# Kapitel 19 „Fachsprache II“



**Inhalt**



Mit den in Kapitel 18 behandelten Befehlen und Funktionen kann bereits die Mehrzahl aller Grundkonstruktionen in ein Fachsprachenprogramm umgesetzt werden. Im ersten Abschnitt werden diverse spezielle Programmierstrukturen und –funktionen behandelt. Gegenstand des zweiten Abschnittes ist die automatische Längenanpassung am Beispiel eines Kragens. Danach folgt die Nutzung Externer Funktionen und das Erstellen von Konstruktionsbausteinen.

```
'************************************************************
  Program Main()
'------------------------------------------------------------
' Kragenkonstruktion mit automatischer Längenanpassung       .
'------------------------------------------------------------
  nVar n,nNextPos,nT
  rVar rZl,rA,rL,rA1,rL1,rA2,rL2
  pVar p0,p1,p2,p3,p4,p5
  qVar q1,q2,q3,qV,qH,qQqq
  cVar cV,cH
'---------------------------------------- Definition der XWerte
  XTitel("XWerte der Hemdkragen-Konstruktion")
  Defx(1,"Auslaufrichtung der Kragenansatzlinie",-45)
  Defx(2,"Auslaufrichtung der Steglinie",-55)
  Defx(3,"Hochstellbetrag Kragen",35)
  Defx(4,"Stegbreite",15)
  Defx(5,"Hintere Kragenbreite",45)
  Defx(6,"Spitzenlänge",60)
  Defx(7,"Spitzenhöhe",60)
  Defx(8,"Auslaufrichtung der Kragen-Außenkurve",-30)
'--------------------------------- Länge der Halslinie erfragen
  qV = qKop(pXY(0,0)+pXY(100,0))            ' Vorbelegung der qV
  qH = qKop(pXY(0,0)+pXY(50,0))             ' Vorbelegung der qH
  cV = cPick(1,4,"Halslochkurve VORN anpicken !","Kragen",nT)
  qV = qCo(cV,"qq")
  cH = cPick(2,4,"Halslochkurve HINTEN anpicken !","Kragen",nT)
  qH = qCo(cH,"qq")
'---------------------------------------- Ziellänge berechnen
  rZl= rLngQ(qV)+rLngQ(qH)
'---------------------------------------- Aufgabe nicht lösbar
  If(rZl<=rX(3)) Then
    n  = nIBox("Der Kragen ist nicht konstruierbar !")
    Exit Program
  End If
'----------------------------- Anfangseinstellung P0 => P1 (=rA)
  rA = rZl
'------------- 0.Näherung mit Konstruktion der Kragenansatzlinie
  p0 = pXY(0,0)
  p1 = pXY(rA,0)
  p2 = pXY(0,rX(3))
  q1 = qSpline(p2,0,p1,rX(1))
  rL = rLngQ(q1)                      ! Ermittlung der Kurvenlänge
'---------------------- automatische Längeneinstellung im Zyklus
  rA1= 0              ' Funktionswerte für 1.Näherung vorbelegen
  rL1= rX(3)
  rA2= rA
  rL2= rL
  For n = 1,10,1                 ' maximale 10 Näherungsschritte
    rA = rNahInt(rA1,rL1,rA2,rL2,rZl)         ' nächste Näherung
    p1 = pXY(rA,0)
    q1 = qSpline(p2,0,p1,rX(1))
    rL = rLngQ(q1)                  ! Ermittlung der Kurvenlänge
    If(rAbs(rL-rZl)<<0.01) Then         ! Genauigkeit erreicht ?
      Exit For                    ! wenn JA => Schleife verlassen
    End If
    rA1= rA2   ! Funktionswerte-Neubelegung für nächste Näherung
    rL1= rL2
    rA2= rA
    rL2= rL
  End For
```

## 19.1 Themen für Fortgeschrittene

### IF-THEN-Struktur

Die IF-THEN-Struktur ist eine Kontrollstruktur, mit der Berechnungen oder Konstruktionsschritte nur dann ausgeführt werden, wenn eine bestimmt Bedingung erfüllt ist.

Die einfache Struktur ist

```
If (logischer Ausdruck) Then
   [Anweisungen]
End If
```

Nur wenn der logische Ausdruck wahr ist (den Wert True hat), werden die Anweisungen abgearbeitet.
Der logische Ausdruck kann entweder direkt eine logische Variable sein

```
lSchalter=true
If(lSchalter) Then
   [Anweisungen]
End If
```

oder das Ergebnis einer Vergleichsoperation zwischen ganzen / reellen Zahlen oder Variablen.

```
If(rMw<<0) Then
  rSn = rMw
  rHr = 0
  rVr = 0
End If
```

Als Vergleichsoperatoren zwischen ganzzahligen/ reelen Variablen sind zugelassen:

| Zeichen | Bedeutung |
|---|---|
| << | kleiner als |
| >> | größer als |
| == | gleich |
| <= | kleiner gleich |
| >= | größer gleich |
| <> | ungleich |

Zur Verknüpfung logischer Variablen sind zugelassen:

| Zeichen | Bedeutung |
|---|---|
| NOT | „Nicht“ |
| AND | „Und“ |
| OR | „Oder“ |

Die Operationen „==“ und „<>“ sind nur für den Vergleich zwischen ganzen Zahlen geeignet, da bis einschließlich zur 6. Nachkommastelle verglichen wird.

***Beispiel:***
Für extreme individuelle Größen kann der Taillenumfang größer als der Hüftumfang werden. In diesem Fall muß die negative Mehrweite komplett in die Seitennaht gelegt werden. Dieser Fall wird im Programm wie folgt berücksichtigt:

```
'----------------- Mehrweite aufteilen
  rSn = 3/6*rMw      'Anteil in die SN
  rHr = 2/6*rMw      'Anteil in den HR
  rVr = 1/6*rMw      'Anteil in den VR
  If(rMw<<0) Then
    rSn = rMw
    rHr = 0
    rVr = 0
  End If
```

***Innerhalb von IF-THEN-Strukturen dürfen keine Objekte ausgegeben werden, da sich dadurch die Objektanzahl, -art oder –reihenfolge ändern kann. Eine geänderte Objektausgabe kann zu Fehlern bei der Modellentwicklung führen. Es gelten die Bemerkungen zur Verknüpfung Fachsprache ⇔ Protokollbetrieb aus dem letzten Abschnitt des vorhergehenden Kapitels.***

Die ausführliche Struktur ist

```
If (logischer Ausdruck 1) Then
      [Anweisungen 1]
Else If (logischer Ausdruck 2) Then
      [Anweisungen 2]
Else If (logischer Ausdruck 3) Then
      [Anweisungen 3]
Else
      [Anweisungen 4]
End if
```

Eine Erläuterung befindet sich in Bild 19-1. Die „`Else If() Then`“ Abfragen können mehrfach nach „`If() Then`“ folgen. „`Else`“ darf nur einmal vor „`End If`“ stehen.

***Beispiel:***
In einer Rock-Grundkonstruktion für individuelle Größen soll die Mehrweite anders verteilt werden, wenn die Mehrweite für das halbe Erzeugnis größer als 40mm ist. Im Programm ist zu schreiben:

```
'----------------- Mehrweite aufteilen
'                 rSn Anteil in die SN
'                 rHr Anteil in den HR
```

| | |
|---|---|
| **If (**logischer Ausdruck 1**) Then**<br>[Anweisungen 1] | **Wenn (**logischer Ausdruck 1**) Dann**<br>Die Anweisungen 1 werden nur abgearbeitet, wenn “logischer Ausdruck 1” wahr ist. Die IF-ENDIF-Struktur wird verlassen. |
| **Else If (**logischer Ausdruck 2**) Then**<br>[Anweisungen 2] | **Oder Wenn (**logischer Ausdruck 2**) Dann**<br>Die Anweisungen 2 werden nur abgearbeitet, wenn “logischer Ausdruck 2” wahr ist und “logischer Ausdruck 1” falsch war. Die IF-ENDIF-Struktur wird verlassen. |
| **Else**<br>[Anweisungen 3] | **Anderenfalls**<br>Die Anweisungen 3 werden nur abgearbeitet, wenn die vorhergehenden Abfragen falsch waren. |
| **End If** | **Ende Wenn** |

Bild 19-1

```
'                 rVr Anteil in den VR
'------------------------- Fall rMw<0
  If(rMw<<0) Then
    rSn = rMw
    rHr = 0
    rVr = 0
'------------------------ Fall rMw<40
  Else If(rMw<<40) Then
    rSn = 1/6*rMw
    rHr = 3/6*rMw
    rVr = 2/6*rMw
'----------------------- Fall rMw>=40
  Else
    rSn = 1/4*rMw
    rHr = 2/4*rMw
    rVr = 1/4*rMw
  End If
```

### FOR-NEXT-Struktur

Mit der FOR-NEXT-Struktur können Laufschleifen gebildet werden. Die Laufschleife beginnt mit

**For n**Lauf **= n**A,**n**E,**n**Schritt

und endet mit

**End For**

nLauf ist die Laufvariable. Beim ersten Durchlauf hat sie den Wert nA. Nach jedem Durchlauf wird nLauf automatisch um nSchritt erhöht, bzw. reduziert, falls nSchritt negativ ist. Die Anweisungen zwischen For und End For werden bei jedem Durchlauf der Schleife wiederholt bearbeitet.

Die Laufschleife wird erst dann verlassen, wenn die Laufvariable den Endwert überschritten hat oder Exit For angewiesen wurde. Die Variablen nLauf, nA, nE und nSchritt müssen ganzzahlige Variablen sein.

Die gesamte Struktur im Überblick enthält Bild 19-2.

```
For nLauf = nA,nE,nSchritt
      [Anweisungen]
  [Next For]   (nächster Schleifendurchlauf)
      [Anweisungen]
  [Exit For]   (Schleife sofort verlassen)
      [Anweisungen]
End For
```

Bild 19-2

***Beispiel:***

```
'---------X1 bis X5 prüfen, ob negativ
  nA=1
  nE=5
  For nLauf=nA,nE,1
    If(rX(nLauf)<<0) Then
      t1="Der X-Wert X"+tFormat(nLauf)
&       +" ist negativ !"+tc(13,10)+
&        "Der Kragen kann nicht
&         konstruiert werden."
      nBox= nIBox(t1,31)
      Exit Program
    End If
  End For
```

Mit diesen Programmzeilen wird geprüft, ob einer der X-Werte X1 bis X5 negativ ist. Ist einer der Werte negativ, wird das Programm mit einem Hinweis sofort beendet.

### Größeninterpolation

Die Funktion rGroInt() führt eine größenabhängige Interpolation durch. In der vorhergehenden Fachsprache wurden dazu Y-Werten definiert. Eine größenabhängige Interpolation ist dann sinnvoll, wenn sich ein Wert abhängig von der aktuellen Maßtabelle ändern soll. Gleiches wird durch die Definition größenabhängiger X-Werte erreicht. Im Unterschied zu den X-Werten sind die mit rGroInt() berechneten Werte jedoch nur innerhalb des Fachsprachen-Programmes einstellbar. Der Nutzer des freigegebenen Progammes hat keinen Zugriff auf diese Werte. Er kann sie nicht verändern.

Ein größenabhängiger Zahlenwert vom reellen Zahlentyp kann an beliebiger Stelle im Programm mit folgender Befehlszeile definiert werden.

```
  rKorr1=rGroInt("Größe",Wert
&                [,"Größe",Wert,])
```

Als Parameter werden beliebig viele Paare aus Größe und dem zugehörigen Wert übergeben. Die Funktion berechnet den Wert für die aktuelle Maßtabelle aus den Wertepaaren Größe/Wert. Der Größenbezeichner muß mit Anführungsstrichen übergeben werden, wobei ein Unterstrich „_“ die Größe als Standardgröße kennzeichnet. Es ist empfehlenswert, die Größen in aufsteigender Reihenfolge einzugeben. Ist für einen Figurtyp kein Wertepaar angegeben, dann wird für alle Größen dieses Figurtyps der Wert des ersten Wertepaares verwendet. Bitte beachten Sie dazu das folgende Beispiel.

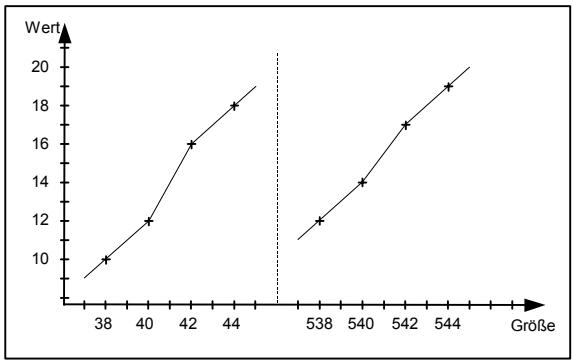


Bild 19-3

Es soll ein Korrekturwert für die Figurtypen normal und starkhüftig gemäß Bild 19-3 definiert werden. Die folgende Befehlszeile belegt den Wert r1 wie gewünscht.

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

Die folgende Übersicht zeigt, welchen Wert die Variable r1 für die angegebene Maßtabelle annimmt.

| Größe | r1 | Größe | r1 | Größe | 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 |

Für den Figurtyp schmalhüftig (vorangestellte „0“) wurde kein Wert definiert. Die Variable erhält daher für alle Größen dieses Figurtyps den Wert 10.

Für eine individuelle Maßtabelle wird der Wert der entsprechenden Verweisgröße (Spalte X-Wert-Verweis in der Gradiertabelle) eingetragen. Ist in dieser Spalte kein Verweis eingetragen, gilt wiederum der Wert des ersten Wertepaares.

### Dialogfunktionen

In Grundkonstruktionen oder Konstruktionsbausteinen werden häufig Informationen aus dem Grafis-Protokoll benötigt. Diese Informationen können Prozeßparameter (Längen, Abstände,...), aber auch Objekte (Punkte, Linien) sein. Beispielsweise werden für den Kragen diverse Informationen vom Halsloch und für einen Ärmel diverse Informationen vom Armloch benötigt. In der neuen Fachsprache kann dazu ein Dialog aufgebaut werden, mit dem der Nutzer des Programmes angewiesen wird, die benötigten Objekte anzuklicken.
Für den Dialog mit dem Nutzer stehen die Funktionen

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

zur Verfügung.

#### *nIBox()*

Infobox baut ein Fenster auf, mit dem der Anwender einen Hinweis erhalten oder das der Anwender mit Ja/Nein schließen kann. In einem Fenster können dem Anwender beispielsweise die Gründe mitgeteilt werden, warum eine Konstruktion unter den konkreten Bedingungen nicht möglich ist. Falls ein X-Wert mit einem extremen Wert belegt ist, kann der Anwender gewarnt werden. ***Die Infobox erscheint auch beim Gradieren! Nutzen Sie `nIBox` in der Regel für Fehlermeldungen aus Ihrem Programm bei extremen Konstruktionsvorgaben.***
Die Infobox kann in verschiedenen Darstellungsvarianten erscheinen. Die Darstellungsvariante wird mit dem optionalen ganzzahligen Parameter nD festgelegt. Die Zehnerposition dieses Parameters steuert, welche Buttons angezeigt werden. Die Einerposition legt das Symbol im Fenster fest. Bild 19-4 enthält die Darstellungsvarianten im Überblick.
Der Funktionswert von `nIBox` ist eine Zahl, die den angeklickten Button kennzeichnet. Die Werte von `nIBox` stehen für folgende Button:

1 - Button „OK”
2 - Button „Ja”
3 - Button „Nein“
4 - Button „Ignorieren”
5 - Button „Wiederholen“
6 - Button „Beenden“

Anklicken des Button „Abbrechen“ führt immer zum sofortigen Beenden des Programmes. Es ist gleichbedeutend mit `Exit Program`.

***Beispiel:***
Bevor in der Fachsprache die Konstruktionsschritte für eine Brief-/Kuvertecke ausgeführt werden, wird getestet, ob die angewiesenen Linien zum Bilden einer Ecke geeignet und nicht parallel sind.

```
'----------------- Strecken parallel ?
rWi = rWiSS(s1,s2)
t1  = "Die Linien der Ecke sind fast"
& " parallel (Winkel<5°) !"+tC(13,10)
& +"Die Kuvertecke kann nicht"
& " konstruiert werden."
t2  = "Error – Konstruktion
& Kuvertecke"
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
```

Falls der Winkel zwischen den Strecken `s1` und `s2` fast 0° oder fast 180° ist (zwischen –5°/5°, 175°/180° oder –175°/-180°), dann erscheint die Meldung gemäß Bild 19-5 und das Programm wird abgebrochen.

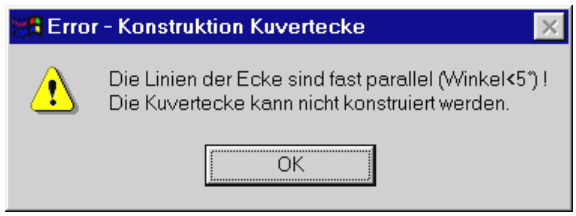


Bild 19-5

Die Parameter beim Aufruf von

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

| | Zeichen/ Symbol: | ohne Symbol | Ausrufe-zeichen | Info-Zeichen | Frage-zeichen | Kreuz als Warnzeichen |
|---|---|---|---|---|---|---|
| **Tasten:** | | **+0** | **+1** | **+2** | **+3** | **+4** |
| **Beenden/Wieder-holen/Ignorieren** | **10+** | 10 | 11 | 12 | 13 | 14 |
| **OK** | **20+** | 20 | 21 | 22 | 23 | 24 |
| **OK/Abbrechen** | **30+** | 30 | 31 | 32 | 33 | 34 |
| **Wiederholen/Abbrechen** | **40+** | 40 | 41 | 42 | 43 | 44 |
| **Ja/Nein** | **50+** | 50 | 51 | 52 | 53 | 54 |
| **Ja/Nein/Abbrechen** | **60+** | 60 | 61 | 62 | 63 | 64 |

Bild 19-4

sind:

- t1 der Text der Mitteilung. Hier: „Die Linien der Ecke sind fast parallel ...“
- t2 der Titel der Infobox; hier: „Error – Konstruktion Kuvertecke“
- 21 die Nummer für die Darstellungsvariante; hier: 21 für das Symbol Ausrufezeichen und den Button „OK“.

***cPick()***

Mit cPick() erscheint analog zu nIBox eine Mitteilung auf dem Bildschirm, mit der der Anwender aufgefordert wird, einen Punkt bzw. eine Linie anzuklicken. Alle Informationen zum angeklickten Objekt werden zunächst in einen Container gepackt und können bei Bedarf dem Container entnommen werden.

Für Kragen- oder Ärmelprogramme können mit cPick() Dialoge erarbeitet werden, in denen der Anwender gezielt die erforderlichen Linien anklickt. Die Verwendung von Z-Werten zur Übergabe von Längen und Abständen an das Fachsprachenprogramm ist nicht mehr erforderlich.

***Beispiel:***

Für die Ermittlung der Gesamtlänge der Halslochlinien von Vorder- und Rückenteil sind folgende Programmzeilen einzugeben:

```
'-------------------- qV, qH vorbelegen
  qV = qKop(pXY(0,0)+pXY(100,0))
  qH = qKop(pXY(0,0)+pXY(50,0))
'---------------------- qV, qH anpicken
  cV = cPick(1,4,"Halslochkurve VORN"+
& " anpicken !","Kragen",nT)
  qV = qCo(cV,"qq")
  cH = cPick(2,4,"Halslochkurve HINTEN"
& +" anpicken !","Kragen",nT)
  qH = qCo(cH,"qq")
'------------------ Ziellänge berechnen
  rZl= rLngQ(qV)+rLngQ(qH)
```

Mit dem Aufruf von cPick wird zunächst ein Container gefüllt, hier: cV und cH. Aus diesen Containern werden anschließend mit der Funktion qCo die benötigten Informationen entnommen.

Die Parameter der Funktion cPick() haben folgende Bedeutung:

cPick(nI,nV,tK,tT,tB,nT)

- nI eineindeutiger Identifikator; Dieser Identifikator muß für jeden Aufruf von cPick() anders sein. Die verschiedenen Pick-Anweisungen werden damit bei Probelauf und Gradieren identifiziert.
- nV Anpickvariante
  - 1 Punkt anpicken
  - 2 Linien-/Kurven-Stützpunkt anpicken
  - 4 Linien/Kurven anpicken
  - 8 Picken im „freien“ erlaubt

  Die einzelnen Varianten können durch Addieren kombiniert werden. So bedeutet nV=3 das Anpicken von Punkten und Linien-/Kurven-Stützpunkten.
- tK Kommentartext; Mit diesem Text wird der Anwender zum Picken aufgefordert.
- tT Titeltext für die Dialogbox
- tB Bild-/Symboltext; Erlaubt sind , “!“, “+“, “i“, “g“ oder der komplette Pfad zu einer Bitmap.
- nT Bei Aufruf: Nummer des Teiles, in dem gepickt werden darf. Der Fall nT=0 erlaubt das Picken in allen Teilen, deren Teilenummer <= der des aktiven Teiles ist.
  Nach dem Aufruf: Nummer des Teiles in dem gepickt wurde.

Folgende Informationen können mit folgenden Funktionen aus dem Pick-Container abgerufen werden. Der erste Parameter ist jeweils der Variablenname des Pick-Containers.

| | |
|---|---|
| nCo(c,"tl") | Nummer des Teiles, in dem gepickt wurde |
| nCo(c,"nr") | Pos-Nummer des angepickten Objektes |
| nCo(c,"ty") | Typ des angepickten Objektes (-1: Fehler, 0- Digi-Punkt, 1-Punkt, 2- Linie/Kurve) |
| lCo(c,"rl") | Linie/Kurve wurde Rechts angepickt (JA/NEIN) |
| lCo(c,"st") | Pickpunkt ist Linien-/ Kurvenstützpunkt (JA/NEIN) |
| lCo(c,"ri") | Kurvenrichtung im Pickpunkt in Grad |
| rCo(c,"rln") | relative Länge der Kurve im Pickpunkt in % |
| pCo(c,"pp") | Pickpunkt |
| qCo(c,"qq") | angepickte Linie/Kurve |

**Zugabenklassen**

Die Mehrweite einer Grundkonstruktion kann entweder über Zugabenklassen oder mit X-Werten eingestellt werden. Welche der beiden Varianten angewendet wird oder ob beide Varianten gemischt werden, entscheidet der Programmierer.

Werden Zugabenklassen verwendet, dann legt der Programmierer die Mehrweite je Zugabenklasse in der Taille, in der Hüfte, im Brustumfang, in der Armlochvertiefung,... im Programm fest. Bleiben die Zugabenklassen unberücksichtigt, stellt der Anwender die Mehrweiten später mit X-Werten ein.

Die jeweils gültige Zugabenklasse wird in der Gradiertabelle festgelegt und mit der Funktion nZKlasse() als ganzzahliger Wert übergeben.

In einem Fachsprachenprogramm mit der Befehlszeile nZkl=nZKlasse() hat die Variable nZkl beim Abarbeiten mit der jeweiligen Maßtabelle den folgenden Wert:

| Gradiertabelle | Wert von 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 |

Der Programmierer legt mit der Verrechnung von `nZKlasse` fest, wieviel Mehrweite im Brustumfang, Taillenumfang... berücksichtigt wird. Mit den Programmzeilen

```
'------------------------- Brustweite
  rBu = rG(1)+10*nZKlasse()
'------------------------ Taillenweite
  rTa = rG(4)+13*nZKlasse()
'------------------------- Gesäßweite
  rGe = rG(2)+15*nZKlasse()
'------------------- Armlochvertiefung
  rAt = 2*nZKlasse()
```

berechnet sich die Brustweite rBu für das halbe Erzeugnis wie folgt

| Gradiertabelle | | | Wert von rBu | Zugabe |
|---|---|---|---|---|
| > 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 | e02 | ____40_0 | 920+10*2 | +20 |
| > 05 | e06 | ____40_0 | 920+10*6 | +60 |
| > 06 | e08 | ____40_0 | 920+10*8 | +80 |

Je Zugabenklasse werden 10mm Mehrweite im Brustumfang, 13mm Mehrweite im Taillenumfang, 15mm Mehrweite im Gesäßumfang verrechnet. Gleichzeitig wird das Armloch um je 2mm vertieft. Mit den Faktoren vor `nZKlasse()` legt der Programmierer die Mehrweite je Zugabenklasse fest. Soll sich die Mehrweite je Zugabenklasse ungleichmäßig ändern, muß die IF-ENDIF-Struktur angewendet werden.

Für körpernahe Bekleidung, z.B. Bademoden oder Sportbekleidung, kann auch `10*nZKlasse()-40` gerechnet werden. Damit ist das Maß in Zugabenklasse a00 um 4cm kleiner als das Körpermaß.

### Kreisfunktionen

Kreise werden in der neuen Fachsprache wie Kurven behandelt. Alle Kurven-Funktionen können auch auf Kreise angewendet werden. Ein Kreis wird mit den Funktionen

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

gebildet. Als Parameter sind immer der Mittelpunkt des Kreises (Punktvariable) und der Radius des Kreises (reele Variable) anzugeben. Je nach Kreistyp folgen Richtungsangaben.

Sofern der Kreis zur Schnittpunktbildung mit einer Strecke oder einer anderen Kurve benötigt wird, sollte grundsätzlich `qVollKr()` verwendet werden. Zur Schnittpunktbildung muß ein Richtpunkt angegeben werden, der bei mehreren möglichen Schnittpunkten auf den gesuchten zeigt. Der Kreismittelpunkt ist als Richtpunkt ungeeignet.

***Beispiel:***

Im folgenden Beispielprogramm wird jede Kreisvariante definiert und danach ausgegeben (Bild 19-6).

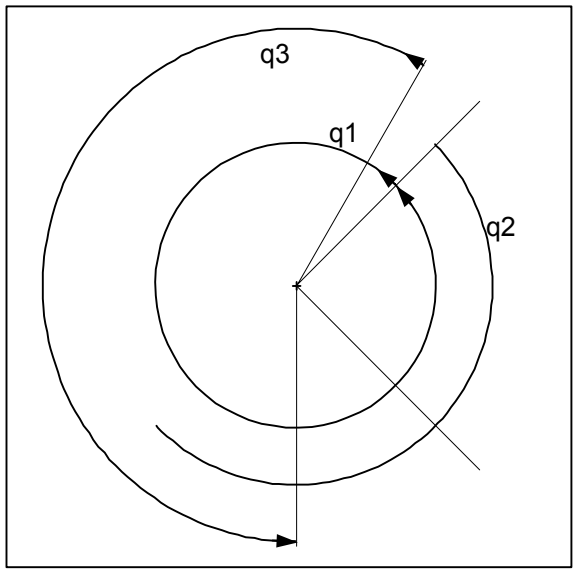


Bild 19-6

Der Parameter 45 in der Zeile

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

bestimmt, daß die Symmetrieachse des Vollkreises (Anfang und Ende) die 45° Richtung ist. Der Parameter –45 in der Zeile

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

bestimmt, daß die Symmetrieachse des Halbkreises die –45° Richtung ist. Die Parameter 60 und –90 in der Zeile

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

bestimmen, daß der Teilkreis bei 60° beginnt und bei –90° endet. Der Kreis wird in mathematisch positivem Drehsinn erzeugt.

```
'************************************
  Program Main()      ' Kreisvarianten
'------------------------------------
  rVar rRad1,rRad2,rRad3
  qVar q1,q2,q3
  pVar p0
'------------------------ Mittelpunkt
  p0  = pXY(0,0)
'---------------------------- Radien
  rRad1  = 100
  rRad2  = 140
  rRad3  = 180
'------------------- Kreise definieren
  q1  = qVollKr(p0,rRad1,45)
  q2  = qHalbKr(p0,rRad2,-45)
  q3  = qTeilKr(p0,rRad3,60,-90)
'--------------------------- Ausgaben
  AusP(p0)
  AusQ(q1,q2,q3)
  End Program
'************************************
```

### Externe Funktionen

Zur besseren Übersicht können häufig benötigte Programmschritte als externe Funktion abgelegt werden. Eine externe Funktion (hier: vom Typ n) beginnt mit

**Function n**Xxx**(**[*Parameterliste*]**)**
und endet mit
**End Function**
Der Funktionsname ist analog einer Variablenbezeichnung zu bilden und auch einem Variablentyp zuzuordnen. Innerhalb der Funktion ist der Funktionsname eine Variable, die mit einem Wert belegt werden kann. Dieser Wert wird nach dem Abarbeiten der Funktion zurückgegeben. Beim Funktionsaufruf können in der Parameterliste beliebig viele Variablen unterschiedlichen Typs an die Funktion übergeben werden. Die Parameteranzahl und die Parametertypen müssen im Funktionsaufruf und in der Funktionsdefinition identisch sein. Die Parameter im Funktionsaufruf werden auch zurückgegeben. Externe Funktionen können in eigenen Modulen stehen. Zukünftig werden spezielle Module für Kurven, Ecken und anderes entwickelt werden, die als Bibliotheken auch anderen Grafis-Programmierern zur Verfügung stehen. Die Strukturen `Program Main()/ End Program` und `Function xXxx()/ End Function` dürfen nicht miteinander oder untereinander verschachtelt werden.

***Beispiel:***

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

'************************************
  Function lEck(p1,p2,p3,p4,pEck)
'  Berechnung des Schnittpunktes
'  zweier Strecken
'  Die Punkte p1 und p2 bilden die
'  erste Strecke,
'  die Punkte p3 und p4 bilden die
'  zweite Strecke.
'  Der Eckpunkt wird als fünfter
'  Parameter zurückgegeben.
'  Falls beide Strecken fast parallel
'  sind wird mit einer Warnung
'  abgebrochen.
'  Erstellt: 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)
'---------------- Strecken parallel ?
  rWi = rWiSS(s1,s2)
t1  = "Die Linien der Ecke sind fast"
& +" parallel (Winkel<5°) !"+tC(13,10)
& +"Die Ecke kann nicht konstruiert"
& +" werden."
t2  = "Error – Konstruktion Ecke"
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
  pEck= pSchnSS(s1,s2)
  lEck= True
  End Function
'************************************
```

Die externe Funktion `lEck()` konstruiert einen Eckpunkt, der durch vier Punkte bestimmt wird. In der Parameterliste werden die vier Punkte übergeben. Der erste und zweite sowie der dritte und vierte Punkt bilden je eine Strecke. Ist der Winkel zwischen beiden Strecken kleiner als 5°, dann wird das Programm mit einer Mitteilung abgebrochen. Anderenfalls wird der Eckpunkt `pEck` berechnet und als fünfter Parameter der Parameterliste zurückgegeben. Die Funktion kann mehrfach (hier: zweimal) jeweils mit anderen Punkten aufgerufen werden.

Häufig werden logische Funktionen programmiert, die nur mit dem Wert `True` abschließen, wenn die Funktion korrekt abgearbeitet werden konnte. Für eine Verschiebung der Schulternaht um X6 am Halsloch und um X7 am Armloch müßte die Funktion `lTranslSchulter()` folgende Struktur haben:

```
'************************************
  Program Main()
  [Anweisungen]
  lIst=lTranslSchulter(qHalsV,qHalsH,
&            qArmV,qArmH,rX(6),rX(7))
  [Anweisungen]
  End Program
'************************************
'************************************
  Function lTranslSchulter(q1,q2,
&            q3,q4,r1,r2)
'  Transformation von q2 an q2 und
'  q4 an q3
'  Endpunkt von q1 um r1 verschieben
'  Endpunkt von q3 um r2 verschieben
'  Kurven neu bilden
'  neue q2 und neue q4 rücktransform.
'************************************
  lTranslSchulter=False
  [Anweisungen]
  q1=...
  q2=...
  q3=...
  q4=...
  lTranslSchulter=True
  End Function
'************************************
```

Die neuen Kurven von Hals- und Armloch werden in der Parameterliste übergeben. Vor dem Abarbeiten der Zeile

```
  lIst=lTranslSchulter(qHalsV,qHalsH,
&            qArmV,qArmH,rX(6),rX(7))
```

in Main() ist die Schulternaht der Kurven `qHalsV`, `qHalsH`, `qArmV`, `qArmH` noch nicht verlegt. Nach Abarbeiten der Zeile sind die Kurven neu belegt, die Schulter ist verschoben.

## 19.2 Automatische Längenanpassung

**rNahInt(r**A1,**r**Ist1,**r**A2,**r**Ist2,**r**Ziel**)**

Die Näherungsinterpolation rNahInt() wird für automatische Längenanpassungen, beispielsweise von Kragen und Ärmeln benötigt. Die Kragen sind an das Halsloch und die Ärmel an das Armloch anzupassen.

Für die Umsetzung von Längenanpassungen gilt prinzipiell die folgende Vorgehensweise:

1. Formulieren einer eindeutigen Konstruktionsbeschreibung mit einer Ziellänge, die von einem anderen Konstruktionsparameter abhängig ist.
2. Festlegen eines variablen Konstruktionsparameters rA, der zum Einstellen der Ziellänge rZiel verändert werden kann.
3. Vorbelegen einer nullten Näherung. Beispiel: Für rA1=0 würde sich rIst1=0 ergeben.
4. Erste Entwicklung der Konstruktion mit einem geeigneten Anfangswert rA2 des variablen Konstruktionsparameter bis zur Istlänge rIst2 und Berechnung der Istlänge rIst2. Je nach Umfang können diese Entwicklungsschritte als Externe Funktion programmiert werden.
5. Berechnen eines neuen Anfangswertes rA mit der Funktion rNahInt().
6. Wiederholte Entwicklung der Konstruktion mit dem neuen Anfangswert rA und Berechnung der Istlänge rIst.
7. Abfrage, ob die Ziellänge bereits erreicht wurde. Wenn ja, wird gemäß Punkt 8. die Konstruktion fortgesetzt.
   Wenn nein, werden die Werte von rA2 und rIst2 auf die Variablen rA1 und rIst1 umgespeichert. Die Variablen rA2 und rIst2 erhalten danach die Werte von rA und rIst, die zuletzt ermittelt wurden. Danach wird gemäß Punkt 5. ein neuer Wert für rA berechnet.
8. Fertigstellen der Konstruktion.

**Die Funktion**
**rNahInt(r**A1,**r**Ist1,**r**A2,**r**Ist2,**r**Ziel**)**

Die Funktion rNahInt(r**A1**,r**Ist1**,r**A2**,r**Ist2**,r**Ziel**) berechnet aus zwei Anfangswerten rA1 und rA2 und den zugehörigen Ergebniswerten rIst1 und rIst2 den vermutlichen Anfangswert, der als Ergebnis rZiel liefert.

***Es wird dabei von einer weitestgehend linearen Abhängigkeit zwischen Anfangs- und Ergebniswert ausgegangen (Bild 19-7).***

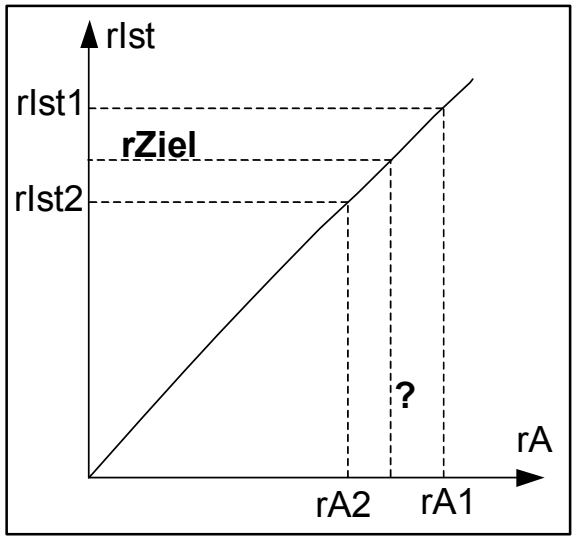


Bild 19-7

Bei der folgenden Bündchenkragenkonstruktion ist der variable Konstruktionsparameter der Abstand von p1 zu p3.

Beispiel:
Für einen Abstand p1⇔p3 von 160mm wird die Ansatzlinie 165.8mm lang und für 200mm wird sie 204.7mm lang. Mit der Belegung

```
rA1  =160
rIst1=165.8
rA2  =200
rIst2=204.7
```

kann der benötigte Abstand p1⇔p3 für eine Ziellänge der Ansatzlinie von 183mm wie folgt berechnet werden:

```
rZiel=183
rA   =rNahInt(rA1,rIst1,rA2,
&             rIst2,rZiel)
```

Diese Berechnung liefert einen Wert von 177.6mm für den Abstand p1⇔p3. Mit diesem Wert entsteht eine Ansatzlinie der Länge 182.9mm. Eine nochmalige Berechnung mit den Werten

```
rA1  =200
rIst1=204.7
rA2  =177.6
rIst2=182.9
```

liefert einen Wert von 177.7mm für den Abstand p1⇔p3. Mit diesem Wert hat die Ansatzlinie die vorgegebene Länge von 183.0mm.

**Konstruktion: Bündchenkragen mit automatischer Längenanpassung**

Ein Bündchenkragen gemäß Bild 19-8 soll unter Verwendung folgender X-Werte programmiert werden:

| X | Bezeichnung | Schritt | Wert |
|---|---|---|---|
| 1 | Zugabe Kragenlänge | | 0mm |
| 2 | Höherstellung HM | p1⇒p2 | 35mm |
| 3 | Kragenumfaltbreite | p2⇒p4 | 20mm |
| 4 | Kragenbreite HM | p4⇒p5 | 40mm |
| 5 | Kragenspitze (X) zu p3 | p3⇒p6 | 40mm |
| 6 | Kragenspitze (Y) zu p3 | p6⇒p7 | 45mm |
| 7 | Faktor für den Richtpkt Ansatzlinie bezogen auf X2 | p1⇒p9 | 2.6 |
| 8 | Richtpunkt für Außenlinie | p1⇒p8 | 155mm |

***Konstruktionsschritte:***

| von | bis | Richtg | Abstand |
|---|---|---|---|
| 1 | 2 | ⇑ | X2 (Höherstellung HM) |
| 2 | 4 | ⇑ | X3 (Kragenumfaltbreite) |
| 4 | 5 | ⇑ | X4 (Kragenbreite) |
| 1 | 8 | ⇑ | X8 (Richtpunkt Außenlinie) |
| 1 | 9 | ⇑ | X2*X7 |
| 1 | 3 | ⇒ | **Variabler Abstand**, so daß Ansatzlinie=Halslochmaß+X1 |
| | | | Ansatzlinie konstruieren und optimieren |
| 3 | 6 | ⇒ | X5 |
| 6 | 7 | ⇑ | X6 |
| | | | Umfalt- und Außenlinie konstruieren |

Alle Kurven sollen rechtwinklig in die Hintere Mitte einlaufen. Die Ansatz- und Umfaltlinie haben in p3 die Richtung p3⇒p9. Die Außenlinie hat in p7 die Richtung p7⇒p8.

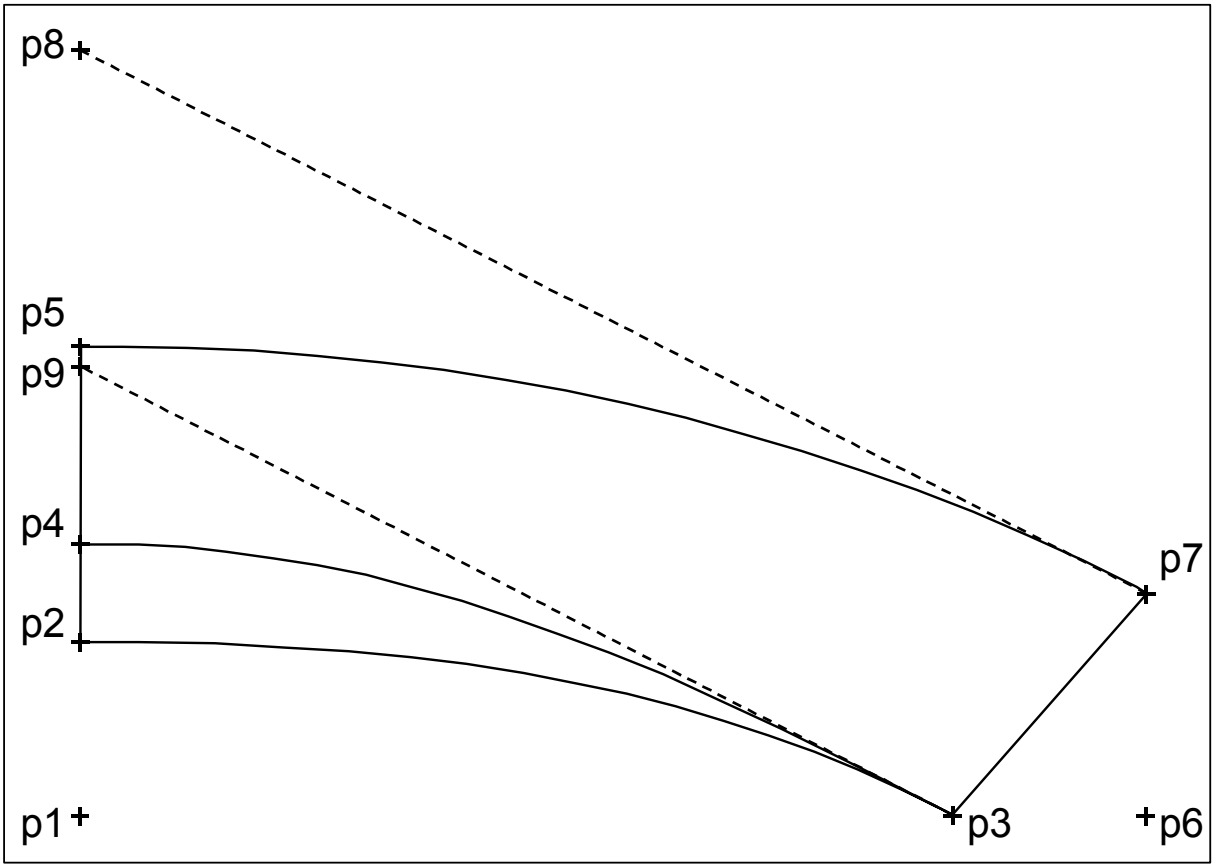


Bild 19-8

**Das Programm: Bündchenkragen mit automatischer Längenanpassung**

```
'************************************************************************
  Program Main()
' Konstruktion eines Bündchenkragens nach einer Vorlage
' von Frau Prof. H.Brückner, Berlin
' Der Anwender muß die Halslochlinien von Vorder- und Rückenteil
' anklicken. Der Kragen wird automatisch so konstruiert, daß die Länge
' der Kragenansatzlinie gleich der Länge beider Halslochlinien
' plus Zugabe X1 ist.
'------------------------------------------------------------------------
  lVar
  nVar n
  rVar rWi3,rWi7,rA,rA1,rA2,rIst,rIst1,rIst2,rKgLng,rZiel
  pVar p1,p2,p3,p4,p5,p6,p7,p8,p9
  sVar
  qVar q1,q2,q3,qV,qH
  tVar
  cVar cV,cH
'------------------------------------------------------------------------
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  rCon
  tCon
'---------------------------------------------------- X-Wert-Definitionen
  XTitel("Bündchenkragen")
  Defx(1,"Zugabe Kragenlänge",0)
  Defx(2,"Höherstellung HM",35)
  Defx(3,"Kragenumfaltbreite",20)
  Defx(4,"Kragenbreite HM",40)
  Defx(5,"Kragenspitze (X) in Bezug zu p3",40)
  Defx(6,"Kragenspitze (Y) in Bezug zu p3",45)
  Defx(7,"Faktor für Richtpunkt Ansatzl. bezogen auf X2",2.6)
  Defx(8,"Richtpunkt für Außenlinie",155)
'----------------------------------- Länge des Halslochlinien erfragen
  cV  = cPick(1,4,"VORDERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cV, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qV  = qCo(cV,"qq")
  cH  = cPick(2,4,"HINTERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cH, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qH  = qCo(cH,"qq")
  rKgLng = rLngQ(qV)+rLngQ(qH)
  If(rKgLng<<rX(2)) Then
    n   = nIBox("Die Halslochlinien sind zu kurz !")
    Exit Program
  End If
'------------------------------------------------------------ Punkte der HM
  p1  = pXY(0,0)
  p2  = pXY(0,rX(2))
  p4  = pPRiLng(p2,rOb,rX(3))
  p5  = pPRiLng(p4,rOb,rX(4))
  p8  = pPRiLng(p1,rOb,rX(8))
  p9  = pPRiLng(p1,rOb,rX(2)*rX(7))
'------------------------------------------- Kragenansatzlinie 0.Näh.
  rZiel  = rKgLng+rX(1)
  rA  = rKgLng
  p3  = pXY(rA,0)
  rWi3= rRiPP(p3,p9)
  q1  = qSpline(p3,rWi3,p2,rLi)
  rIst= rLngQ(q1)
```

```
'------------------------------------------------ Werte für 1.Näherung
  rA1 = 0
  rIst1  = 0
  rA2 = rA
  rIst2  = rIst
'-------------------------------------------------------- Atom.Näherung
  For n = 1,10,1
    rA  = rNahInt(rA1,rIst1,rA2,rIst2,rZiel)
    p3  = pXY(rA,0)
    rWi3= rRiPP(p3,p9)
    q1  = qSpline(p3,rWi3,p2,rLi)
    rIst= rLngQ(q1)
    If(rAbs(rIst-rZiel)<<0.01) Then
      Exit For
    End If
    rA1 = rA2
    rIst1  = rIst2
    rA2 = rA
    rIst2  = rIst
  End For
'------------------------------------------------------------ Kragenspitze
  p6  = pPRiLng(p3,rRe,rX(5))
  p7  = pPRiLng(p6,rOb,rX(6))
'---------------------------------------------------- Kragenumfaltlinie
  q2  = qSpline(p3,rWi3,p4,rLi)
'---------------------------------------------------- Kragenaußenlinie
  rWi7= rRiPP(p7,p8)
  q3  = qSpline(p7,rWi7,p5,rLi)
'------------------------------------------------ Punkte+Linien ausgeben
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(p2+p5)
  AusQ(p3+p7)
  AusQ(q1,q2,q3)
'-----------------------------------------------------------------------
  End Program
'***********************************************************************
```

## 19.3 Ansatzlinie mit Minimum als Externe Funktion

Als Basis für diverse Kragenentwicklungen soll eine Externe Funktion `qKgAnsatz()` programmiert werden, die eine Ansatzlinie (Bild 19-9) mit vorgegebener Länge unter Berücksichtigung folgender Parameter liefert:

- Höherstellung Hintere Mitte
- Höherstellung Vordere Mitte
- zusätzliche Richtung an der VM
- Lage des Minimums in Prozent (ab HM)

Auf dieser Seite befindet sich zunächst eine Testumgebung für die Funktion `qKgAnsatz()`. Auf Seite 12 folgt dann die Funktion `qKgAnsatz()` selbst. Die Funktion `qKgAnsatz()` kann auch in einem neuen Modul mit weiteren Funktionen für Ansatzlinien anderer Form gespeichert werden.

***Konstruktionsschritte Funktion `qKgAnsatz()`:***

| von | bis | Richtg | Abstand |
|---|---|---|---|
| 1 | 2 | ⇑ | rHm (Höherstellung HM) |
| 1 | 4 | ⇒ | **Variabler Abstand**, Dieser Abstand wird so optimiert, daß die Ansatzlinie gleich rZiel lang ist. |
| 1 | 3 | ⇒ | rMin /100* Variabler Abstand |
| 4 | 5 | ⇑ | rVm (Höherstellung VM) |
| | | | Ansatzlinie konstruieren und optimieren |

Die Ansatzlinie soll rechtwinklig in die Hintere Mitte einlaufen. Im Punkt p5 soll die Kurve die Richtung p3⇒p5 plus der Korrektur rRi5z haben.

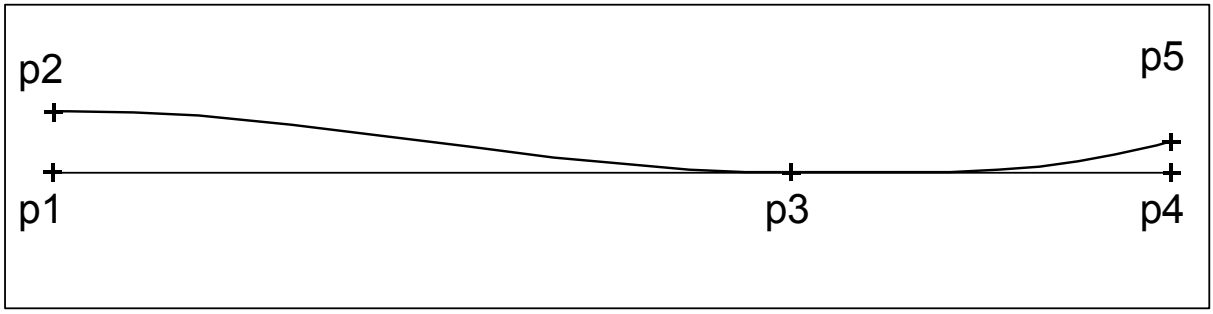


Bild 19-9

```
'**********************************************************************
  Program Main()
' Testumgebung für die Entwicklung der Funktion qKgAnsatz()
'----------------------------------------------------------------------
  nVar n
  rVar rKgLng,rZiel
  pVar p1,p2,p3,p4,p5
  qVar qV,qH,q1
  rCon rRe=0,rLi=180,rOb=90,rUn=270
'------------------------------------------------------------- X-Werte
  XTitel("Ansatzlinie für Hemdkragen")
  Defx(1,"Zugabe zur Kragenansatzlinie",0)
  Defx(2,"Höherstellung HM",10)
  Defx(3,"Höherstellung VM",5)
  Defx(4,"Zusätzl. Richtung Stegansatz in p5",10)
  Defx(5,"Lage p3 zwischen p1-p4 in %",66)
'----------------------------------- Länge des Halslochlinien erfragen
  cV  = cPick(1,4,"VORDERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cV, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qV  = qCo(cV,"qq")
  cH  = cPick(2,4,"HINTERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cH, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qH  = qCo(cH,"qq")
  rKgLng  = rLngQ(qV)+rLngQ(qH)
  If(rKgLng<<rX(2)) Then
    n    = nIBox("Die Halslochlinien sind zu kurz !")
    Exit Program
  End If
  rZiel= rKgLng+rX(1)
'------------------------------------------------- Kragenansatzlinie
  q1   = qKgAnsatz(rX(2),rX(3),rX(4),rX(5),rZiel,p1,p2,p3,p4,p5)
'------------------------------------------------------------ Ausgaben
  AusP(p1,p2,p3,p4,p5)
  AusQ(q1)
  End Program
'**********************************************************************
```

```
'************************************************************************
  Function qKgAnsatz(rHm,rVm,rRi5z,rMin,rZiel,p1,p2,p3,p4,p5)
' ... ermittelt die Kragenansatzlinie, die ein Minimum bei rRi5 %
' zwischen HM und VM hat. Die Übergabe-Parameter:
'   rHm  - Höherstellung HM
'   rVm  - Höherstellung VM
'   rRi5 - zusätzliche Richtung in p5 (an der VM)
'   rMin - Lage des Minimums in Prozent (ab HM)
'   rZiel- Ziellänge der Ansatzlinie (incl.Zugabe) von HM bis VM
' Zurückgegeben wird die Kurve qKgAnsatz, die in der HM beginnt,
' und die Punkte p1 bis p5.
'------------------------------------------------------------------------
  nVar n
  rVar rA,rRi5,rIst,rA1,rIst1,rA2,rIst2
'------------------------------------------------------------------------
  rCon rRe=0,rLi=180,rOb=90,rUn=270
'------------------------------------------------------- Punkte der HM
  p1   = pXY(0,0)
  p2   = pPRiLng(p1,rOb,rHm)
'------------------------------------------------------------ 0.Näherung
  rA   = rZiel
  p3   = pPRiLng(p1,rRe,rMin/100*rA)
  p4   = pPRiLng(p1,rRe,rA)
  p5   = pPRiLng(p4,rOb,rVm)
  rRi5 = rRiPP(p3,p5)+rRi5z
  qKgAnsatz  = qSpline(p2,rRe,p3,rRe,p5,rRi5)
  rIst = rLngQ(qKgAnsatz)
'------------------------------------------------- Werte für 1.Näherung
  rA1  = 0
  rIst1= 0
  rA2  = rA
  rIst2= rIst
'------------------------------------------------------- Atom.Näherung
  For n = 1,10,1
    rA   = rNahInt(rA1,rIst1,rA2,rIst2,rZiel)
    p3   = pPRiLng(p1,rRe,rMin/100*rA)
    p4   = pPRiLng(p1,rRe,rA)
    p5   = pPRiLng(p4,rOb,rVm)
    rRi5 = rRiPP(p3,p5)+rRi5z
    qKgAnsatz  = qSpline(p2,rRe,p3,rRe,p5,rRi5)
    rIst = rLngQ(qKgAnsatz)
    If(rAbs(rIst-rZiel)<<0.01) Then
      Exit For
    End If
    rA1  = rA2
    rIst1= rIst2
    rA2  = rA
    rIst2= rIst
  End For
  End Function
'************************************************************************
```

## 19.4 Hemdkragenkonstruktion unter Nutzung der Externen Funktion qKgAnsatz()

Unter Nutzung der Externen Funktion `qKgAnsatz()` des vorhergehenden Abschnittes soll ein Hemdkragen (Bild 19-10) mit folgenden X-Werten konstruiert werden: werden.

| X | Bezeichnung | Schritt | Wert |
|---|---|---|---|
| 1 | Zugabe Kragenlänge | | 0mm |
| 2 | Höherstellung HM | p1⇒p2 | 10mm |
| 3 | Höherstellung VM | p4⇒p5 | 5mm |
| 4 | Zusätzl. Richtung Stegansatz in p5 | | 10° |
| 5 | Stegbreite HM | p2⇒p6 | 25mm |
| 6 | Kragenbreite HM | p6⇒p7 | 65mm |
| 7 | Übertrittbreite Steg | p5⇒p8 | 20mm |
| 8 | Minderung Stegbreite am Übertritt | p8⇒p10 | 5mm |
| 9 | Spitzenübertritt in X | p4⇒p11 | 15mm |
| 10 | Spitzenübertritt in Y | p4⇒p11 | 10mm |
| 11 | Richtung Kragenspitze | | 10° |
| 12 | Lage p3 zwischen p1-p4 | p1⇒p4 | 66% |

***Konstruktionsschritte:***

| von | bis | Richtg | Abstand |
|---|---|---|---|
| 1 | 2 | ⇑ | X2 (Höherstellung HM) |
| 1 | 4 | ⇒ | **Variabler Abstand**, so daß Ansatzlinie=Halslochmaß+X1 |
| 1 | 3 | ⇒ | X12 /100* Variabler Abstand |
| 4 | 5 | ⇑ | X3 (Höherstellung VM) |
| | | | Ansatzlinie konstruieren und optimieren |

Die bisherigen Schritte werden von der Externen Funktion `qKgAnsatz()` abgearbeitet. Alle folgenden Schritte werden im Programm Main() programmiert.

| | | | |
|---|---|---|---|
| 2 | 6 | ⇑ | X5 |
| | | | Parallele zur Ansatzlinie im Abstand X5 |
| | 9 | | Endpunkt der Parallele |
| 6 | 7 | ⇑ | X6 (Kragenbreite) |
| 5 | 8 | Ansatzl. in p5 | X7 (Übertrittbreite Steg) |
| 8 | 10 | senkr. wie | X5-X8 (Minderung Stegbreite am Übertritt) |

| | | vorher | |
|---|---|---|---|
| 4 | 11 | ⇑ | X2+X5+X6+X10 (Spitzenübertritt in Y) |
| 11 | 11 | ⇒ | X9 (Spitzenübertritt in X) |
| | | | Außenlinie konstruieren mit Richtung X11 in p11 |

Alle Kurven sollen rechtwinklig in die Hintere Mitte einlaufen.

***Schrittfolge:***

Erstellen Sie ein neues Projekt „Hemdkragen“ und eröffnen in diesem Projekt über *Modul | Neu....* das neue Modul „Ansatzlinien“. Das Modul erscheint in der Variablen Liste unter der Rubrik „Module“. Durch Anklicken wird es geöffnet. Kopieren Sie die getestete Funktion `qKgAnsatz()` des vorhergehenden Abschnittes in das Modul „Ansatzlinien“ und compilieren das neue Modul. Anschließend wählen Sie aus der Rubrik „Module“ durch Anklicken von Main.qpr wieder das Hauptmodul und entwickeln den Hemdkragen, analog den Programmzeilen auf Seite 14.

Für die Entwicklung eines anderen Kragens mit Ansatzlinie gleicher Form muß nur das Modul „Ansatzlinien“ eingefügt werden. Damit kann auch dort die Funktion `qKgAnsatz()` genutzt werden.

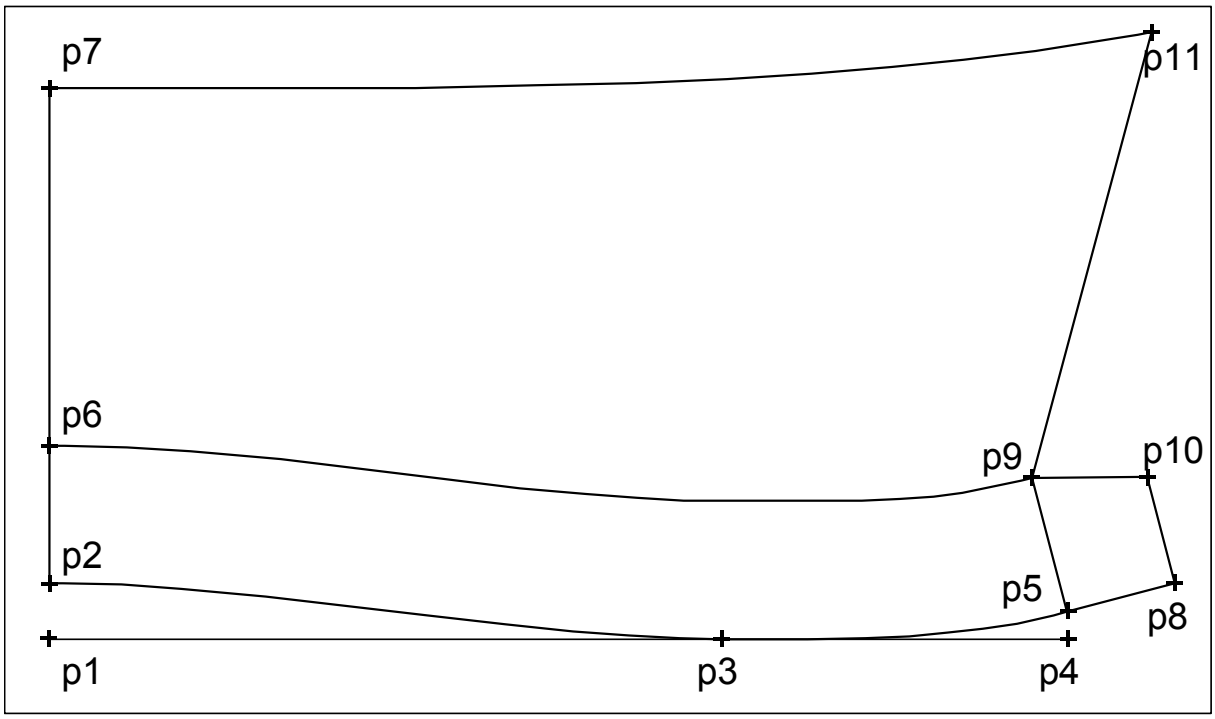


Bild 19-10

**Inhalt von Modul_Ansatzlinien.qpr:**

```
'************************************************************************
  Function qKgAnsatz(rHm,rVm,rRi5z,rMin,rZiel,p1,p2,p3,p4,p5)
' ... ermittelt die Kragenansatzlinie, die ein Minimum bei rRi5 %
' zwischen HM und VM hat. Die Übergabe-Parameter:
'   rHm  - Höherstellung HM
'   rVm  - Höherstellung VM
'   rRi5 - zusätzliche Richtung in p5 (an der VM)
'   rMin - Lage des Minimums in Prozent (ab HM)
'   rZiel- Ziellänge der Ansatzlinie (incl.Zugabe) von HM bis VM
' Zurückgegeben wird die Kurve qKgAnsatz, die in der HM beginnt,
' und die Punkte p1 bis p5.
'------------------------------------------------------------------------
  nVar n
  rVar rA,rRi5,rIst,rA1,rIst1,rA2,rIst2
'------------------------------------------------------------------------
  rCon rRe=0,rLi=180,rOb=90,rUn=270
'---------------------------------------------------------- Punkte der HM
  p1   = pXY(0,0)
  p2   = pPRiLng(p1,rOb,rHm)
'------------------------------------------------------------ 0.Näherung
  rA   = rZiel
  p3   = pPRiLng(p1,rRe,rMin/100*rA)
  p4   = pPRiLng(p1,rRe,rA)
  p5   = pPRiLng(p4,rOb,rVm)
  rRi5 = rRiPP(p3,p5)+rRi5z
  qKgAnsatz  = qSpline(p2,rRe,p3,rRe,p5,rRi5)
  rIst = rLngQ(qKgAnsatz)
'------------------------------------------------- Werte für 1.Näherung
  rA1  = 0
  rIst1= 0
  rA2  = rA
  rIst2= rIst
'------------------------------------------------------- Atom.Näherung
  For n = 1,10,1
    rA   = rNahInt(rA1,rIst1,rA2,rIst2,rZiel)
    p3   = pPRiLng(p1,rRe,rMin/100*rA)
    p4   = pPRiLng(p1,rRe,rA)
    p5   = pPRiLng(p4,rOb,rVm)
    rRi5 = rRiPP(p3,p5)+rRi5z
    qKgAnsatz  = qSpline(p2,rRe,p3,rRe,p5,rRi5)
    rIst = rLngQ(qKgAnsatz)
    If(rAbs(rIst-rZiel)<<0.01) Then
      Exit For
    End If
    rA1  = rA2
    rIst1= rIst2
    rA2  = rA
    rIst2= rIst
  End For
  End Function
'************************************************************************
```

**Inhalt von Modul Main.gpr:**

```
'*********************************************************************
  Program Main()
' Hemdkragenkonstruktion mit automatischer Anpassung der
' Kragenansatzlinie an die Länge des gemessenen Halsloches
' nach einer Vorlage von Frau Prof. H.Brückner, Berlin
'---------------------------------------------------------------------
  nVar n
  rVar rKgLng,rZiel,rRi5,rRi8,rRi11
  pVar p1,p2,p3,p4,p5,p6,p9,p7,p8,p10,p11
  qVar qV,qH,q1,q2,q3
  cVar cV,cH
'---------------------------------------------------------------------
  rCon rRe=0,rLi=180,rOb=90,rUn=270
'------------------------------------------------------------- X-Werte
  XTitel("Hemdkragen")
  Defx(1,"Zugabe zur kragenansatzlinie",0)
  Defx(2,"Höherstellung HM",10)
  Defx(3,"Höherstellung VM",5)
  Defx(4,"Zusätzl. Richtung Stegansatz in p5",10)
  Defx(5,"Stegbreite HM",25)
  Defx(6,"Kragenbreite HM",65)
  Defx(7,"Übertrittbreite Steg",20)
  Defx(8,"Minderung Stegbreite am Übertritt",5)
  Defx(9,"Spitzenübertritt in X-Richtung",15)
  Defx(10,"Spitzenübertritt in Y-Richtung",10)
  Defx(11,"Richtung Kragenspitze",10)
  Defx(12,"Lage p3 zwischen p1-p4 in %",66)
'---------------------------------- Länge des Halslochlinien erfragen
  cV  = cPick(1,4,"VORDERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cV, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qV  = qCo(cV,"qq")
  cH  = cPick(2,4,"HINTERE Halslochkuve anpicken !","Kragen","!",nT)
  If (not lCo(cH, "iO")) Then
     FEnd(0)
     Exit Program
  Endif
  qH  = qCo(cH,"qq")
  rKgLng = rLngQ(qV)+rLngQ(qH)
  If(rKgLng<<rX(2)) Then
    n   = nIBox("Die Halslochlinien sind zu kurz !")
    Exit Program
  End If
  rZiel  = rKgLng+rX(1)
'------------------------------------------------------ Kragenansatzlinie
  q1  = qKgAnsatz(rX(2),rX(3),rX(4),rX(12),rZiel,p1,p2,p3,p4,p5)
'------------------------------------------------------ Punkte p6 bis p11
  p6  = pPRiLng(p2,rOb,rX(5))
  Paral(-rX(5):q2=q1)
  p9  = pQend(q2)
  p7  = pPRiLng(p6,rOb,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,rOb,rX(2)+rX(5)+rX(6)+rX(10))
  p11 = pPRiLng(p11,rRe,rX(9))
'------------------------------------------------- Kragenaussenlinie
  rRi11  = rX(11)
  q3  = qSpline(p7,rRe,p11,rRi11)
'------------------------------------------------------------- Ausgaben
  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 Konstruktionsbaustein Schulternahtverlegung mit dem Ersetzen von Pos-Objekten

Bisher wurden ausschließlich neue Objekte an das Grafis-Protokoll übergeben. Für Konstruktionsbausteine, die auf bereits existierende Objekte angewendet werden sollen, muß es auch eine Möglichkeit geben, auf Objekte des Grafis-Protokolls zuzugreifen. Die sogenannten Pos-Nummern erlauben dies.

***Jedes Objekt (Punkt, Linie) des Grafis-Protokolls hat eine eindeutige Pos-Nummer, mit der es identifiziert wird.***

Die Pos-Nummer gilt innerhalb eines Teiles und wird für jedes neue Objekt automatisch angelegt. Alle Konstruktionsschritte des Grafis-Protokolls verweisen über die Pos-Nummer auf das betreffende Objekt.

**Die Verwendung der Pos-Nummern**

Vorhandene Objekte, die mit einem Konstruktionsbaustein zu bearbeiten sind, werden mit der Funktion `cPick()` eingelesen. Aus dem Pick-Container kann die Pos-Nummer als ganzzahliger Parameter entnommen werden. Beispiel:

```
cXx=cPick(1,1,tK,tT,tB,nT)
```

Die Bedeutung der Parameter ist auf Seite 19-5 erläutert.

Diesem Container wird die Pos-Nummer des angepickten Punktes wie folgt entnommen:

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

Ein Punkt/ eine Linie des Grafis-Protokolls wird ersetzt, indem bei der Ausgabe aus dem Fachsprachen-Programm auch die Pos-Nummer des zu ersetzenden Objektes angegeben wird. In Ausgabeanweisungen mit Pos-Nummern kann jeweils nur ein Objekt ausgegeben werden. Die Pos-Nummern können für Punkte, Strecken und Kurven übergeben werden. Im Grafis-Protokoll wird später nicht zwischen Strecken und Kurven unterschieden; beides sind Linien.

**`AusP(nXx,pXx)`**

| | |
|---|---|
| `nXx` | Pos-Nummer des Punktes |
| `pXx` | auszugebender Punkt |

**`AusQ(nXx,qXx)`** oder **`AusQ(nXx,sXx)`**

| | |
|---|---|
| `nXx` | Pos-Nummer der Linie |
| `qXx` | auszugebende Linie/Kurve |

**Anwendung der Pos-Nummern im Konstruktionsbaustein „Schulternahtverlegung“**

Neben der Anwendung der Pos-Nummern soll dieses Beispiel auch verdeutlichen:

- Der Aufbau eines sinnvollen Dialoges mit dem Nutzer. Zunächst erhält der Nutzer Informationen und wird dann aufgefordert, die benötigten Objekte anzuklicken.
- Die Sicherheit des Programms hat eine sehr große Bedeutung. Das Programm muß später auf die verschiedensten Anwendungsfälle sicher „reagieren“. Für die Schulternahtverlegung wird dazu die Länge der Schulterlinien verglichen und jede Ecke auf Lücken und überstehende Linien geprüft.
- Jeder Konstruktionsbaustein sollte so wenig wie möglich an den ursprünglichen Objekten ändern, auch wenn zunächst keine negativen Folgeeffekte zu erkennen sind. Die ursprüngliche Orientierung der Linien wird daher am Ende des Konstruktionsbausteines wieder hergestellt.
- Eine ausreichende Kommentierung ist notwendig. Dies gilt unabhängig davon, ob spätere Änderungen/Korrekturen voraussichtlich vom ursprünglichen Programmierer oder einer anderen Person durchgeführt werden. Beurteilen Sie beim Einarbeiten in den Konstruktionsbaustein Schulternahtverlegung, ob Ihnen die Kommentierung ausreicht.
- Eine Skizze mit Punkt- und Linienbezeichnungen analog Bild 19-11 gehört ebenfalls zu jedem Projekt. Gleiches gilt für eine kurze Beschreibung der Vorgehensweise.

Für die Schulternahtverlegung werden Objekte gemäß Bild 19-11 vorausgesetzt bzw. dafür vorbereitet.

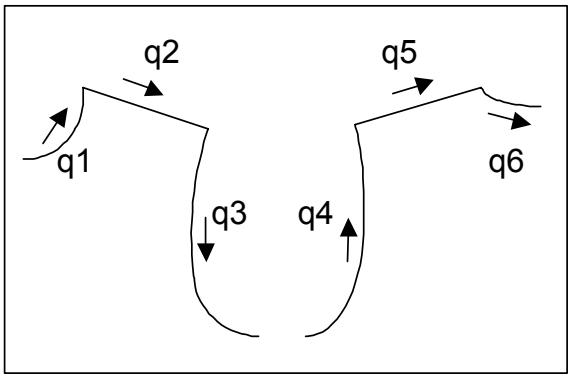


Bild 19-11

Das Fachsprachen-Programm für die Schulternaht ist wie folgt aufgebaut:

- ➢ Info-Maske für den Anwender erstellen und anzeigen. Der Anwender wird über die Voraussetzungen und die Einstellmöglichkeiten informiert.
- ➢ Die Kurven werden geeignet vorbelegt, so daß das Programm in der Entwicklungsumgebung getestet werden kann.
- ➢ Den Anwender auffordern, das Halsloch VT anzuklicken. Die Pos-Nummer des Halsloch VT wird dabei auf `nPos1` gespeichert.
- ➢ Den Anwender auffordern, die Schulterlinie VT anzuklicken. Die Pos-Nummer der Schulterlinie VT wird dabei auf `nPos2` gespeichert.

- Mit der externen Funktion `lKurven_richten_alle()` die Halsloch- und Schulterlinie analog Bild 19-11 orientieren und die Ecke prüfen. Falls keine eindeutige Ecke gefunden wird, mit einer Meldung abbrechen. Auf den Variablen `lDreh1` usw. wird abgelegt, ob die jeweilige Kurve umorientiert wurde oder nicht.
- Den Anwender auffordern, die Armlochlinie VT anzuklicken. Die Pos-Nummer der Armlochlinie VT wird dabei auf `nPos3` gespeichert.
- Mit der externen Funktion `lKurven2_richten()` die Armlochlinie analog Bild 19-11 orientieren und die Ecke prüfen. Falls keine eindeutige Ecke gefunden wird, mit einer Meldung abbrechen.
- Den Anwender auffordern, das Armloch RT anzuklicken. Die Pos-Nummer des Armloch RT wird dabei auf `nPos4` gespeichert.
- Den Anwender auffordern, die Schulterlinie RT anzuklicken. Die Pos-Nummer der Schulterlinie RT wird dabei auf `nPos5` gespeichert.
- Mit der externen Funktion `lKurven_richten_alle()` die Armloch- und Schulterlinie analog Bild 19-11 orientieren und die Ecke prüfen. Falls keine eindeutige Ecke gefunden wird, mit einer Meldung abbrechen.
- Die Schulterlinien von VT und RT vergleichen und gegebenenfalls mit einer Meldung abbrechen.
- Den Anwender auffordern, die Halslochlinie RT anzuklicken. Die Pos-Nummer der Halslochlinie VT wird dabei auf `nPos6` gespeichert.
- Mit der externen Funktion `lKurven2_richten()` die Halslochlinie analog Bild 19-11 orientieren und die Ecke prüfen. Falls keine eindeutige Ecke gefunden wird, mit einer Meldung abbrechen.

Jetzt folgen die eigentlichen Transformationsschritte:

- Halsloch- und Armlochlinien des RT an das VT transformieren und anschließend koppeln.
- Schulterpunkt auf dem Halsloch um X1 und auf dem Armloch um X2 nach vorn verschieben.
- Kurven neu belegen und zurücktransformieren.
- Falls eine Kurve gedreht wurde, diese in die ursprüngliche Richtung umorientieren.
- Kurven des Konstruktionsprotokolls durch die neuen Kurven ersetzen. Die Kurven werden auf die ursprünglichen Pos-Nummern ausgegeben.

Es folgt das komplette Programm Main() mit den Externen Funktionen `lKurven_richten_alle` und `lKurven2_richten` im Modul „Zusatz“.

**Inhalt von Modul Main.qpr:** **Seite 1/4**

```
'************************************************************************
  Program Main()
'------------------------------------------------------------------------
' Konstruktionsbaustein Schulternahtverlegung
'---------------------------------------- Informationen für den Anwender
' Die Verlegebeträge werden über X-Werte eingestellt.
' Voraussetzungen für diesen Konstruktionsbaustein:
' - Schulterlinien gleicher Länge in VT und RT
' - gerade Schulterlinien (keine Kurven) und
' - keine Lücken zwischen den Kurven und den Schulterlinien.
'-------------------------------- Informationen über den internen Ablauf
' q1: Halsloch VT, q2: Schulter VT, q3: Armloch VT
' q4: Armloch RT, q5: Schutler RT, q6: Halsloch RT
' lDreh1 bis lDreh6 zeigt an, ob die o.g. Kurve umorientiert wurde.
' Die Kurven werden am Ende des Programms in der ursprünglichen
' Richtung wieder ausgegeben.
'------------------------------------------------------------------------
  lVar l,lDreh1,lDreh2,lDreh3,lDreh4,lDreh5,lDreh6
  nVar n,nPos1,nPos2,nPos3,nPos4,nPos5,nPos6,nT
  rVar r1,r2
  pVar pShals,pSarml
  sVar s1,s2
  qVar q1,q2,q3,q4,q5,q6,q1t,q3t
  tVar tInfo,t,tAllg,tTop,t1
  cVar c1,c2,c3,c4,c5,c6
```

**Inhalt von Modul Main.qpr:** **Seite 2/4**

```
'-----------------------------------------------------------------------
  lCon
  nCon
  rCon
  tCon
'------------------------------------------------------------- X-Werte
  XTitel("Schulternahtverlegung")
  Defx(1,"Verlegebetrag am Halsloch nach vorn",10)
  Defx(2,"Verlegebetrag am Armloch nach vorn",10)
'------------------------------------------------ Vorbelegung der Kurven
  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))
'--------------------------------- Objekte abfragen und jeweils prüfen
  tAllg= "bilden keine eindeutige Ecke."+tC(13,10)+
&        "Korrigieren Sie die Ecke und verlegen"+tC(13,10)+
&        "dann die Schulter erneut !"
  tTop = "Schulternahtverlegung"
'------------------------------------------------------------ Halsloch VT
  tInfo= "Halsloch VT anklicken !"
  nT   = 0
  c1   = cPick(1,4,tInfo,tTop,"i",nT)
  q1   = qCo(c1,"qq")
'------------------------------------------------------------- Schulter VT
  tInfo= "Schulterlinie VT anklicken !"
  c2   = cPick(2,4,tInfo,tTop,"i",nT)
  q2   = qCo(c2,"qq")
  l    = lKurven_richten_alle(q1,q2,lDreh1,lDreh2)
  If(Not l) Then
    t    = "Die Kurven Halsloch VT und Schulterlinie VT"+tC(13,10)+tAllg
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'---------------------------------------- Schulterlinie eine Gerade ?
  If(rAbs(rLngQ(q2)-rAbstPP(pQanf(q2),pQend(q2)))>>0.05) Then
    t    = "Die Schulterlinie ist gekrümmt."+tC(13,10)
&         +"Dieser Fall ist nicht vorbereitet."
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'------------------------------------------------------------ Armloch VT
  tInfo= "Armlochlinie VT anklicken !"
  c3   = cPick(3,4,tInfo,tTop,"i",nT)
  q3   = qCo(c3,"qq")
  l    = lKurve2_richten(q2,q3,lDreh3)
  If(Not l) Then
    t    = "Die Kurven Schulterlinie VT und Armloch VT"+tC(13,10)+tAllg
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'------------------------------------------------------------- Armloch RT
  tInfo= "Armloch RT anklicken !"
  c4   = cPick(4,4,tInfo,tTop,"i",nT)
  q4   = qCo(c4,"qq")
'------------------------------------------------------------- Schulter RT
  tInfo= "Schulterlinie RT anklicken !"
  c5   = cPick(5,4,tInfo,tTop,"i",nT)
  q5   = qCo(c5,"qq")
  l    = lKurven_richten_alle(q4,q5,lDreh4,lDreh5)
```

**Inhalt von Modul Main.qpr:** **Seite 3/4**

```
  If(Not l) Then
    t    = "Die Kurven Armloch RT und Schulterlinie RT"+tC(13,10)+tAllg
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'---------------------------------------- Schulterlinie eine Gerade ?
  If(rAbs(rLngQ(q5)-rAbstPP(pQanf(q5),pQend(q5)))>>0.05) Then
    t    = "Die Schulterlinie ist gekrümmt."+tC(13,10)
&         +"Dieser Fall ist nicht vorbereitet."
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'---------------------------------------- Schulterlinien vergleichen
  If(rAbs(rLngQ(q2)-rLngQ(q5))>>0.5) Then
    t    = "Die Schulterlinien von VT und RT"+tC(13,10)
&         +"sind unterschiedlich lang !"
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If '------------------------------------------------------- Halsloch RT
  tInfo= "Halslochlinie RT anklicken !"
  c6   = cPick(6,4,tInfo,tTop,"i",nT)
  q6   = qCo(c6,"qq")
  l    = lKurve2_richten(q5,q6,lDreh6)
  If(Not l) Then
    t    = "Die Kurven Schulterlinie RT und Halsloch RT"+tC(13,10)+tAllg
    n    = nIBox(t,"Schulternahtverlegung",24)
    Exit Program
  End If
'--------------------- Halsloch- und Armlochkurven des RT transformieren
  s1   = sPP(pQanf(q5),pQend(q5))
  s2   = sPP(pQend(q2),pQanf(q2))
  DrehTr(s1,s2:q4,q6)
'---------------------------------------------------------- Kurven koppeln
  q1t  = qKop(q1+q6)
  q3t  = qKop(q4+q3)
'------------------------------------------- Schulterpunkt verschieben
  pShals  = pQend(q1)
  pSarml  = pQanf(q3)
  pShals  = pQPlng(q1t,pShals,-rX(1))
  pSarml  = pQPlng(q3t,pSarml,rX(2))
'------------------------------------------------ alle Kurven neu bilden
  q1   = qQbisP(q1t,pShals)
  q2   = qKop(pShals+pSarml)
  q3   = qQabP(q3t,pSarml)
  q4   = qQbisP(q3t,pSarml)
  q5   = qKop(pSarml+pShals)
  q6   = qQabP(q1t,pShals)
'------------------------------------------- Kurven des RT zurücksetzen
  DrehTr(s2,s1:q4,q5,q6)
```

**Inhalt von Modul Main.qpr:** **Seite 4/4**

```
'---------------------------------- Kurven in Originalrichtung drehen
  If(lDreh1) Then
    q1   = -q1
  End If
  If(lDreh2) Then
    q2   = -q2
  End If
  If(lDreh3) Then
    q3   = -q3
  End If
  If(lDreh4) Then
    q4   = -q4
  End If
  If(lDreh5) Then
    q5   = -q5
  End If
  If(lDreh6) Then
    q6   = -q6
  End If
'------------------------------------------------------------------ Ausgaben
  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
'**************************************************************************
```

**Inhalt von Modul: Zusatz.qpr** **Seite 1/2**

```
'**************************************************************************
  Function lKurven_richten_alle(q1,q2,lDreh1,lDreh2)
'--------------------------------------------------------------------------
' Die Kurven werden so orientiert, dass der Endpunkt der ersten
' direkt am Anfangspunkt der zweiten Kurve liegt. Konnten die Kurven
' korrekt orientiert werden, hat die Funktion den Wert True,
' anderenfalls den Wert False.
'--------------------------------------------------------------------------
  nVar n
  qVar q1t,q2t
  pVar pEndq1,pAnfq2
'------------------------------ Varianten abfragen, Kurve umorientieren
  lKurven_richten_alle= False
  For n = 1,4,1
    If(n==1) Then
      q1t  = q1
      q2t  = q2
      lDreh1  = False
      lDreh2  = False
    Else If(n==2) Then
      q1t  = -q1
      q2t  = q2
      lDreh1  = True
      lDreh2  = False
```

**Inhalt von Modul: Zusatz.qpr** **Seite 2/2**

```
    Else If(n==3) Then
      q1t  = q1
      q2t  = -q2
      lDreh1  = False
      lDreh2  = True
    Else If(n==4) Then
      q1t  = -q1
      q2t  = -q2
      lDreh1  = True
      lDreh2  = True
    End If
    pEndq1  = pQend(q1t)
    pAnfq2  = pQanf(q2t)
    If(rAbs(rAbstPP(pEndq1,pAnfq2))<<0.5) Then
      lKurven_richten_alle= True
      q1   = q1t
      q2   = q2t
      Exit For
    End If
  End For
  End Function
'************************************************************************
'************************************************************************
  Function lKurve2_richten(q1,q2,lDreh2)
'------------------------------------------------------------------------
' Die ZWEITE Kurve wird so orientiert, dass der Endpunkt der ersten
' direkt am Anfangspunkt der zweiten Kurve liegt. Konnten die zweite
' Kurven korrekt orientiert werden, hat die Funktion den Wert True,
' anderenfalls den Wert False.
' lDreh2 ist True, falls q2 gedreht wurde.
' q1 wird nicht gedreht.
'------------------------------------------------------------------------
  nVar n
  qVar q1t,q2t
  pVar pEndq1,pAnfq2
'------------------------------ Varianten abfragen, Kurve umorientieren
  lKurve2_richten  = False
  For n = 1,2,1
    If(n==1) Then
      q1t  = q1
      q2t  = q2
      lDreh2  = False
    Else If(n==2) Then
      q1t  = q1
      q2t  = -q2
      lDreh2  = True
    End If
    pEndq1  = pQend(q1t)
    pAnfq2  = pQanf(q2t)
    If(rAbs(rAbstPP(pEndq1,pAnfq2))<<0.5) Then
      lKurve2_richten  = True
      q1   = q1t
      q2   = q2t
      Exit For
    End If
  End For
'------------------------------------------------------------------------
  End Function
'************************************************************************
```