Gentle Beispiele

```
Gentle Beispielprogramme mit besonderen Gentle-Befehlen
Zwischen zwei Sternchen-Zeilen steht jeweils ein lauffaehiges
Gentle-Programm. Am Ende des Programms findet man Kommentarzeilen,
die die Ausgabe des Programms enthalten.
*************************************************************************
// ---------------------------------------------------------------------
// File pred03\alternatives01.g
// An example in which an alternatives statement is possibly
// of great advantage.
//
// Does not contain token or phrase predicates.
//----------------------------------------------------------------------
// The predicates processA and processB solve the same problem,
// processA with several rules and
// processB with an alternatives statement in only one rule.
// processA calls the procedure expensive more often then processB.
//----------------------------------------------------------------------
proc processA(N:int)
   rule processA(N):
      expensive(N -> RES)
      RES -> 1  "*** The result was 1!\n\n"
   rule processA(N):
      expensive(N -> RES)
      RES -> 2  "*** The result was 2!\n\n"
   rule processA(N):
      expensive(N -> RES)
      RES -> 3  "*** The result was 3!\n\n"
   rule processA(N):
      "*** The result was something else!\n"
//----------------------------------------------------------------------
proc processB(N:int)
   rule processB(N):
      expensive(N -> RES)
      {
         RES -> 1  "*** The result was 1!\n\n" |
         RES -> 2  "*** The result was 2!\n\n" |
         RES -> 3  "*** The result was 3!\n\n" |
         "*** The result was something else!\n"
      }
//----------------------------------------------------------------------
proc expensive(N:int -> RES:int)
   // Computes RES as N mod 17
   // Assume that this predicate is expensive to execute
   rule expensive(N -> N - (N/17*17)):
     "expensive was called!\n"
//----------------------------------------------------------------------
root
   "-------------------------------\n"
   "processA:\n\n"
   processA(1)
   processA(2)
   processA(3)
   processA(-123)
   "-------------------------------\n"
   "processB:\n\n"
   processB(1)
   processB(2)
   processB(3)
   processB(-123)
   "-------------------------------\n"
//----------------------------------------------------------------------
// Output of this program:
// ------------------------------
// processA:
//
```

```
// expensive was called!
// *** The result was 1!
//
// expensive was called!
// expensive was called!
// *** The result was 2!
//
// expensive was called!
// expensive was called!
// expensive was called!
// *** The result was 3!
//
// expensive was called!
// expensive was called!
// expensive was called!
// *** The result was something else!
// -------------------------------
// processB:
//
// expensive was called!
// *** The result was 1!
//
// expensive was called!
// *** The result was 2!
//
// expensive was called!
// *** The result was 3!
//
// expensive was called!
// *** The result was something else!
// -------------------------------
************************************************************************
// ----------------------------------------------------------------------
// File pred03\alternatives02.g
// Demonstrates the alternatives statement.
//----------------------------------------------------------------------
// The predicates evalA, evalB and evalC compute the same function
// (they all evalute expressions of type Exp)
//
// evalA consists of 5 rules, one for each case                (14 lines)
// evalB consists of 2 rules, the first of which contains
//       an alternatives statement                             (14 lines)
// evalC the same as evalB, only typed with a different layout (11 lines)
// ----------------------------------------------------------------------
type Op2 add() sub() mul() div()

type Exp
   exp(Op2, Exp, Exp)
   lit(int)
// ----------------------------------------------------------------------
proc evalA(Exp -> int)
   rule evalA(exp(add(), E1, E2) -> N1+N2):
      evalA(E1 -> N1)
      evalA(E2 -> N2)
   rule evalA(exp(sub(), E1, E2) -> N1-N2):
      evalA(E1 -> N1)
      evalA(E2 -> N2)
   rule evalA(exp(mul(), E1, E2) -> N1*N2):
      evalA(E1 -> N1)
      evalA(E2 -> N2)
   rule evalA(exp(div(), E1, E2) -> N1/N2):
      evalA(E1 -> N1)
      evalA(E2 -> N2)
   rule evalA(lit(N) -> N):
// ----------------------------------------------------------------------
proc evalB(Exp -> int)
   rule evalB(exp(OP, E1, E2) -> V):
```

```
133       evalB(E1 -> V1)
134       evalB(E2 -> V2)
135       {
136          OP -> add() V <- V1+V2
137       |
138          OP -> sub() V <- V1-V2
139       |
140          OP -> mul() V <- V1*V2
141       |
142          OP -> div() V <- V1/V2
143       }
144    rule evalB(lit(N) -> N):
145 // ------------------------------------------------------------------------
146 proc evalC(Exp -> int)
147    rule evalC(exp(OP, E1, E2) -> V):
148       evalC(E1 -> V1)
149       evalC(E2 -> V2)
150       {
151          OP -> add() V <- V1+V2 |
152          OP -> sub() V <- V1-V2 |
153          OP -> mul() V <- V1*V2 |
154          OP -> div() V <- V1/V2
155       }
156    rule evalC(lit(N) -> N):
157 // ------------------------------------------------------------------------
158 proc test1(Exp)
159    rule test1(E):
160    evalA(E -> VA)
161    evalB(E -> VB)
162    evalC(E -> VC)
163    "------\n"
164    "VA: " $VA "\n"
165    "VB: " $VB "\n"
166    "VC: " $VC "\n"
167 // ------------------------------------------------------------------------
168 root
169    test1(exp(add(), lit(1), lit(1)))
170    test1(exp(sub(),
171              exp(mul(), lit(2), lit(3) ),
172              exp(div(), lit(8),
173                         exp(add(), lit(1), lit(3))
174                 )
175             )
176         )
177 // ------------------------------------------------------------------------
178 // Output of this program:
179 //
180 // ------
181 // VA: 2
182 // VB: 2
183 // VC: 2
184 // ------
185 // VA: 4
186 // VB: 4
187 // VC: 4
188 ************************************************************************
189 // ------------------------------------------------------------------------
190 // File pred03\alternatives03.g
191 // Nested alternatives statements
192 //-------------------------------------------------------------------------
193 proc pBase3(N1:int, N2:int)
194    // Prints a single entry of the multiplication table
195    // for base 3 numbers, from 1x1 up to 10x10 (i.e. 3x3)
196    rule pBase3(N1, N2):
197    {
198       N1 -> 1
199       {
200          N2 -> 1 " 1x 1 =   1\n" |
```

```
201          N2 -> 2 " 1x 2 =   2\n" |
202          N2 -> 3 " 1x10 =  10\n" |
203          "Parameter N2: " $N2 " is out of range\n"
204       }
205    |
206       N1 -> 2
207       {
208          N2 -> 1 " 2x 1 =   2\n" |
209          N2 -> 2 " 2x 2 =  11\n" |
210          N2 -> 3 " 2x10 =  20\n" |
211          "Parameter N2: " $N2 " is out of range\n"
212       }
213    |
214       N1 -> 3
215       {
216          N2 -> 1 "10x 1 =  10\n" |
217          N2 -> 2 "10x 2 =  20\n" |
218          N2 -> 3 "10x10 = 100\n" |
219          "Parameter N2: " $N2 " is out of range\n"
220       }
221    |
222       "Parameter N1: " $N1 " is out of range\n"
223    }
224 //------------------------------------------------------------------------
225 root
226    "-----------------------\n"
227    "1x1 for base 3 numbers:\n"
228    "-----------------------\n"
229    pBase3(1, 1)
230    pBase3(1, 2)
231    pBase3(1, 3)
232    "------------\n"
233    pBase3(2, 1)
234    pBase3(2, 2)
235    pBase3(2, 3)
236    "------------\n"
237    pBase3(3, 1)
238    pBase3(3, 2)
239    pBase3(3, 3)
240    "------------\n"
241    pBase3(0, 1)
242    pBase3(1, 4)
243    "------------\n"
244 (@ ------------------------------------------------------------------------
245 Output of this program (as a multi line comment):
246
247 -----------------------
248 1x1 for base 3 numbers:
249 -----------------------
250  1x 1 =   1
251  1x 2 =   2
252  1x10 =  10
253 ------------
254  2x 1 =   2
255  2x 2 =  11
256  2x10 =  20
257 ------------
258 10x 1 =  10
259 10x 2 =  20
260 10x10 = 100
261 ------------
262 Parameter N1: 0 is out of range
263 Parameter N2: 4 is out of range
264 ------------------------------------------------------------------------ @)
265 ************************************************************************
266 (@ ------------------------------------------------------------------------
267 File pred03\qups02.g
268 Demonstriert QUPs (query update pairs)
```

```
269 NaechsteMatNr ist ein einfaches QUP (simple   query update pair)
270 MatNr         ist ein Funktions-QUP (function query update pair)
271 ------------------------------------------------------------------------ @)
272 data NaechsteMatNr(-> int)
273 // Definiert 2 Praedikate:
274 // Set-NaechsteMatNr(int)
275 // Get-NaechsteMatNr(-> int)
276 // ---------------------------------------------------------------------
277 data Name2MatNr(string -> int)
278 // Definiert 2 Praedikate:
279 // Set-Name2MatNr(string, int)
280 // Get-Name2MatNr(string -> int)
281 // ---------------------------------------------------------------------
282 proc imma(Name:string -> MatNr:int)
283    // "imma" wie "Immatrikulation"
284    // Holt sich aus dem Qups NaechsteMatNr die naechste (noch freie)
285    // Matrikrel-Nr NM und erhoeht den Wert von NaechsteMatNr um 1.
286    // Traegt den Name als Schluessel mit dem Wert NM in das Qups
287    // Name2MatNr ein. Liefert NM als Wert des out-Params MatNr.
288    rule imma(NAM -> NM):
289       Get-NaechsteMatNr(-> NM)
290       Set-NaechsteMatNr(NM+1)
291
292       Set-Name2MatNr(NAM, NM)
293 // ---------------------------------------------------------------------
294 proc p(Name:string)
295    // Gibt den Name und die dazugehoerige Matrikel-Nr (falls vorhanden)
296    // aus. Fehlermeldung, wenn Name noch nicht immatrikuliert ist.
297    rule p(NAM):
298       {
299          Get-Name2MatNr(NAM -> MN)
300          "Name: " $NAM ", MatNr: " $MN "\n"
301       |
302          $NAM " ist noch nicht immatrikuliert!\n"
303       }
304 // ---------------------------------------------------------------------
305 root
306    // Die NaechsteMatNr initialisieren:
307    Set-NaechsteMatNr(100101)
308
309    // Zwei StudentInnen werden imatrikuliert:
310    imma("Anna" -> ANR)
311    imma("Bert" -> BNR)
312
313    // Immatrikulations-Daten ausgeben:
314    p("Anna")
315    p("Bert")
316    p("Carl")
317 (@ ---------------------------------------------------------------------
318 Ausgabe:
319
320 Name: Anna, MatNr: 100101
321 Name: Bert, MatNr: 100102
322 Carl ist noch nicht immatrikuliert!
323 ------------------------------------------------------------------------ @)
324 ************************************************************************
325 // ---------------------------------------------------------------------
326 // File pred03\optionTypes01.g
327 // Demonstrates option types.
328 //----------------------------------------------------------------------
329 type NAME
330    nam(LastName:string, FirstName:string?)
331    // LastName is mandatory, FirstName is optional
332
333 proc pNAME(NAME)
334    // Pretty prints a NAME
335    rule pNAME(nam(LN, string?())):
```

```
336       "Last name: " $LN "\n"
337    rule pNAME(nam(LN, string?(FN))):
338       "First and last Name: " $FN " " $LN "\n"
339
340 root
341    Present <- int?(123)
342    "------------------------------- A\n"
343    "log Present: \n"
344    log Present
345    Present -> int?(N)
346    "------------------------------- B\n"
347    "log N:\n"
348    log N
349    "------------------------------- C\n"
350    Missing <- int?()
351    "log Missing: \n"
352    log Missing
353    "------------------------------- D\n"
354    {
355       Present -> int?()
356       "Present -> int?() suceeded!\n"
357    |
358       "Present -> int?() failed!\n"
359    }
360
361    "------------------------------- E\n"
362    {
363       Missing -> int?()
364       "Missing -> int?() suceeded!\n"
365    |
366       "Missing -> int?() failed!\n"
367    }
368    "------------------------------- F\n"
369    // A Name WITH a first name (which is optional):
370    Name1 <- nam("Meyer", string?("Otto"))
371    "log Name1:\n"
372    log Name1
373    "-------------------------------  \n"
374    "pNAME(Name1):\n"
375    pNAME(Name1)
376    "------------------------------- G\n"
377    // A Name WITHOUT a first name (which is optional):
378    Name2 <- nam("Schulz", string?())
379    "log Name2:\n"
380    log Name2
381    "-------------------------------  \n"
382    "pNAME(Name2):\n"
383    pNAME(Name2)
384    "------------------------------- H\n"
385 (@ ------------------------------------------------------------------------------
386 Output of this program:
387
388 ------------------------------- A
389 log Present:
390 int?(
391    123
392 )
393 ------------------------------- B
394 log N:
395 123
396 ------------------------------- C
397 log Missing:
398 int?()
399 ------------------------------- D
400 Present -> int?() failed!
401 ------------------------------- E
402 Missing -> int?() suceeded!
403 ------------------------------- F
```

```
404 log Name1:
405 nam(
406    "Meyer",
407    string?(
408       "Otto"
409    )
410 )
411 --------------------------------
412 pNAME(Name1):
413 First and last Name: Otto Meyer
414 -------------------------------- G
415 log Name2:
416 nam(
417    "Schulz",
418    string?()
419 )
420 --------------------------------
421 pNAME(Name2):
422 Last name: Schulz
423 -------------------------------- H
424 ------------------------------------------------------------------------------ @)
425 ******************************************************************************
```