Chapter 19 „Programming Language II“

Content

With the instructions and functions introduced in Chapter 18, the majority of basic blocks can be translated into programming language programs. In the first section of this chapter, a number of special programming structures and functions are explored. Content of the second section is the automatic length adjustment using a collar as an example; followed by the application of external functions and the generation of construction components.

```
'*************************************************************
  Program Main()
'-------------------------------------------------------------
' Collar construction with automatic length adjustment        .
'-------------------------------------------------------------
  nVar n,nNextPos,nT
  rVar rZl,rA,rL,rA1,rL1,rA2,rL2
  pVar p0,p1,p2,p3,p4,p5
  qVar q1,q2,q3,qF,qB,qQqq
  cVar cF,cB
'---------------------------------------- x value definitions
  XTitel("x values for shirt collar construction")
  Defx(1,"direction for collar neck",-45)
  Defx(2,"direction for collar stand",-55)
  Defx(3,"raise collar",35)
  Defx(4,"stand width",15)
  Defx(5,"collar width back",45)
  Defx(6,"point length",60)
  Defx(7,"point height",60)
  Defx(8,"direction collar edge curve",-30)
'------------------------------------ query length of neckline
  qF = qKop(pXY(0,0)+pXY(100,0))                  ' allocate qF
  qB = qKop(pXY(0,0)+pXY(50,0))                   ' allocate qB
  cF = cPick(1,4,"Click FRONT neckline !","collar","!",nT)
  qF = qCo(cF,"qq")
  cB = cPick(2,4,"Click BACK neckline !","collar","!",nT)
  qB = qCo(cB,"qq")
'------------------------------------- calculate target length
  rZl= rLngQ(qF)+rLngQ(qB)
'-------------------------------------- task cannot be solved
  If(rZl<=rX(3)) Then
    n  = nIBox("The collar cannot be constructed !")
    Exit Program
  End If
'--------------------------- starting adjustment P0 => P1 (=rA)
  rA = rZl
'-------------- 0.approximation with construction of collar edge
  p0 = pXY(0,0)
  p1 = pXY(rA,0)
  p2 = pXY(0,rX(3))
  q1 = qSpline(p2,0,p1,rX(1))
  rL = rLngQ(q1)                      ! curve length calculation
'------------------------- automatic length adjustment in cycle
  rA1= 0              ' allocate function values for 1st approx.
  rL1= rX(3)
  rA2= rA
  rL2= rL
  For n = 1,10,1                ' maximum 10 approximation steps
    rA = rNahInt(rA1,rL1,rA2,rL2,rZl)       ' next approximation
    p1 = pXY(rA,0)
    q1 = qSpline(p2,0,p1,rX(1))
    rL = rLngQ(q1)                    ! curve length calculation
    If(rAbs(rL-rZl)<<0.01) Then           ! achieved accuracy ?
      Exit For                          ! if YES  => quit loop
    End If
    rA1= rA2 !re-allocate function values for next approximation
    rL1= rL2
    rA2= rA
    rL2= rL
  End For
```

19.1 Subjects for advanced users

IF-THEN structure

The IF-THEN structure is a control structure which carries out calculations or construction steps only, if a certain condition is fulfilled.

The simple structure is

```
If (logical term) Then
   [instruction]
End If
```

Only if the logical term is true (has the value True), are the instructions processed.

The logical term can either be a direct logical variable

```
lSwitch=true
If(lSwitch) Then
   [instruction]
End If
```

or the result of a comparison operation between whole / real numbers or variables.

```
If(rEa<<0) Then
  rSs = rEa
  rBk = 0
  rFt = 0
End If
```

The following comparison operators are permitted for whole number / real variables:

Character	Significance
<<	smaller than
>>	greater than
==	equal
<=	smaller or equal
>=	greater or equal
<>	not equal

For connection of logical variables the following is permitted:

Character	Significance
NOT	„not“
AND	„and“
OR	„or“

The operations „==“ and „<>“ are only suitable for whole numbers as the numbers are compared up to the 6th decimal, inclusive.

Example:

For extreme individual sizes, the waist girth may be greater than the hip girth. In this case, the negative ease must be added, completely to the side seam. This case is considered in the program as follows:

```
'-------------------- distribute ease
  rSs = 3/6*rEa  'portion in side seam
  rBk = 2/6*rEa          'portion in Bk
  rFt = 1/6*rEa          'portion in Ft
  If(rEa<<0) Then
    rSs = rEa
    rBk = 0
    rFt = 0
  End If
```

Within IF-THEN structures, no objects may be output, as the number of objects, the object type or the order of the objects may change. Changes in the object output may lead to errors during style development. The notes on linking programming language ⇔ construction record in the previous chapter apply.

The detailed structure is

```
If (logical term 1) Then
      [instruction 1]
Else If (logical term 2) Then
      [instruction 2]
Else If (logical term 3) Then
      [instruction 3]
Else
      [instruction 4]
End if
```

Explanations can be found in Picture 19-1. The „`Else If() Then`“ queries can follow „`If() Then`“ a number of times. „`Else`“ is only permitted once before „`End If`“.

Example:

In a skirt block for individual sizes, the ease is to be distributed differently, if the ease for half of the skirt is greater than 40 mm. Enter the following in the program:

If (logical term 1**) Then**	**If (**logical expression 1**) Then**
[instruction 1]	The instruction 1 is only processed if „logical term 1“ is true. The IF-ENDIF structure is quit.
Else If (logical term 2**) Then**	**Or If (**logical term 2**) Then**
[instruction 2]	The instruction 2 is only processed if „logical term 2“ is true and „logical term 1“ was false. The IF-ENDIF structure is quit.
Else	**Otherwise**
[instruction 3]	The instruction 3 is only processed if the previous queries were false.
End If	**End If**

Picture 19-1

```
'--------------------- distribute ease
'              rSs portion in side seam
'              rBk portion in Bk
'              rFt portion in Ft
'------------------------- case rEa<0
  If(rEa<<0) Then
    rSs = rEa
    rBk = 0
    rFt = 0
'------------------------ case rEa<40
  Else If(rEa<<40) Then
    rSs = 1/6*rEa
    rBk = 3/6*rEa
    rFt = 2/6*rEa
'----------------------- case rEa>=40
  Else
    rSs = 1/4*rEa
    rBk = 2/4*rEa
    rFt = 1/4*rEa
  End If
```

FOR-NEXT structure

With the FOR-NEXT structure, loops can be created. The loops begins with

```
For nLauf = nA,nE,nStep
```

and ends with

```
End For
```

`nLauf` is the loop variable. During the first run, it has the value nA. After each run, `nLauf` is automatically raised by `nStep` or reduced if `nStep` is negative. The instructions between `For` and `End For` are processed, repeatedly during each run of the loop.

The loop is only quit when the loop variable has exceeded the end value or `Exit For` has been instructed. The variables `nLauf`, `nA`, `nE` and `nStep` must be whole number variables.

The complete structure in an overview is shown in Picture 19-2.

```
For nLauf = nA,nE,nStep
      [instruction]
  [Next For]    (next loop run)
      [instruction]
  [Exit For]    (quit loop, immediately)
      [instruction]
End For
```

Picture 19-2

Example:

```
'-----------check if x1 to x5 negative
  nA=1
  nE=5
  For nLauf=nA,nE,1
    If(rX(nLauf)<<0) Then
     t1="The x value x"+tFormat(nLauf)
&       +" is negative !"+tc(13,10)+
&        "The collar cannot be
&        constructed."
      nBox= nIBox(t1,31)
      Exit Program
    End If
  End For
```

These program lines check whether one of the x values x1 to x5 is negative. If one of the values is negative, the program is immediately aborted with a message.

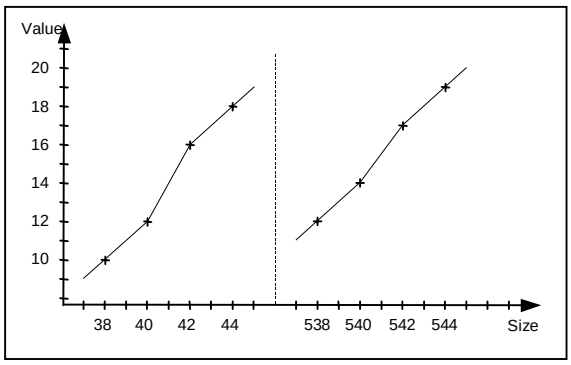
Size interpolation

The function `rGroInt()` carries out a size-related interpolation. In the previous programming language, y values were defined for this process. Size-related interpolation is useful, if a value is to be altered depending on the current measurement chart. The same result is achieved through use of size-related x values. As opposed to the x values, values calculated with `rGroInt()` can only be adjusted in the programming language program. The user of the released program has no access to these values. (S)he cannot alter them.

A size-related value of the real number type can be defined at any point in the program with the following instruction:

```
  rCorr1=rGroInt("Size",value
&                [,"Size“,value,])
```

As parameters, any number of pairs made up from size and the relevant value can be entered. The function calculates the value for the current measurement chart from the value pairs size/value. Size names must be entered in inverted commas, where the underscore „_“ indicates a standard size. It is recommended to enter the sizes in ascending order. If no value pair is entered for a figure type, the value of the first entered value pair is applied to all sizes of the figure type. Please note the following example:



Picture 19-3

A correction value for the figure types normal and broad hip is to be defined according to Picture 19-3. The following instruction defines the value `r1` as required:

```
  r1  = rGroInt("_38",10,"_40",12,
&       "_42",16,"_44",18,"_538",12,
&       "_540",14,"_542",17,"_544",19)
```

The following overview shows which value the variable `r1` takes on for the respective measurement chart:

Size	r1	Size	r1	Size	r1
36	8	036	10	536	12
38	10	038	10	538	14
40	12	040	10	540	17
42	16	042	10	542	19
44	18	044	10	544	21
46	20	046	10	546	23

For the figure type narrow hips (pre-fix „0“), no value was defined. Thus, the variable obtain the value 10 for all sizes of this figure type.

For individual measurement charts, the value of the respective reference size (column x value reference in the size table) is entered. If no reference is entered in this column, again, the value of the first value pair applies.

Dialogue functions

In basic blocks or construction components, information from the Grafis record are often required. This information can be process parameters (lengths, distances,...), but also objects (points, lines). For example, for the collar, information about the neckline is required; for a sleeve, information about the armhole. In the new programming language, a dialogue can be created which instructs the user of the program to click the required objects.

For the dialogue with the user, the functions

```
nIBox()
cPick()
```

are available.

nIBox()

Infobox builds a window with a note for the user or which the user can close with Yes/No. In this window, for example, the user can be told the reason why a construction cannot run under the particular conditions. If an x value is assigned with an extreme value, the user can be warned. ***The infobox appears also during grading! As a rule, use nIBox for error messages for extreme construction entries from your program.***

The infobox can appear in different display options. The display option is determined with the optional whole number parameter nD. The first digit of this 2-digit parameter controls which button is displayed. The second digit determines the icon in the window. Picture 19-4 contains an overview of the display options.

The function value of `nIBox` indicates the clicked button. The values of `nIBox` stand for the following buttons:

1 - Button „OK“
2 - Button „Yes“
3 - Button „Non“
4 - Button „Ignore“
5 - Button „Repeat“
6 - Button „Quit“

Clicking the button „Abort“ always leads to immediate abort of the program. It is identical with `Exit Program`.

Example:

Before the construction steps for a mitred corner are carried out in the programming language, it is tested, whether the requested lines are suitable for creation of a corner and are not parallel.

```
'-------------------- Lines parallel ?
rWi = rWiSS(s1,s2)
t1  ="Lines for corner are almost "+
& "parallel (angle<5°)!"+tC(13,10)+
& "The mitred corner cannot be "+
& "constructed."
t2  = "Error - Construction mitred "+
& "corner"
If(rAbs(rWi)<<5) Then
  nMsg= nIBox(t1,t2,21)
  Exit Program
Else If(rAbs((rAbs(rWi)-180))<<5) Then
  nMsg= nIBox(t1,t2,21)
  Exit Program
End If
```

If the angle between the lines `s1` and `s2` is almost 0° or almost 180° (between –5°/5°, 175°/180° or –175°/-180°), the message according to Picture 19-5 appears and the program is aborted.

	Icon:	No Icon	Exclamation mark 	Info Icon 	Question mark 	Cross as Warning sign 
Buttons:		**+0**	**+1**	**+2**	**+3**	**+4**
End/Repeat/Ignore	**10+**	10	11	12	13	14
OK	**20+**	20	21	22	23	24
OK/Abort	**30+**	30	31	32	33	34
Repeat/Abort	**40+**	40	41	42	43	44
Yes/No	**50+**	50	51	52	53	54
Yes/No/Abort	**60+**	60	61	62	63	64

Picture 19-4

The parameters for calling

```
nMsg= nIBox(t1,t2,21)
```

are:

`t1` message text; here: „Lines for corner are almost parallel ...“

Picture 19-5

`t2` title of the infobox; here: „Error – Construction mitred corner“

`21` the number for the display option; here: 21 for the icon exclamation mark and the button „OK“.

cPick()

With `cPick()`, analogous to `nIBox`, a message appears on the screen which asks the user to click a point or a line. All information about the clicked object are gathered in a container and can be taken out of the container as required.

For collar and sleeve programs, dialogues can be developed with `cPick()` in which the user is guided through clicking the required lines. The use of z values for transfer of lengths and distances to the programming language program is not longer necessary.

Example:

For the calculation of the total length of the necklines in front and back, the following program lines are to be entered:

```
'-------------------- allocate qF, qB
  qF = qKop(pXY(0,0)+pXY(100,0))
  qB = qKop(pXY(0,0)+pXY(50,0))
'---------------------- click qF, qB
  tF ="Click neckline FRONT !"
  cF = cPick(1,4,tF,"Collar",nT)
  qF = qCo(cF,"qq")
  tB ="Click neckline BACK !"
  cB = cPick(2,4,tB,"Collar",nT)
  qB = qCo(cB,"qq")
'------------ calculate target length
  rZl= rLngQ(qF)+rLngQ(qB)
```

Calling `cPick`, first fills a container; here: `cF` and `cB`. From this container, the required information is then, taken with the function `qCo`.

The parameters of the function `cPick()` have the following significance:

```
cPick(nI,nV,tC,tT,tP,nT)
```

`nI` unambiguous identifier; this identifier must be different for each call of cPick(). Thus, the different click instructions can be identified during test run and grading.

`nV` click variation
- 1 click point
- 2 click line/curve fulcrum
- 4 click line/curve
- 8 „freehand“ click permitted

The individual variations can be combined by adding. Thus, nV=3 means clicking of points and line/curve fulcrums.

`tC` comment text; With this text, the user is requested to click.

`tT` title text for the dialogue box

`tP` picture/symbol text; permitted are , “!“, “+“, “i“, “g“ or the complete path to a bitmap.

`nT` during call: number of the part in which clicking is permitted. nT=0 permits clicking in all parts with a part number <= the active part.
after call: number of the part which was clicked.

The following information can be called from the click container with the following functions. The first parameter is the variable name of the respective click container.

`nCo(c,"tl")`	Number of the piece which was clicked
`nCo(c,"nr")`	Pos-Number of the clicked object
`nCo(c,"ty")`	Type of the clicked object (-1: error, 0- digi point, 1- point, 2- line/curve)
`lCo(c,"rl")`	line/curve was clicked on the right (YES/NO)
`lCo(c,"st")`	click point is line/ curve fulcrum (YES/NO)
`lCo(c,"ri")`	curve direction in the click point in degrees
`rCo(c,"rln")`	relative length of the curve in the click point in %
`pCo(c,"pp")`	click point
`qCo(c,"qq")`	clicked line/curve

Proportion classes

The ease in a basic block can be adjusted via proportion classes or via x values. Which of the two options is used or whether both options are mixed, is decided by the programmer.

If proportion classes are used, the programmer defines the ease at waist, hip, bust, drop armhole... per proportion class in the program. If proportion classes are not used, the user adjusts the ease, later with x values.

The applicable proportion class is determined in the size table and transferred as a whole number value with the function `nZKlasse()`.

In a programming language program with the instruction `nZkl=nZKlasse()`, the variable `nZkl` has the following value for the respective measurement chart during processing:

Size table	value for nZKlasse()
> 01 e04 ____40_0	4
> 02 e04 ____42_0	4
> 03 e04 ____44_0	4
> 04 c02 ____40_0	2
> 05 g06 ____40_0	6
> 06 i08 ____40_0	8

The programmer determines the amount of ease to be added to bust, waist ... with the calculation of `nZKlasse`. With the instructions

```
'------------------------- Bust width
  rBu = rG(1)+10*nZKlasse()
'------------------------- Waist width
  rWa = rG(4)+13*nZKlasse()
'-------------------------- Hip width
  rHi = rG(2)+15*nZKlasse()
'----------------------- Drop armhole
  rAh = 2*nZKlasse()
```

the bust width rBu for half of the garment is calculated as follows:

Size table	value of rBu	ease
> 01 e04 ____40_0	920+10*4	+40
> 02 e04 ____42_0	960+10*4	+40
> 03 e04 ____44_0	1000+10*4	+40
> 04 c02 ____40_0	920+10*2	+20
> 05 g06 ____40_0	920+10*6	+60
> 06 i08 ____40_0	920+10*8	+80

Per proportion class, 10 mm ease are added to the bust, 13 mm ease at the waist, 15 mm ease at the hip. At the same time, the armhole is dropped by 2 mm per proportion class. With the factors before `nZKlasse()`, the programmer determines the ease per proportion class. If the change of ease is to be irregular in the different proportion classes, the IF-ENDIF structure is to be used.

For tight-fitting garments, e.g. swimwear or sportswear, the calculation can also be `10*nZKlasse()-40`. Thus, the measurement in proportion class a00 is 4cm smaller than the body measurement.

Circle functions

Circles are treated like curves in the new programming language. All curve functions can also be applied to circles. A circle can be created with the functions

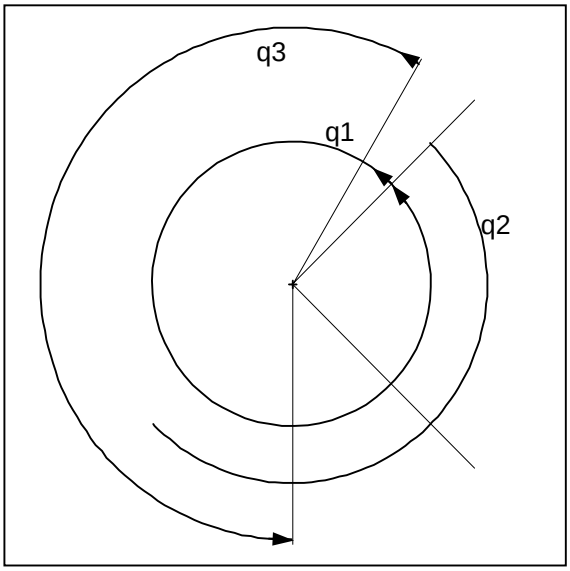
```
qTeilKr()
qHalbKr()
qVollKr()
```

As parameters, the centre point of the circle (point variable) and the radius of the circle (real variable) are to be entered. Depending on the circle type, direction assignments follow.

If the circle is required to create an intersection with a line or another curve, the function `qVollKr()` (full circle) should always be used. For creation of intersections, a direction point must be entered which points to the required intersection if more than one option are available. The circle centre is unsuitable as a direction point.

Example:

In the following example program, each circle variation is defined and then, output (Picture 19-6).



Picture 19-6

The parameter 45 in the instruction

```
q1  = qVollKr(p0,rRad1,45)
```

determines that the symmetry axis of the full circle (start and end) is the 45° direction.

The parameter –45 in the instruction

```
q2  = qHalbKr(p0,rRad2,-45)
```

determines that the symmetry axis of the semicircle is the –45° direction.

The parameter 60 and –90 in the instruction

```
q3  = qTeilKr(p0,rRad3,60,-90)
```

determine that the partial circle starts at 60° and ends at –90°. The circle is created in a mathematically positive direction.

```
'************************************
  Program Main()   ' Circle variations
'------------------------------------
  rVar rRad1,rRad2,rRad3
  qVar q1,q2,q3
  pVar p0
'------------------------ Centre point
  p0  = pXY(0,0)
'---------------------------- Radius
  rRad1  = 100
  rRad2  = 140
  rRad3  = 180
```

```
'---------------------- define circles
  q1  = qVollKr(p0,rRad1,45)
  q2  = qHalbKr(p0,rRad2,-45)
  q3  = qTeilKr(p0,rRad3,60,-90)
'----------------------------- Output
  AusP(p0)
  AusQ(q1,q2,q3)
  End Program
'************************************
```

External functions

For more transparency, often required programming steps can be saved as external functions. A external function (here: type n) begins with

Function nXxx([*parameter list*]**)**

and ends with

End Function

The function name is to be created analogous to a variable name and is to be assigned with a variable type. Within the function, the function name is a variable which can be assigned with a value. This value is returned after processing of the function. When calling the function, any number of variables of different types can be transferred to the function. The amount of parameters and the parameter type must be identical in the call of the function and the function definition. The parameters in the function call are also returned. External functions can be saved in their own modules. In the future, special modules for curves, corners etc. can be developed which will be made available to other Grafis programmers as libraries. The structures `Program Main()/ End Program` and `Function xXxx()/ End Function` must not be interlocked with one another.

Example:

```
'************************************
  Program Main()
  [instructions]
  lIo=lEck(p20,p21,p27,p28,p31)
  [instructions]
  lIo=lEck(p31,p37,p56,p57,p57a)
  [instructions]
  End Program
'************************************

'************************************
  Function lEck(p1,p2,p3,p4,pEck)
'  Calculation of the intersection
'  of two lines
'  The points p1 and p2 create the
'  first line,
'  the points p3 and p4 create the
'  second line.
'  The corner point pEck is returned
'  as fifth parameter.
'  If both lines are almost parallel
'  the program is aborted with
'  a warning.
'  Generated: 10-09-2000 KF
'************************************
  pVar
  sVar s1,s2
  rVar rWi
  tVar t1,t2
  nVar nMsg
'-------------------------------------
  lEck= False
  s1  = sPP(p1,p2)
  s2  = sPP(p3,p4)
'-------------------- lines parallel ?
  rWi = rWiSS(s1,s2)
t1  = "Lines for corner are almost "+
& "parallel (angle<5deg) !"+tC(13,10)+
& "The corner cannot be constructed."
t2  = "Error - Construction corner"
If(rAbs(rWi)<<5) Then
  nMsg= nIBox(t1,t2,21)
  Exit Function
Else If(rAbs((rAbs(rWi)-180))<<5) Then
  nMsg= nIBox(t1,t2,21)
  Exit Function
End If
  pEck= pSchnSS(s1,s2)
  lEck= True
  End Function
'************************************
```

The external function `lEck()` constructs a corner point which is defined by four points. The four points are transferred in the parameter list. The first and second point and the third and fourth point create a line, respectively. If the angle between the two lines smaller than 5°, the program is aborted with a message. Otherwise, the corner point `pEck` is calculated and returned as fifth parameter of the parameter list of the function. The function can be called a number of times (here: twice) with different points each time.

Often, logical functions are programmed which only end with the value `True`, if the function has been processed, correctly. To relocate the shoulder seam by x6 at the neck and x7 at the armhole, the function `lTranslShoulder()` must have the following structure:

```
'************************************
  Program Main()
  [instructions]
  lIs=lTranslShoulder(qNeckF,qNeckB,
&            qArmF,qArmB,rX(6),rX(7))
  [instructions]
  End Program
'************************************
'************************************
  Function lTranslShoulder(q1,q2,
&            q3,q4,r1,r2)
'  Transformation of q2 to q1 and
'  q4 to q3
'  relocate end point of q1 by r1
'  relocate end point of q3 by r2
'  create new curves
'  reverse transform new q2 and new q4
'************************************
  lTranslShoulder=False
  [instructions]
  q1=...
  q2=...
  q3=...
  q4=...
  lTranslShoulder=True
  End Function
'************************************
```

The new curves of the neck and armhole are transferred in the parameter list. Before processing the instruction line

```
  lIs=lTranslShoulder(qNeckF,qNeckB,
&            qArmF,qArmB,rX(6),rX(7))
```

in Main(), the shoulder seam of the curves `qNeckF`, `qNeckB`, `qArmF`, `qArmB` has not

been relocated. After processing the line, the curves are re-allocated and the shoulder is relocated.

19.2 Automatic length adjustment

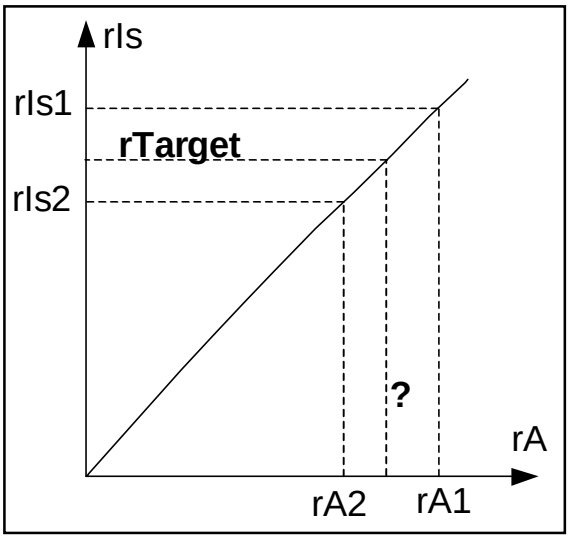
`rNahInt(``rA1,rIs1,`**`rA2,rIs2,`**`rTarg`**`)`**

The approximation interpolation `rNahInt()`is required for automatic length adjustment, e.g. for collars and sleeves. The collar is to be adjusted to the neck, the sleeve to the armhole.
For application of length adjustments, the following procedure applies:

1. Formulate a new, unambiguous construction instruction with a target length depending on other construction parameters.
2. Determine a variable construction parameter `rA`, which can be altered for adjustment of the target length `rTarg`.
3. Allocate a zero approximation. Example: For `rA1=0`, the result would be `rIs1=0`.
4. First development of the construction with a suitable starting value `rA2` for the variable construction parameter up to actual length `rIs2` and calculation of actual length `rIs2`. Depending on the extent, the development steps can be programmed as a external function.
5. Calculate the new starting value `rA` with the function `rNahInt()`.
6. Repeated development of the construction with the new starting value `rA` and calculation of the actual length `rIs`.
7. Query, whether the target length has been achieved.
   If yes, the construction is continued according to point 8.
   If no, the values of `rA2` and `rIs2` are switched to the variables `rA1` and `rIs1`. The variables `rA2` and `rIs2` then, obtain the values of `rA` and `rIs` which were last calculated. Then, a new value is calculated for `rA` according to point 5.
8. Finish the construction.

The function
`rNahInt(``rA1,rIs1,`**`rA2,rIs2,`**`rTarg`**`)`**

The function `rNahInt(`**`rA1`**`,`**`rIs1`**`,`**`rA2`**`,`**`rIs2`**`,`**`rTarg`**`)`calculates the probable starting value from two starting values rA1 and rA2, and the corresponding result values rIs1 and rIs2, with the result rTarg.
A mainly linear interdependence between starting value and result value is assumed (Picture 19-7).



Picture 19-7

In the following collar band construction, the variable construction parameter is the distance between p1 and p3.

Example:
For a distance p1⇔p3 of 160mm, the collar neck is 165.8mm long and for 200mm the collar neck is 204.7mm long.
With the allocation

```
rA1  =160
rIs1 =165.8
rA2  =200
rIs2 =204.7
```

the required distance p1⇔p3 can be calculated for a target length of the collar neck of 183mm as follows:

```
rTarg=183
rA   =rNahInt(rA1,rIs1,rA2,
&             rIs2,rTarg)
```

This calculation results in a value of 177.6mm for the distance p1⇔p3. With this value, a collar neck of 182.9mm is obtained. A further calculation with the values

```
rA1  =200
rIs1 =204.7
rA2  =177.6
rIs2 =182.9
```

results in a value of 177.7mm for the distance p1⇔ p3. With this value, the collar neck has the required length of 183.0mm.

Construction: collar band with automatic length adjustment

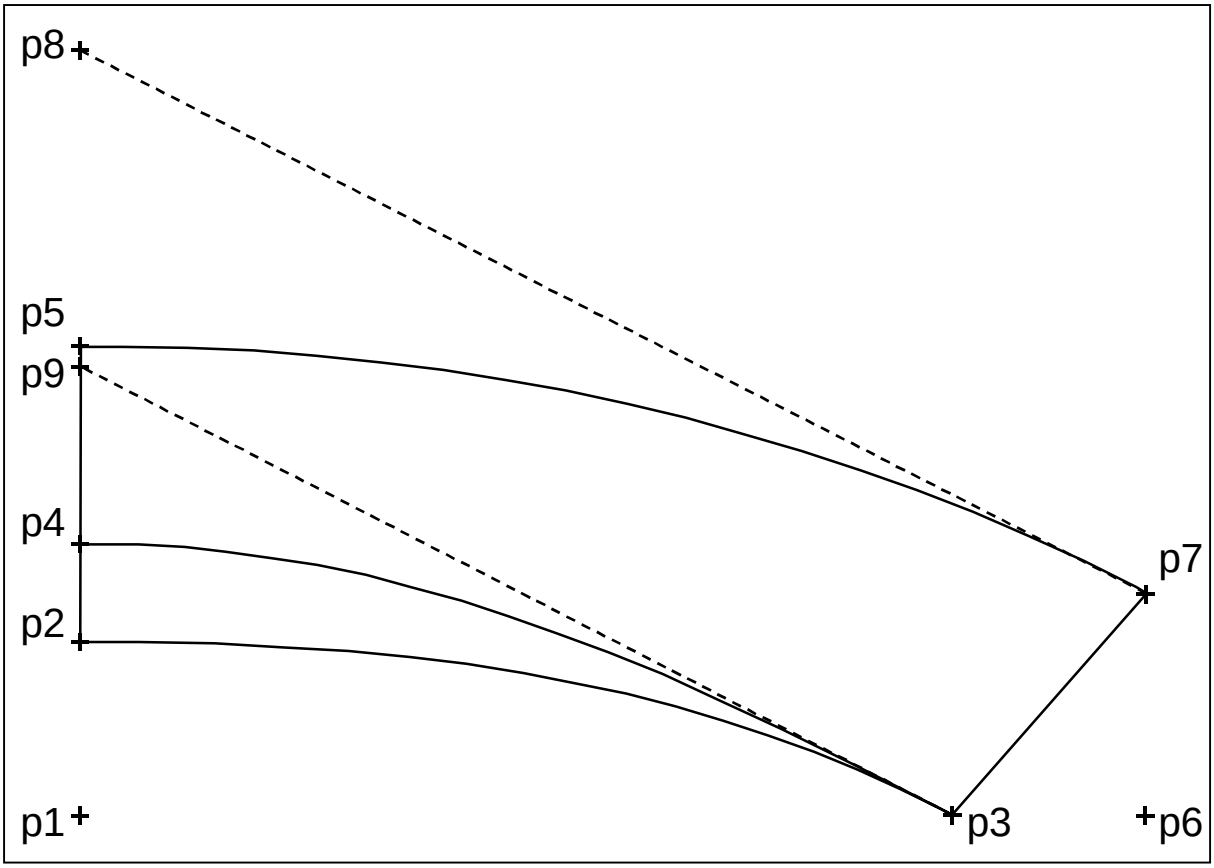
A collar band according to Picture 19-8 is to be programmed with application of the following x values:

X	Definition	Step	Value
1	addition collar length		0mm
2	raise CB	p1⇒p2	35mm
3	collar fall width	p2⇒p4	20mm
4	collar width CB	p4⇒p5	40mm
5	collar point (x) to p3	p3⇒p6	40mm
6	collar point (y) to p3	p6⇒p7	45mm
7	factor for the direction p. collar neck dep. on x2	p1⇒p9	2.6
8	direction p. f. collar edge	p1⇒p8	155mm

Construction steps:

from	to	direction	distance
1	2	⇑	x2 (raise CB)
2	4	⇑	x3 (collar fall width)
4	5	⇑	x4 (collar width)
1	8	⇑	x8 (dir. p. collar edge)
1	9	⇑	x2*x7
1	3	⇒	**variable distance**, so that «collar neck» = «neck meas. +x1»
			construct and optimise collar neck
3	6	⇒	x5
6	7	⇑	x6
			construct collar fold and collar edge

All curves are to run into the CB at right angle. The collar fold and the collar neck have the direction p3 ⇒p9 in p3. The collar edge has the direction p7⇒p8 in p7.



Picture 19-8

The program: collar band with automatic length adjustment

```
'**************************************************************************
  Program Main()
' Construction of a collar band after instructions by
' Mrs. Prof. H.Brückner, Berlin
' The user has to click the necklines in the front and back
' The collar is automatically constructed, so that the length of the collar
' neck equals the length of both necklines plus addition x1.
'--------------------------------------------------------------------------
  lVar
  nVar n
  rVar rWi3,rWi7,rA,rA1,rA2,rIs,rIs1,rIs2,rClLng,rTarg
  pVar p1,p2,p3,p4,p5,p6,p7,p8,p9
  sVar
  qVar q1,q2,q3,qF,qB
  tVar
  cVar cF,cB
'--------------------------------------------------------------------------
  lCon
  nCon
  rCon rRi=0,rLe=180,rUp=90,rDo=270
  rCon
  tCon
'------------------------------------------------- x value definitions
  XTitel("collar band")
  Defx(1,"addition collar length",0)
  Defx(2,"raise CB",35)
  Defx(3,"collar fold width",20)
  Defx(4,"collar width CB",40)
  Defx(5,"collar point (x) in relation to p3",40)
  Defx(6,"collar point (y) in relation to p3",45)
  Defx(7,"factor for direction point neck relating to x2",2.6)
  Defx(8,"direction point for collar edge",155)
'------------------------------------------- query length of necklines
  cF = cPick(1,4,"Click neckline FRONT !","collar","!",nT)
  If (not lCo(cF, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qF = qCo(cF,"qq")
  cB = cPick(2,4,"Click neckline BACK !","collar","!",nT)
  If (not lCo(cB, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qB = qCo(cB,"qq")
  rClLng = rLngQ(qF)+rLngQ(qB)
  If(rClLng<<rX(2)) Then
    n   = nIBox("The necklines are too short !")
    Exit Program
  End If
'------------------------------------------------------- points at CB
  p1  = pXY(0,0)
  p2  = pXY(0,rX(2))
  p4  = pPRiLng(p2,rUp,rX(3))
  p5  = pPRiLng(p4,rUp,rX(4))
  p8  = pPRiLng(p1,rUp,rX(8))
  p9  = pPRiLng(p1,rUp,rX(2)*rX(7))
'----------------------------------------- collar neck 0 approximation
  rTarg  = rClLng+rX(1)
  rA  = rClLng
  p3  = pXY(rA,0)
  rWi3= rRiPP(p3,p9)
  q1  = qSpline(p3,rWi3,p2,rLe)
  rIs= rLngQ(q1)
```

```
'------------------------------------------------- values for 1st approx.
  rA1   = 0
  rIs1  = 0
  rA2   = rA
  rIs2  = rIs
'------------------------------------------------------- autom.approx.
  For n = 1,10,1
    rA  = rNahInt(rA1,rIs1,rA2,rIs2,rTarg)
    p3  = pXY(rA,0)
    rWi3= rRiPP(p3,p9)
    q1  = qSpline(p3,rWi3,p2,rLe)
    rIs= rLngQ(q1)
    If(rAbs(rIs-rTarg)<<0.01) Then
      Exit For
    End If
    rA1 = rA2
    rIs1  = rIs2
    rA2 = rA
    rIs2  = rIs
  End For
'------------------------------------------------------- collar point
  p6  = pPRiLng(p3,rRi,rX(5))
  p7  = pPRiLng(p6,rUp,rX(6))
'------------------------------------------------- collar fold line
  q2  = qSpline(p3,rWi3,p4,rLe)
'------------------------------------------------------- collar edge
  rWi7= rRiPP(p7,p8)
  q3  = qSpline(p7,rWi7,p5,rLe)
'------------------------------------------- output points + lines
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(p2+p5)
  AusQ(p3+p7)
  AusQ(q1,q2,q3)
'-------------------------------------------------------------------
  End Program
'*******************************************************************
```

19.3 Collar neck with minimum as external function

As basis for various collar developments, a external function `qClNeck1()` is to be programmed which delivers a collar neck (Picture 19-9) of a given length, considering the following parameters:

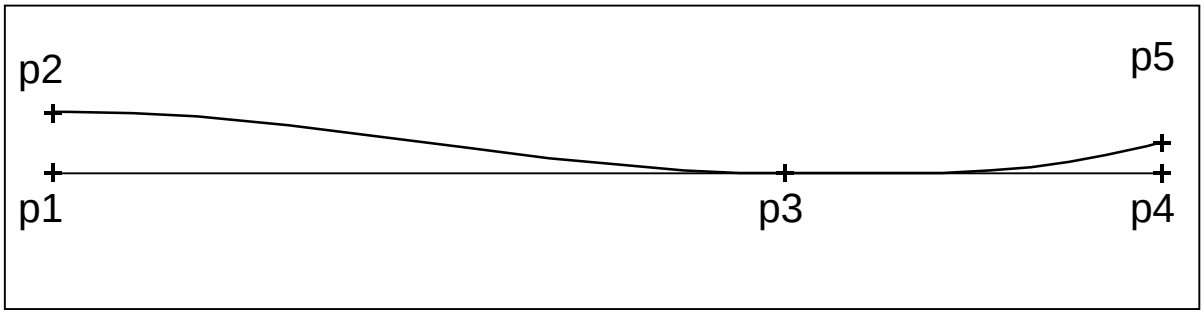
- raise centre back
- raise centre front
- additional direction at the CF
- position of minimum in % (from CB)

The next page shows a test environment for the function `qClNeck1()`. The actual function `qClNeck1()` is shown on the following page. The function `qClNeck1()` can also be saved in a new module, together with further functions for collar necks of different shapes.

Construction steps function `qClNeck1()`:

from	to	direction	distance
1	2	⇑	rCb (raise CB)
1	4	⇒	**variable distance**, This distance is optimised so that the collar neck equals the length of rTarg.
1	3	⇒	rMin /100* variable distance
4	5	⇑	rCf (raise CF)
			construct and optimise collar neck

The collar neck is to run into the centre back at right angle. In point p5 the curve is to have the direction p3⇒p5 plus the correction rRi5z.



Picture 19-9

```
'************************************************************************
  Program Main()
' Test environment for development of the function qClNeck1()
'------------------------------------------------------------------------
  nVar n
  rVar rClLng,rTarg
  pVar p1,p2,p3,p4,p5
  qVar qF,qB,q1
  cVar cF,cB
  rCon rRi=0,rLe=180,rUp=90,rDo=270
'------------------------------------------------------------- x values
  XTitel("collar neck for shirt collars")
  Defx(1,"addition to collar neck line",0)
  Defx(2,"raise CB",10)
  Defx(3,"raise CF",5)
  Defx(4,"additional direction collar stand in p5",10)
  Defx(5,"position p3 between p1-p4 in %",66)
'------------------------------------------- query length of necklines
  cF = cPick(1,4,"Click neckline FRONT !","collar","!",nT)
  If (not lCo(cF, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qF = qCo(cF,"qq")
  cB = cPick(2,4,"Click neckline BACK !","collar","!",nT)
  If (not lCo(cB, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qB = qCo(cB,"qq")
  rClLng  = rLngQ(qF)+rLngQ(qB)
  If(rClLng<<rX(2)) Then
    n    = nIBox("The necklines are too short !")
    Exit Program
  End If
  rTarg= rClLng+rX(1)
'------------------------------------------------------ collar neck line
  q1   = qClNeck1(rX(2),rX(3),rX(4),rX(5),rTarg,p1,p2,p3,p4,p5)
'------------------------------------------------------------------ output
  AusP(p1,p2,p3,p4,p5)
  AusQ(q1)
  End Program
'************************************************************************
```

```
'************************************************************************
  Function qClNeck1(rCb,rCf,rRi5z,rMin,rTarg,p1,p2,p3,p4,p5)
' ... calculates the collar neck line which has a minimum at rRi5 %
' between CB and CF. The parameters to be given:
'   rCb  - raise CB
'   rCf  - raise CF
'   rRi5 - additional direction in p5 (at CF)
'   rMin - position of minimum in percent (from CB)
'   rTarg- target length of the collar neck (incl. add) from CB to CF
' returned are the curve qClNeck1, which starts at CB,
' and the points p1 to p5.
'------------------------------------------------------------------------
  nVar n
  rVar rA,rRi5,rIs,rA1,rIs1,rA2,rIs2
'------------------------------------------------------------------------
  rCon rRi=0,rLe=180,rUp=90,rDo=270
'------------------------------------------------------- points at CB
  p1   = pXY(0,0)
  p2   = pPRiLng(p1,rUp,rCb)
'------------------------------------------------------ 0.approximation
  rA   = rTarg
  p3   = pPRiLng(p1,rRi,rMin/100*rA)
  p4   = pPRiLng(p1,rRi,rA)
  p5   = pPRiLng(p4,rUp,rCf)
  rRi5 = rRiPP(p3,p5)+rRi5z
  qClNeck1  = qSpline(p2,rRi,p3,rRi,p5,rRi5)
  rIs  = rLngQ(qClNeck1)
'------------------------------------------------ values for 1st approx.
  rA1  = 0
  rIs1 = 0
  rA2  = rA
  rIs2 = rIs
'------------------------------------------------------- autom. approx.
  For n = 1,10,1
    rA   = rNahInt(rA1,rIs1,rA2,rIs2,rTarg)
    p3   = pPRiLng(p1,rRi,rMin/100*rA)
    p4   = pPRiLng(p1,rRi,rA)
    p5   = pPRiLng(p4,rUp,rCf)
    rRi5 = rRiPP(p3,p5)+rRi5z
    qClNeck1  = qSpline(p2,rRi,p3,rRi,p5,rRi5)
    rIs  = rLngQ(qClNeck1)
    If(rAbs(rIs-rTarg)<<0.01) Then
      Exit For
    End If
    rA1  = rA2
    rIs1 = rIs2
    rA2  = rA
    rIs2 = rIs
  End For
  End Function
'************************************************************************
```

19.4 Shirt collar construction with application of the external function qClNeck1()

A shirt collar (Picture 19-10) with the following x values is to be constructed with application of the external function `qClNeck1()` from the previous section.

X	Definition	Step	Value
1	addition collar length		0mm
2	raise CB	p1⇒p2	10mm
3	raise CF	p4⇒p5	5mm
4	additional direction collar stand in p5		10°
5	stand width CB	p2⇒p6	25mm
6	collar width CB	p6⇒p7	65mm
7	overlap collar stand	p5⇒p8	20mm
8	reduce stand width at overlap	p8⇒p10	5mm
9	collar point in x	p4⇒p11	15mm
10	collar point in y	p4⇒p11	10mm
11	direction collar point		10°
12	position p3 between p1-p4	p1⇒p4	66%

Construction steps:

from	to	direct.	distance
1	2	⇑	x2 (raise CB)
1	4	⇒	**variable distance**, so that collar neck = neckline + x1
1	3	⇒	x12 /100* variable distance
4	5	⇑	x3 (raise CF)
			construct and optimise collar neck line

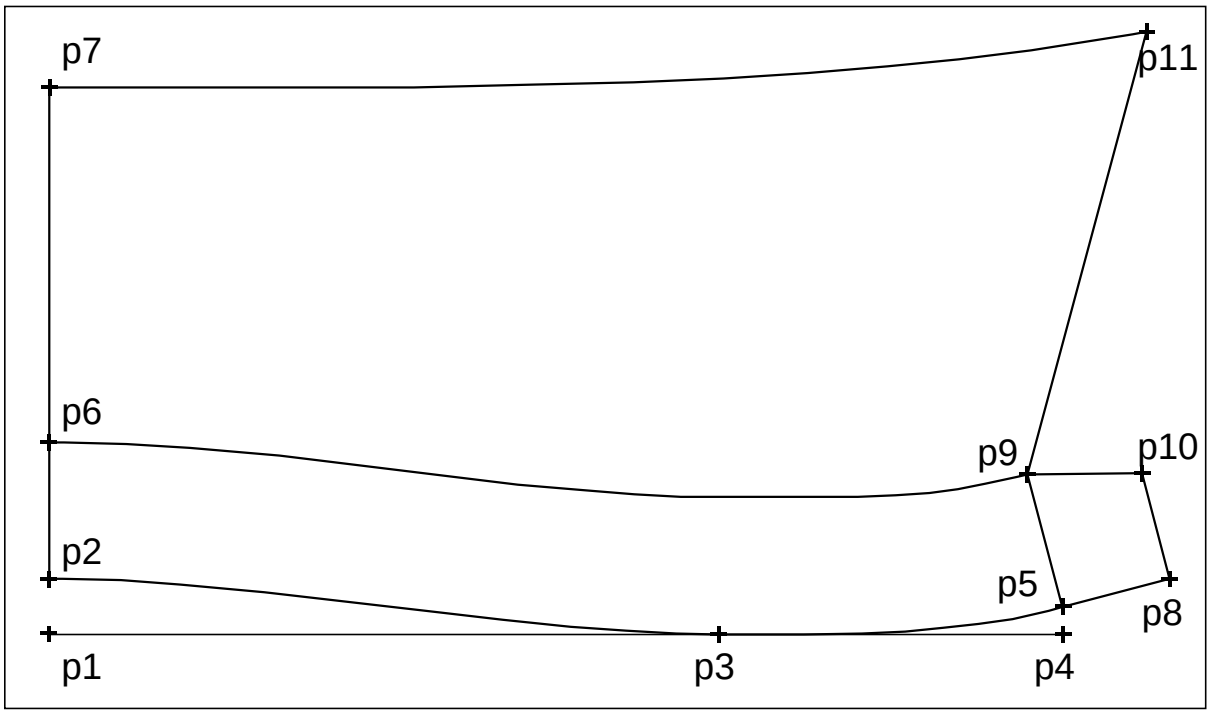
The steps thus far are processed by the external function `qClNeck1()`. All following steps are programmed in the program Main().

2	6	⇑	x5
			parallel to collar neck at distance x5
	9		end point of parallel
6	7	⇑	x6 (collar width)
5	8	collar neck in p5	x7 (overlap collar stand)
8	10	vertical as before	x5-x8 (reduce stand width at overlap)
4	11	⇑	x2+x5+x6+x10 (collar point overlap in y)
11	11	⇒	x9 (collar point in x)
			construct collar edge with direction x11 in p11

All curves are to run into the centre back at right angle.

Step-by-step guide:

Create the new project „shirt collar“ and open the new module „collar necks“ in this project with *Module | New...*. The module appears in the variable list under the section „Module“. Clicking, opens the module. Copy the tested function `qClNeck1()` from the previous section into the module „collar necks„ and compile the new module. Then, select the main module by clicking on Main.qpr in the „Module“ section and develop the shirt collar according to the program instructions on page 16.
For development of a different collar with the same collar neck line, the module „collar necks“ only has to be inserted. Thus, the function `qClNeck1()` can also be used, there.



Picture 19-10

Content of Module collarnecks.qpr:

```
'************************************************************************
  Function qClNeck1(rCb,rCf,rRi5z,rMin,rTarg,p1,p2,p3,p4,p5)
' ... calculates the collar neck line, which has a minimum at rRi5 %
' between CB and CF. The parameters to be given:
'   rCb  - raise CB
'   rCf  - raise CF
'   rRi5 - additional direction in p5 (at CF)
'   rMin - position of minimum in percent (from CB)
'   rTarg- target length for collar neck (incl. add) from CB to CF
' returned are the curve qClNeck1, which starts at CB,
' and the points p1 to p5.
'------------------------------------------------------------------------
  nVar n
  rVar rA,rRi5,rIs,rA1,rIs1,rA2,rIs2
'------------------------------------------------------------------------
  rCon rRi=0,rLe=180,rUp=90,rDo=270
'------------------------------------------------------------ points at CB
  p1   = pXY(0,0)
  p2   = pPRiLng(p1,rUp,rCb)
'---------------------------------------------------------- 0.approximation
  rA   = rTarg
  p3   = pPRiLng(p1,rRi,rMin/100*rA)
  p4   = pPRiLng(p1,rRi,rA)
  p5   = pPRiLng(p4,rUp,rCf)
  rRi5 = rRiPP(p3,p5)+rRi5z
  qClNeck1  = qSpline(p2,rRi,p3,rRi,p5,rRi5)
  rIs  = rLngQ(qClNeck1)
'------------------------------------------------- values for 1st approx.
  rA1  = 0
  rIs1 = 0
  rA2  = rA
  rIs2 = rIs
'------------------------------------------------------------ autom.approx.
  For n = 1,10,1
    rA   = rNahInt(rA1,rIs1,rA2,rIs2,rTarg)
    p3   = pPRiLng(p1,rRi,rMin/100*rA)
    p4   = pPRiLng(p1,rRi,rA)
    p5   = pPRiLng(p4,rUp,rCf)
    rRi5 = rRiPP(p3,p5)+rRi5z
    qClNeck1  = qSpline(p2,rRi,p3,rRi,p5,rRi5)
    rIs  = rLngQ(qClNeck1)
    If(rAbs(rIs-rTarg)<<0.01) Then
      Exit For
    End If
    rA1  = rA2
    rIs1 = rIs2
    rA2  = rA
    rIs2 = rIs
  End For
  End Function
'************************************************************************
```

Content of Module Main.qpr:

```
'************************************************************************
  Program Main()
' Shirt collar construction with automatic length adjustment
' of the collar neck line to the length of the measured neck
' after instructions by Mrs. Prof. H.Brückner, Berlin
'------------------------------------------------------------------------
  nVar n
  rVar rClLng,rTarg,rRi5,rRi8,rRi11
  pVar p1,p2,p3,p4,p5,p6,p9,p7,p8,p10,p11
  qVar qF,qB,q1,q2,q3
  cVar cF,cB
'------------------------------------------------------------------------
  rCon rRi=0,rLe=180,rUp=90,rDo=270
'------------------------------------------------------------ x values
  XTitel("shirt collar")
  Defx(1,"addition to collar neck line",0)
  Defx(2,"raise CB",10)
  Defx(3,"raise CF",5)
  Defx(4,"additional direction collar stand in p5",10)
  Defx(5,"collar stand width CB",25)
  Defx(6,"collar width CB",65)
  Defx(7,"overlap collar stand",20)
  Defx(8,"reduce stand width at overlap",5)
  Defx(9,"collar point overlap in x direction",15)
  Defx(10,"collar point overlap in y direction",10)
  Defx(11,"direction collar point",10)
  Defx(12,"position p3 between p1-p4 in %",66)
'------------------------------------------- query length of necklines
  cF = cPick(1,4,"Click neckline FRONT !","collar","!",nT)
  If (not lCo(cF, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qF = qCo(cF,"qq")
  cB = cPick(2,4,"Click neckline BACK !","collar","!",nT)
  If (not lCo(cB, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qB = qCo(cB,"qq")
  rClLng = rLngQ(qF)+rLngQ(qB)
  If(rClLng<<rX(2)) Then
    n   = nIBox("The necklines are too short !")
    Exit Program
  End If
  rTarg  = rClLng+rX(1)
'------------------------------------------------------ collar neck line
  q1  = qClNeck1(rX(2),rX(3),rX(4),rX(12),rTarg,p1,p2,p3,p4,p5)
'------------------------------------------------------ points p6 to p11
  p6  = pPRiLng(p2,rUp,rX(5))
  Paral(-rX(5):q2=q1)
  p9  = pQend(q2)
  p7  = pPRiLng(p6,rUp,rX(6))
  rRi5= rRiQend(q1)
  p8  = pPRiLng(p5,rRi5,rX(7))
  rRi8= rRi5+90
  p10 = pPRiLng(p8,rRi8,rX(5)-rX(8))
  p11 = pPRiLng(p4,rUp,rX(2)+rX(5)+rX(6)+rX(10))
  p11 = pPRiLng(p11,rRi,rX(9))
'------------------------------------------------------------ collar edge
  rRi11  = rX(11)
  q3  = qSpline(p7,rRi,p11,rRi11)
'------------------------------------------------------------------ output
  AusP(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11)
  AusQ(p2+p7,p9+p11,p5+p8,p8+p10,p10+p9,p9+p5)
  AusQ(q1,q2,q3)
  End Program
'************************************************************************
```

19.5 Construction component shoulder seam relocation with replacing Pos-objects

So far, only new objects were transferred to the Grafis record. For construction components which are to be applied to existing objects, it must be possible to access objects of the Grafis record. This is possible with the so-called Pos-numbers.

Each object (point, line) of the Grafis record has an unambiguous Pos-number for identification.

The Pos-number applies within one part and is automatically entered for each new object. All

construction steps of the Grafis record point to the respective object via the Pos-number.

The application of Pos-numbers

Existing objects which are to be processed with a construction component, are identified with the function `cPick()`. The Pos-numbers can be extracted from the click container as whole number parameters. Example:

```
cXx=cPick(1,1,tC,tT,tP,nT)
```

The significance of the parameters is explained on page 19-5.

The Pos-number of the clicked point is extracted from this container as follows:

```
nXx=nCo(cXx,"nr")
```

A point or a line of the Grafis record is replaced by entering also the Pos-number of the object to be replaced for output from the programming language program. In output instructions with Pos-numbers, only one object can be output, respectively. The Pos-numbers can be entered for points, lines and curves. The Grafis record does not differentiate between lines and curves; both are regarded as lines.

`AusP(nXx,pXx)`

nXx	Pos-number of the point
pXx	point to be output

`AusQ(nXx,qXx)`

nXx	Pos-number of the line
qXx	line/curve to be output

Application of Pos-numbers in the construction component „shoulder relocation“

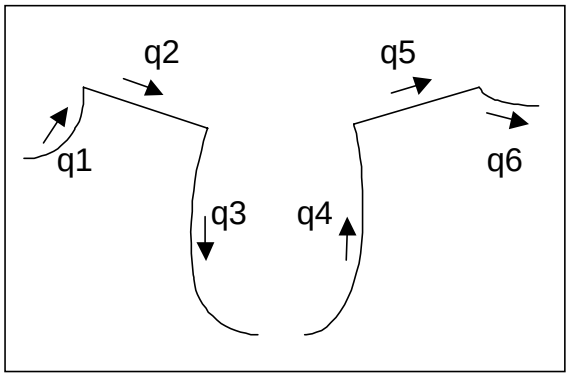
Apart from the application of Pos-numbers, this example should also clarify:

- The creation of a meaningful dialogue with the user. First, the user is given the information and then, is requested to click the required objects.
- The safety of the program has great significance. The program must later „react“, safely to the different cases of applications. For the shoulder relocation, the length of the shoulders is compared and each corner is checked for gaps and protruding lines.
- Each construction component should alter the original objects as little as possible, even though at first, no negative effects may be recognised. Therefore, the original orientation of the lines is reproduced at the end of the construction component.
- A sufficient comment is necessary. This is irrespective of the fact, whether subsequent alterations/corrections are carried out by the original programmer or another person. When working with the construction component shoulder relocation, decide whether the comment is sufficient for you.
- A picture with point and line annotations according to Picture 19-11 is also part of each project. This also applies to a short description of the procedure.

For the shoulder relocation, objects according to Picture 19-11 are assumed or prepared.

The programming language program for the shoulder seam is organised as follows:

- create and display info mask for the user. The user is informed about the pre-requisites and the application options.



Picture 19-11

- The curves are suitably allocated, so that the program can be tested in the development environment.
- Request the user to click the front neckline. The Pos-number of the ft neck is saved to `nPos1` in the process.
- Request the user to click the front shoulder. The Pos-number of the ft shoulder is saved to `nPos2` in the process.
- With the external function lcurve_orient_all(), orientate the neckline and shoulder according to Picture 19-11 check the corner. If no clear corner is found, abort with a message. Whether a curve was re-orientated or not, is saved under the variables lRota1 etc.

- Request the user to click the front armhole. The Pos-number of the ft armhole is saved under `nPos3`.
- With the external function `lcurve2_orient()`, orientate the armhole line according to Picture 19-11 and check the corner. If no clear corner is found, abort with a message.
- Request the user to click the back armhole. The Pos-number of the bk armhole is saved as `nPos4`.
- Request the user to click the back shoulder. The Pos-number of the bk shoulder is saved as `nPos5`.
- With the external function `lcurve_orient_all()`, orientate the armhole and shoulder according to Picture 19-11 and check the corner. If no clear corner is found, abort with a message.
- Compare the front and back shoulder and abort with a message, if necessary.
- Request the user to click the back neckline. The Pos-number of the bk shoulder is saved as `nPos6`.
- With the external function `lKurve2_orient()`, orientate the neckline according to Picture 19-11 and check the corner. If no clear corner is found, abort with a message.

Now, the actual transformation steps follow:

- Transform back armhole and shoulder to the front and then link.
- Relocate shoulder point at the neck by x1 and at the armhole by x2.
- Allocate the curves, again and reverse the transformation.
- If a curve has been rotated, return it to its original orientation.
- Replace the curves of the construction record with the new curves. The curves are output onto the original Pos-numbers.

The complete program Main() with the external functions `lcurve_orient_all` and `lcurve2_orient` in the Module „addition„ follows.

Content of Module Main.qpr: **Page 1/4**

```
'************************************************************************
  Program Main()
'------------------------------------------------------------------------
' Construction component relocate shoulder
'------------------------------------------- Information for the user
' The relocate amount are controlled via x values.
' The construction component requires:
' - shoulder lengths identical in ft and bk
' - straight shoulder lines (no curves) and
' - no gaps between the curves and the shoulder lines.
'----------------------------------- Internal information of the program
' q1: neckline ft, q2: shoulder ft, q3: armhole ft
' q4: armhole bk, q5: shoulder bk, q6: neckline bk
' lRota1 to lRota6 show, whether the curve has be re-orientated.
' At the end of the program, the curves are output in their
' original orientation.
'------------------------------------------------------------------------
  lVar l,lRota1,lRota2,lRota3,lRota4,lRota5,lRota6
  nVar n,nPos1,nPos2,nPos3,nPos4,nPos5,nPos6,nT
  rVar r1,r2
  pVar pSneck,pSarmh
  sVar s1,s2
  qVar q1,q2,q3,q4,q5,q6,q1t,q3t
  tVar tInfo,t,tGen,tTop,t1
  cVar c1,c2,c3,c4,c5,c6
```

Content of Module Main.qpr: **Page 2/4**

```
'-------------------------------------------------------------------------
  lCon
  nCon
  rCon
  tCon
'------------------------------------------------------------- x values
  XTitel("Relocate shoulder")
  Defx(1,"relocate amount at neckline to ft",10)
  Defx(2,"relocate amount at armhole to ft",10)
'---------------------------------------------- allocate the curves
  q1   = qKop(pXY(0,0)+pXY(0,20))
  q2   = qKop(pXY(0,20)+pXY(20,20))
  q3   = qKop(pXY(20,20)+pXY(20,-20))
  q4   = qKop(pXY(30,-20)+pXY(30,20))
  q5   = qKop(pXY(30,20)+pXY(50,20))
  q6   = qKop(pXY(50,20)+pXY(50,0))
'------------------------------------------- query and check objects
  tGen = "create no clear corner."+tC(13,10)+
&        "Correct the corner and relocate"+tC(13,10)+
&        "the shoulder again !"
  tTop = "Relocate shoulder"
'------------------------------------------------------------ neckline ft
  tInfo= "Click front neckline !"
  nT   = 0
  c1   = cPick(1,4,tInfo,tTop,"i",nT)
  q1   = qCo(c1,"qq")
'------------------------------------------------------------ shoulder ft
  tInfo= "Click front shoulder !"
  c2   = cPick(2,4,tInfo,tTop,"i",nT)
  q2   = qCo(c2,"qq")
  l    = lcurve_orient_all(q1,q2,lRota1,lRota2)
  If(Not l) Then
    t    = "The ft neckline curve and the ft shoulder"+tC(13,10)+tGen
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'---------------------------------------- shoulder line straight line?
  If(rAbs(rLngQ(q2)-rAbstPP(pQanf(q2),pQend(q2)))>>0.05) Then
    t    = "The shoulder line is curved."+tC(13,10)
&         +"This case is not prepared."
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'------------------------------------------------------------ armhole ft
  tInfo= "Click ft armhole !"
  c3   = cPick(3,4,tInfo,tTop,"i",nT)
  q3   = qCo(c3,"qq")
  l    = lcurve2_orient(q2,q3,lRota3)
  If(Not l) Then
    t    = "The curves ft shoulder and ft armhole"+tC(13,10)+tGen
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'------------------------------------------------------------ armhole bk
  tInfo= "Click back armhole !"
  c4   = cPick(4,4,tInfo,tTop,"i",nT)
  q4   = qCo(c4,"qq")
'------------------------------------------------------------ shoulder bk
  tInfo= "Click back shoulder !"
  c5   = cPick(5,4,tInfo,tTop,"i",nT)
  q5   = qCo(c5,"qq")
  l    = lcurve_orient_all(q4,q5,lRota4,lRota5)
```

Content of Module Main.qpr: **Page 3/4**

```
  If(Not l) Then
    t    = "The curves bk armhole and back shoulder"+tC(13,10)+tGen
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'------------------------------------------------ shoulder straight line?
  If(rAbs(rLngQ(q5)-rAbstPP(pQanf(q5),pQend(q5)))>>0.05) Then
    t    = "The shoulder line is curved."+tC(13,10)
&         +"This case is not prepared."
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'------------------------------------------------ compare shoulder lines
  If(rAbs(rLngQ(q2)-rLngQ(q5))>>0.5) Then
    t    = "The shoulder lines in ft and bk"+tC(13,10)
&         +"have different lengths !"
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'------------------------------------------------------------ neckline bk
  tInfo= "Click back neckline !"
  c6   = cPick(6,4,tInfo,tTop,"i",nT)
  q6   = qCo(c6,"qq")
  l    = lcurve2_orient(q5,q6,lRota6)
  If(Not l) Then
    t    = "The curves bk shoulder and bk neckline"+tC(13,10)+tGen
    n    = nIBox(t,"Relocate shoulder",24)
    Exit Program
  End If
'---------------------------- transform back neckline and armhole curves
  s1   = sPP(pQanf(q5),pQend(q5))
  s2   = sPP(pQend(q2),pQanf(q2))
  DrehTr(s1,s2:q4,q6)
'------------------------------------------------------------ link curves
  q1t  = qKop(q1+q6)
  q3t  = qKop(q4+q3)
'------------------------------------------------ relocate shoulder point
  pSneck  = pQend(q1)
  pSarmh  = pQanf(q3)
  pSneck  = pQPlng(q1t,pSneck,-rX(1))
  pSarmh  = pQPlng(q3t,pSarmh,rX(2))
'------------------------------------------------ create all curves again
  q1   = qQbisP(q1t,pSneck)
  q2   = qKop(pSneck+pSarmh)
  q3   = qQabP(q3t,pSarmh)
  q4   = qQbisP(q3t,pSarmh)
  q5   = qKop(pSarmh+pSneck)
  q6   = qQabP(q1t,pSneck)
'------------------------------------------------ reset curves in the bk
  DrehTr(s2,s1:q4,q5,q6)
```

Content of Module Main.qpr: **Page 4/4**

```
'---------------------------------- rotate curves in original direction
  If(lRota1) Then
    q1   = -q1
  End If
  If(lRota2) Then
    q2   = -q2
  End If
  If(lRota3) Then
    q3   = -q3
  End If
  If(lRota4) Then
    q4   = -q4
  End If
  If(lRota5) Then
    q5   = -q5
  End If
  If(lRota6) Then
    q6   = -q6
  End If
'------------------------------------------------------------------ output
  AusQ(nCo(c1,"nr"),q1)
  AusQ(nCo(c2,"nr"),q2)
  AusQ(nCo(c3,"nr"),q3)
  AusQ(nCo(c4,"nr"),q4)
  AusQ(nCo(c5,"nr"),q5)
  AusQ(nCo(c6,"nr"),q6)
'-------------------------------------------------------------------------
  End Program
'*************************************************************************
```

Content of Module: addition.qpr **Page 1/2**

```
'*************************************************************************
  Function lcurve_orient_all(q1,q2,lRota1,lRota2)
'-------------------------------------------------------------------------
' The curves are orientated so that the end point of the first curve
' is positioned, directly at the beginning of the second. If the
' curves were oriented, correctly the function has the value True,
' other wise the value False.
'-------------------------------------------------------------------------
  nVar n
  qVar q1t,q2t
  pVar pEndq1,pAnfq2
'----------------------------------- query variations, re-orientate curve
  lcurve_orient_all= False
  For n = 1,4,1
    If(n==1) Then
      q1t  = q1
      q2t  = q2
      lRota1  = False
      lRota2  = False
    Else If(n==2) Then
      q1t  = -q1
      q2t  = q2
      lRota1  = True
      lRota2  = False
```

Content of Module: addition.qpr **Page 2/2**

```
    Else If(n==3) Then
      q1t  = q1
      q2t  = -q2
      lRota1  = False
      lRota2  = True
    Else If(n==4) Then
      q1t  = -q1
      q2t  = -q2
      lRota1  = True
      lRota2  = True
    End If
    pEndq1  = pQend(q1t)
    pAnfq2  = pQanf(q2t)
    If(rAbs(rAbstPP(pEndq1,pAnfq2))<<0.5) Then
      lcurve_orient_all= True
      q1   = q1t
      q2   = q2t
      Exit For
    End If
  End For
  End Function
'*************************************************************************
'*************************************************************************
  Function lcurve2_orient(q1,q2,lRota2)
'-------------------------------------------------------------------------
' The SECOND curve is orientated so that the end point of the first curve
' is positioned, directly at the beginning of the second. If the two
' curves were orientated, correctly the function has the value True,
' other wise it has the value False.
' lRota2 is True, if q2 was rotated.
' q1 is not rotated.
'-------------------------------------------------------------------------
  nVar n
  qVar q1t,q2t
  pVar pEndq1,pAnfq2
'--------------------------------- query variation, re-orientate curves
  lcurve2_orient  = False
  For n = 1,2,1
    If(n==1) Then
      q1t  = q1
      q2t  = q2
      lRota2  = False
    Else If(n==2) Then
      q1t  = q1
      q2t  = -q2
      lRota2  = True
    End If
    pEndq1  = pQend(q1t)
    pAnfq2  = pQanf(q2t)
    If(rAbs(rAbstPP(pEndq1,pAnfq2))<<0.5) Then
      lcurve2_orient  = True
      q1   = q1t
      q2   = q2t
      Exit For
    End If
  End For
'-------------------------------------------------------------------------
  End Function
'*************************************************************************
```