit ist allerdings im Gegensatz zum Macro nicht dynamisch)

Der wesentliche Vorteil von Macros ist es, dass man mit ihnen Redundanz vermeiden kann. Korrekturen erfolgen an zentraler Stelle, und die Gefahr ist geringer, dass man bei Änderungen die eine Stelle übersieht, die nachher beim Kunden für Ärger sorgt. Der wesentliche Nachteil sei nicht verschwiegen: Manchmal muss man zweimal nachdenken, um zu verstehen, was man da vor vier Wochen programmiert hat.

Ein Macro wird mit dem Schlüsselwort #MACRO definiert.

#macro <name> (<parameter>)
<macroinhalt>
#endmacro

Ein Macroname muss immer mit einem # (Lattenkreuz) beginnen. Die Parameter stehen in runden Klammern und haben in der Macrodefinition auch Namen, diese Namen müssen mit & (Ampersand) beginnen, in der allereinfachsten Form heißen sie &1 &2 &3 etc., aber oft ist es sinnvoll, sie mit „sprechenden" Namen zu versehen, z.B. &von oder &bis. Ein Beispiel:

```
#macro #example( &parameter )
compute  &parameter = 1;
#endmacro
```

wäre ein mögliches Macro. Wenn man es im Script verwendet, also .z.B. als

```
#example( frage1 )
```

wird es für den Compiler übersetzt zu:

```
compute frage1 = 1;
```

Eine häufige Verwendung von Macros besteht darin, dass man kleine „Unterprogramme" definiert. Wenn man beispielsweise viele Fragen hat, die man einerseits in der ursprünglichen Form verwenden will, zusätzlich aber mit Overcodes für Top- und Bottomboxes, dann kann man sich ein Macro schreiben:

```
#macro #mitOC( &varname )
compute    &varname_OC  = &varname;
copytext   &varname_OC  = &varname;
copytitle  &varname_OC  = &varname;
copylabels &varname_OC  = &varname;
labels     &varname_OC  = add
overcode 1 2 "Top 2"
overcode 9 10 "Bottom 2"
;
#endmacro
```

Wenn wir also für die Frage F1 Overcodes benötigen, dann reicht es also, dieses Macro einfach aufzurufen:

```
#mitOC( F1 )
```

und wir erhalten eine neue Variable &varname_OC mit zwei Overcodes. Hier das Macro einmal exemplarisch in seiner vom Compiler ausgeführten Form:

```
compute    F1_OC  = F1;
copytext   F1_OC  = F1;
copytitle  F1_OC  = F1;
copylabels F1_OC  = F1;
labels     F1_OC  = add
overcode 1 2 "Top 2"
overcode 9 10 "Bottom 2"
;
```

Diese Schreibersparnis können wir nun auch gezielt durch Macro-Aufrufe für weitere Fragen einsetzen:

```
#mitOC( F2 )
#mitOC( F3 )
#mitOC( F4 )
#mitOC( F5 )
```

An die Stelle von solchen Aufruflisten kann auch eine #DOMACRO Anweisung treten. Diese benötigt als ersten Parameter immer den Namen des wiederholt aufzurufenden Macros, und danach eine Liste aller zu iterierenden Elemente:

#domacro(mitOC F1 F2 F3 F4 F5)

Der Übersetzungsprozess wird jetzt zweistufig: Als erstes wird die #DOMACRO-Anweisung in die einzelnen Macros übersetzt, und dann wird jedes einzelne Macro in den Macroinhalt übersetzt. Im Anschluss sind dann die fünf Variablen F1_OC bis F5_OC erzeugt, und können zum Tabellieren verwendet werden.

Man kann dieses Spielchen noch ein wenig weiter treiben. Mal angenommen, wir haben es nicht mit F1 bis F5 zu tun, sondern mit 100 solcher Fragen, also mit F1 bis F100. Dann würde man sich wünschen, dass man nicht alle 100 Variablen explizit in die #DOMACRO Anweisung hinschreiben muss. Hierfür bedienen wir uns der Möglichkeit, Macroaufrufe zu schachteln. Wir schreiben ein Macro, dass das Macro #mitOC aufruft, und dieses Macro rufen wir mit einer Domacro-Schleife auf, die nur die Zahlen 1 bis 100 enthält. Solche Zahlenfolgen kann man in einer #DOMACRO Anweisung durch die Form 1 : 100 abkürzen.

```
#macro #mitOC_F( &1 )
#mitOC( F&1 )
#endmacro
#domacro(  mitOC_F 1:100 )
```

Das funktioniert so, wie es soll, ist aber etwas unelegant. Für jedes ähnliche Problem muß man jeweils ein angepasstes Macro schreiben, nur weil z.B. andere Namen wie var1 bis var100 erforderlich sind. Gibt es hierfür eine allgemeinere Lösung, die man wiederverwenden kann? Wir schreiben ein allgemeineres Macro #call, das drei Parameter hat:

```
#macro #call( &index &namepart &macroname )
#&macroname( &namepart&index )
#endmacro
```

Der Aufruf ist jetzt flexibel und könnte also lauten:

```
#call( 1 F mitOC )
```

Das würde übersetzt zu

```
#mitOC( F1 )
```

oder

```
#call( 1 var mitOC )
```

Das würde übersetzt zu

```
#mitOC( var1 )
```

Und den Aufruf für 200 Variablen Var1 bis Var200 können wir jetzt mit einem #DOMACRO2 erledigen. Dies entspricht bis zum Semikolon dem #DOMACRO, danach können weitere „konstante“ Parameter übergeben werden.

```
#domacro2( call 1 : 200 ; Var mitOC )
```

Durch Aufruf des Macros #call mit dem #domacro2 entstehen somit 200 verschiedene Ausdrücke, beginnend mit `#mitOC( var1 )` bis hin zu `#mitOC( var200 )`. Zur Erinnerung, diese 200 Ausdrücke wiederum rufen das Macro #mitOC auf und generieren so weitere 200 Variablen mit den Namen `Var1_OC` bis `Var200_OC`, inklusive Labels, VarText, VarTitle und den gewünschten Overcodes (Top 2, Bottom 2).

Vorausgesetzt, dass wir noch ein weiteres Macro in der Tasche haben, das für jede Variable zwei Tabellen mit #K1 und #K2 macht, können wir schnell noch 400 Tabellen mit Overcodes nachschieben:

```
#macro #tabellen( &variablename )
table = #K1 by &variablename_OC;
table = #K2 by &variablename_OC;
#endmacro

#domacro2( call 1 : 200 ; Var tabellen )
```

Die Macros #mitOC und #tabellen funktionieren wie kleine Unterprogramme; als Resultate werden also vollständige Statements der GESS Scriptsprache expandiert. Wenn wir den Ablauf einmal in Zeitlupe betrachten, so passiert mit den beiden nacheinander ausgeführten Macros folgendes:

```
#domacro2( call 1 : 200 ; Var mitOC )
#domacro2( call 1 : 200 ; Var tabellen )
```

Das `#domacro2( call 1 : 200 ; Var mitOC)` generiert im ersten Durchlauf durch den Aufruf des Macro `#call 200` Ausdrücke wie `#mitOC ( var1)` bis `#mitOC ( var200 )`. Diese 200 Ausdrücke wiederum werden dann über das `Macro #mitOC` zu den gewünschten 200 Variablen `(var1_OC)` bis `(var200_OC)` expandiert / generiert.

Anschließend kann über das folgende Macro #tabellen gezielt auf die generierten Variablen mit Overcodes zugegriffen werden. Damit in unserem Beispiel gezielt auf auf die neu generierten Variablen inklusive der Overcodes zugegriffen werden kann, haben wir den Platzhalter `&variablenname` schlicht um den geänderten Variablennamen inklusive Overcodes ergänzt `&variablenname`_OC (ohne Leerzeichen!). Statt zwei einzelne Macro Aufrufe könnte auch der komplette Ablauf über ein Macro gesteuert werden, an dieser Stelle würde es aber doch ein wenig „zu verschachtelt“ werden.

Das `#domacro2( call 1 : 200 ; Var tabellen )` generiert somit anschließend wieder durch den Aufruf des Macro `#call 200` Ausdrücke wie `#tabellen( var1 )` bis `#tabellen( var200 )`. Jetzt greift das Macro #tabellen (var1) allerdings durch die Ergänzung „_OC“ im Macro #tabellen auf die generierten Variablen mit den gewünschten Overcodes zu und tabelliert diese gegen beide Köpfe `#K1` und `#K2`.

Oft ist es aber praktisch, kleinere Codeschnipsel zu erzeugen. Wenn wir an die 200 Variablen Var1 bis Var200 denken, dann ist eine Übersichtstabelle mit Mittelwerten vielleicht einfach „übersichtlicher“. Wir hätten als Expansionsergebnis dann gern so etwas wie

```
table = #K1 by
mean( Var1 )
mean( Var2 )
…..
Mean( Var200 )
;
```

Wir benötigen hier also „so etwas ähnliches“ wie unser Macro #call, aber leicht modifiziert: es soll kein macro aufgerufen werden, sondern nur der Parameter mit dem zusammengesetzten Variablennamen in Klammern übergeben werden, d.h. ohne das Lattenkreuz:

```
#macro #stat( &index &namepart &statistik )
&statistik ( &namepart&index )
#endmacro
```

Wir können dann formulieren:

```
table = #K1 by
#domacro2( stat 1:200 ; Var mean )
;
```

Wenn wir alle MEAN durch MEANTEST ersetzen und außerdem ein spezielles Format verwenden wollen, dann können wir das jetzt an einer zentralen Stelle ändern:

```
table = #K1 by
#domacro2( stat 1:200 ; Var "Meantest :format '#,#'" )
;
```

Damit `Meantest :format '#,#'` als ein zusammenhängender Parameter interpretiert wird, müssen wir diesen Ausdruck beim Aufruf in Anführungszeichen setzen. Da nun bereits ein Ausdruck in Anführungszeichen im Parameter vorkommt, müssen wir das alternative Anführungszeichen verwenden.

Als kleine Arbeitshilfe kann man sich die expandierten Macros in einer Protokolldatei anschauen, wenn man den Befehl macroprotocol = <dateiname>; verwendet. Dann würden alle expandierten Macros in der folgenden Form dort protokolliert:

```
-- expand #stat (  1 Var Meantest :format '#,#' )
Meantest :format '#,#' ( Var1 )
-- end #stat (  1 Var Meantest :format '#,#' )
```

Wenn man eine quasi 2-dimensionale Struktur hat, kann man das über zweistufige DOMACRO Konstrukte abbilden. Es seien die Variablen V1_1 bis V1_10, V2_1 bis V2_10 … V10_1 bis V10_10 zu bearbeiten. Das kann man z.B. so erschlagen:

```
#macro #stat( &index &namepart &statistik )
&statistik ( &namepart&index )
#endmacro

#macro #schleife( &index &namepart &statistik )
#domacro2( stat 1 : 10 ; &namepart&index_  &statistik )
#endmacro

table =
#domacro2( schleife 1 : 10 ; V Mean )
;
```

Wir verwenden dasselbe Macro #stat wie eben, und verlegen den DOMACRO2-Aufruf von #stat in ein weiteres Macro namens #schleife. # schleife wird ebenfalls mit einem #DOMACRO2-Aufruf aufgerufen.

Die ersten 2 Durchgänge durch #schleife ergeben die Expansion:

```
#domacro2 ( stat 1 : 10 ; V1_ Mean )
#domacro2 ( stat 1 : 10 ; V2_ Mean )
```

Die Durchgänge durch # stat ergeben die gewünschte Expansion:

```
Mean ( V1_1 )
Mean ( V1_2 )
…
…
Mean ( V2_1 )
Mean ( V2_2 )
…
…
Mean ( V10_9 )
Mean ( V10_10 )
```

Soweit, so gut. Das ist alles sehr schön, wenn man es mit völlig gleich zu behandelnden Variablen zu tun hat. Im echten Leben ist das aber oft weniger übersichtlich. Oft ist es ja so, dass man eine Reihe von Variablen hat, die erstens nicht so schön systematische Namen tragen, und die zweitens unterschiedlich dargestellt werden sollen.

Variablen vom Typ A sollen gegen einen Kopf tabelliert werden, die vom Typ B gegen zwei Köpfe, und Variablen vom Typ C gegen zwei Köpfe mit einem zusätzlichen Mittelwert. Der Anfang des Fragebogens enthält folgende Variablen mit den dazugehörigen Auswertungswünschen.

```
F1    typ_a
F2a   typ_a
F2b   typ_a
F2c   typ_a
F2    typ_b
F4    typ_b
F4a   typ_b
F5a   typ_c
F5b   typ_c
```

Die Auswertungswünsche können wir in 3 Macros darstellen:

```
#macro #typ_a( &var )
table = #k1 by &var;
#endmacro

#macro #typ_b( &var )
table = #k1 by &var;
table = #k2 by &var;
#endmacro

#macro #typ_c( &var )
table = #k1 by &var mean :description "Mittelwert" ( &var );
table = #k2 by &var mean :description "Mittelwert" ( &var );
#endmacro
```


Dann greifen wir uns die Tabelle der Variablen mit den Auswertungswünschen, und stecken sie folgendermaßen als einzelne Zeilen in ein Macro namens #vars.

```
#macro #vars( &job )
#&job ( F1    typ_a )
#&job ( F2a   typ_a )
#&job ( F2b   typ_a )
#&job ( F2c   typ_a )
#&job ( F2    typ_b )
#&job ( F4    typ_b )
#&job ( F4a   typ_b )
#&job ( F5a   typ_c )
#&job ( F5b   typ_c )
#endmacro
```

Wenn man in dieser Konstruktion das Macro #vars aufruft, dann expandieren die einzelnen Zeilen zu einem Macroaufruf eines Macros mit dem Namen des Parameters (&job).

Also bekommen wir bei einem Aufruf von #vars(tabs) folgende Expansionen:

```
#tabs( F1 typ_a )
…
#tabs( F4 typ_b )
…
#tabs( F5a typ_c )
```

Jetzt brauchen wir nur noch ein passendes Macro namens tabs. Dies soll seinerseits ein Macro aufrufen, der den zweiten Parameter als Namen trägt, und den ersten Parameter als Parameter. Nichts einfacher als das:

```
#macro #tabs( &var &mac )
#&mac ( &var )
#endmacro

#vars( tabs )
```

Hat doch was? Im ersten Moment verrenkt es einem das Hirn, aber es ist eine sehr übersichtliche Darstellung des Tabellierauftrags.

Für die doch „noch leicht verrenkten einmal zum Einrenken“ der Ablauf der Macro-Aufrufe…:

`#vars( tabs )`

⇩

Der Macroaufruf #vars startet den gesamten Tabellierjob.

`#tabs( F1 typ_A )`
`#tabs( F2a typ_A )usw…`

⇩

Der Macro #vars generiert die verschiedenen Jobs, bzw. #tabs – Aufrufe für die gewünschten Tabellentypen.

`#tabs( F1 typ_A )`

⇩

Zur Veranschaulichung ist jetzt der Macro-Aufruf `#tabs( F1 typ_A )` herausgegriffen. Dieser expandiert zu `#typ_a(F1) und` ruft in diesem Beispiel das Macro:

```
#macro #typ_a( &var )
table = #k1 by &var;
#endmacro
```

für die Variable F1 auf.

⇩

`table = #k1 by &var;`

Diese Anweisung generiert / expandiert jetzt die gewünschte Tabelle(n) für die Variable F1 nach dem Tabellentyp A.