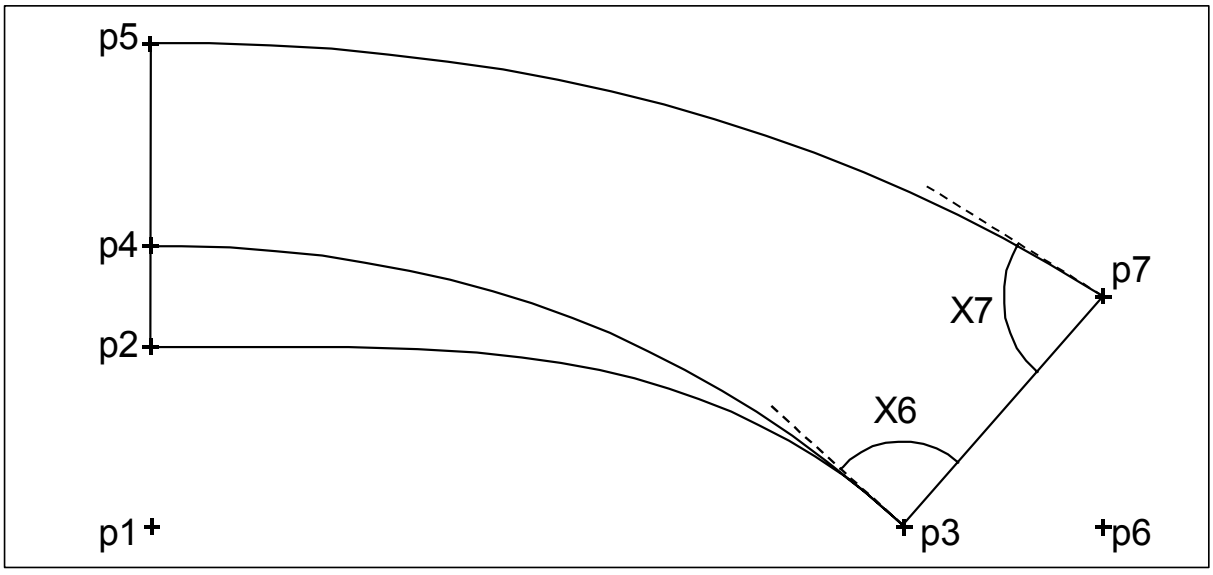
Kapitel 18 „Fachsprache I“

Inhalt

Die Grafis-Fachsprache dient zur Entwicklung von Grundkonstruktionen und Konstruktionsbausteinen. Die einzelnen Schritte zur Erstellung einer Grundkonstruktion werden als Text eingegeben. Grundkonstruktionen sollten dann in der Fachsprache entwickelt werden, wenn die firmenspezifische Passform oder komplette Baukastenlösungen zu entwickeln sind. Es ist zu bedenken, daß die Fachsprache eine abstrakte Form der Schnittentwicklung ist. Neben exzellenten Konstruktionskenntnissen und Erfahrung in der Anwendung von Grafis wird auch eine längere Einarbeitungszeit benötigt. Zur Entwicklung von Produktionsschnitten aus vorbereiteten, angepaßten Grundformen sind die Funktionen des Grafis-Dialoges das geeignetere Werkzeug.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar rWi3,rWi7
  pVar p1,p2,p3,p4,p5,p6,p7
  sVar
  qVar q1,q2,q3
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  rCon rKgLng=150
  tCon
'----------------- X-Wert-Definitionen
  XTitel("Bündchenkragen")
  Defx(1,"Höherstellung HM",35)
  Defx(2,"Kragenumfaltbreite",20)
  Defx(3,"Kragenbreite HM",40)
  Defx(4,"Kragenspitze(X) zu p3",40)
  Defx(5,"Kragenspitze(Y) zu p3",45)
  Defx(6,"Wi Ansatz+Umfaltl.in p3",90)
  Defx(7,"Wi Außenlinie in p7",80)
'----------------------- Punkte der HM
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rOb,rX(2))
  p5 = pPRiLng(p4,rOb,rX(3))
'-------------------- Eckpunkt p3 (VM)
  p3 = pXY(rKgLng,0)
  p6 = pPRiLng(p3,rRe,rX(4))
  p7 = pPRiLng(p6,rOb,rX(5))
'------------------ Kragenansatzlinie
  rWi3  = rWiPPP(p6,p3,p7)
  rWi3  = rWi3+rX(6)
  q1 = qSpline(p3,rWi3,p2,rLi)
  q2 = qSpline(p3,rWi3,p4,rLi)
'------------------- Kragenaußenlinie
  rWi7  = rRiPP(p7,p3)-rX(7)
  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
'************************************
```



18.1 Ein einfaches Programm: Quadrat

Einleitende Bemerkungen

Die neue Fachsprache gehört ab Version 8 zum Lieferumfang von Grafis. Sie ist eine Compiler-orientierte Programmiersprache. Die Programme werden nicht mehr interpretativ abgearbeitet, sondern in einem maschinennahen Code. Die Abarbeitung des Fachsprachenprogrammes wird dadurch deutlich schneller. In der neuen Fachsprache sind viele Werkzeuge enthalten, die auch in anderen Programmiersprachen üblich sind; jeweils angepasst an die Bedingungen der Schnittkonstruktion. Auch die Nutzung von Unterprogrammen ist jetzt möglich, so daß häufige oder sich wiederholende Berechnungen als Unterprogramme abgelegt werden können. Die Übersichtlichkeit vor allem in der Textdarstellung und die Syntaxprüfung sind gegenüber der vorhergehenden Fachsprache besser und genauer geworden. Anwendern, die mit der vorhergehenden Fachsprache gearbeitet haben, wird die jetzige Fachsprache zunächst komplexer erscheinen. Bei näherer Betrachtung wird jedoch deutlich, daß die neuen Techniken eine kürzere und übersichtlichere Programmierung ermöglichen.

Quadrat

Im ersten Projekt soll zunächst ein Quadrat (Bild 18-1) konstruiert werden, das später in das „Haus vom Nikolaus“ abgewandelt wird.

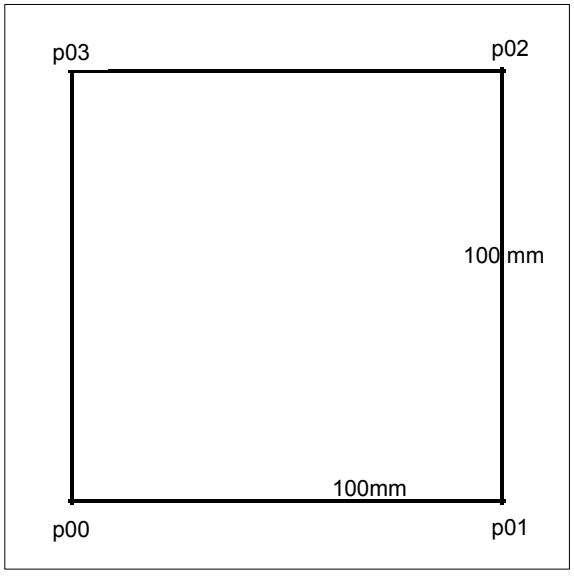


Bild 18-1

Legen Sie wie folgt das neue Projekt „Quadrat“ an:

⇒ *Extras | Neuer Compiler*

⇒ *Projekt | Neu...*

⇒ Projektname: Quadrat

⇒ 9-stelliges Kürzel Ihres Namens (analog Bild 18-4) eingeben

⇒ 2-stelliges Kürzel der Produktgruppe eingeben, z.B. „LB“ für Lehrbuch-Übungen

⇒ Mit <OK> wird das Projekt angelegt.

Die Struktur des Programms `Main()` wurde im Editierfenster (Bild 18-5) angelegt. Bearbeiten Sie das Programm wie folgt:

```
'************************************
Program Main()
'- Programm: Quadrat
'- Deklarationszeilen
lVar
nVar
rVar
pVar
sVar
qVar
tVar
cVar
'- Konstanten
lCon
nCon
rCon
tCon
'- Zuweisungen / Befehlszeilen
'- Programmende
End Program
'************************************
```

Nach einem ersten Compilieren mit dem Button  oder <F4> wurde das bisherige Programm automatisch formatiert:

```
'************************************
  Program Main()
'------------------ Programm: Quadrat
'------------------ Deklarationszeilen
  lVar
  nVar
  rVar
  pVar
  sVar
  qVar
  tVar
  cVar
'-------------------------- Konstanten
  lCon
  nCon
  rCon
  tCon
'--------- Zuweisungen / Befehlszeilen
'----------------------- Programmende
  End Program
'************************************
```

Zur Konstruktion eines Quadrates kann wie folgt vorgegangen werden:

⇒ Punkte p00 bis p03 belegen

⇒ Punkte p00 bis p03 ausgeben

⇒ Verbindungslinien zwischen den Punkten ausgeben

Folgende Zuweisungen / Befehlszeilen führen zum Ziel:

```
'--------- Zuweisungen / Befehlszeilen
  p00= pXY(0,0)
  p01= pXY(100,0)
  p02= pXY(100,100)
  p03= pXY(0,100)
```

Die Funktion `pXY()` konstruiert einen Punkt aus dessen anzugebender X- und Y-Koordinate. Die X-Koordinate des Punktes `p01` hat damit den Wert 100 und dessen Y-Koordinate den Wert 0.

```
  AusP(p00,p01,p02,p03)
```

Mit dem Befehlswort AusP werden die aufgelisteten Punkte auf den Bildschirm ausgegeben. Ohne diese Zeile sind die Punkte zwar im Programm belegt, werden jedoch nicht auf dem Bildschirm dargestellt.
Nach Eingabe der fünf Zeilen ist das Programm mit  zu compilieren und dann mit  im Probelauf zu testen. Erst nach  wird auf den Grafis-Bildschirm umgeschaltet und die Punkte des Quadrates erscheinen. Der Grafis-Bildschirm wird mit der rechten Maustaste wieder geschlossen.

Mit dem Befehlswort AusQ werden einzelne Strecken, Kurven oder Polygonzüge ausgegeben. Wählen Sie „Innere Fkt“ in der Variablen Liste und klicken dort auf AusQ. Unterhalb des Editierfensters erscheint ein Hilfetext zur markierten Funktion. Die Verbindungslinien werden mit

```
AusQ(p00+p01+p02+p03+p00)
```

als über die Ecken hinweg gekoppelte Verbindungslinie ausgegeben, mit den Zeilen

```
AusQ(p00+p01,p01+p02)
AusQ(p02+p03,p03+p00)
```

als einzelne Linien von Ecke zu Ecke. Die Strecken müssen nicht zuvor als Variablen gebildet werden. Die Berechnung kann auch direkt im Funktionsaufruf erfolgen. Die Zeilen

```
s1=sPP(p00,p01)
AusQ(s1)
```

führen zum gleichen Ergebnis wie die Zeile

```
AusQ(sPP(p00,p01))
```

oder wie

```
AusQ(p00+p01)
```

In der ersten Variante wird die Strecke zunächst auf die Variable s1 geschrieben und erst danach ausgegeben. In der zweiten Variante wird die Strecke direkt im Befehlsaufruf gebildet. Die Funktion sPP(p00,p01) bildet dazu eine Streckenvariable als Verbindung zwischen den beiden anzugebenden Punkten.

Variante „Haus vom Nikolaus“

Das Quadrat kann noch zum „Haus vom Nikolaus“ (Bild 18-2) abgewandelt werden. Dazu wird ein zusätzlicher Punkt p04 als Dachspitze konstruiert. Die Linien sind durchgängig auszugeben.
Folgende Zuweisungen / Befehlszeilen führen zum Ziel:

```
'----------Zuweisungen / Befehlszeilen
 p00= pXY(0,0)
 p01= pXY(100,0)
 p02= pXY(100,100)
 p03= pXY(0,100)
 p04= pXY(50,150)
 AusP(p00,p01,p02,p03,p04)
AusQ(p00+p02+p01+p00+p03+p04+p02+p03+p0
1)
```

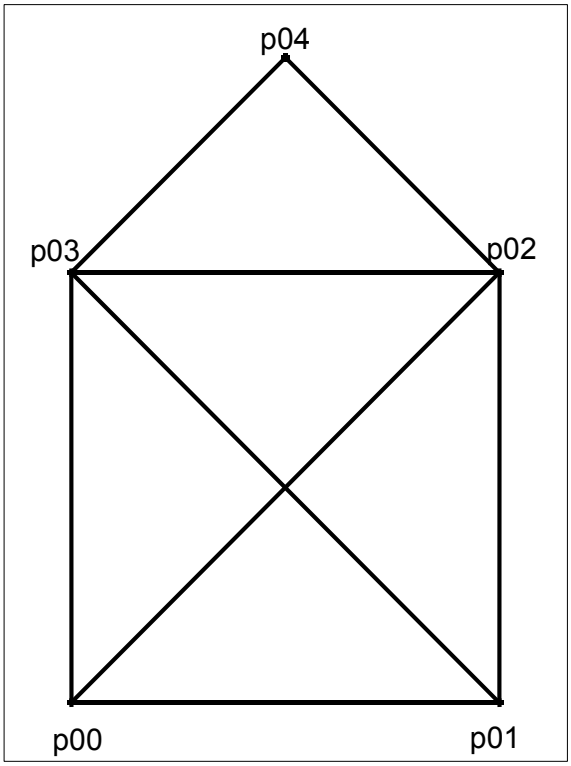


Bild 18-2

Speichern Sie das Projekt über *Projekt* | *Speichern* und verlassen über *Projekt* | *Beenden* die Projektoberfläche.

18.2 Datenbasis und Oberfläche

Datenbasis

Die Entwicklung einer Grundkonstruktion läuft in einem sogenannten Projekt ab. Zu einem Projekt gehören

- die Module im Klartext,
- der Objetcode zum Projekt und
- das ausführbare Fachsprachen-Programm als Ergebnis.

Das Projekt selbst wird als Verzeichnis gespeichert. In diesem Verzeichnis befinden sich die Module im Klartext sowie der Objektcode. Das ausführbare Fachsprachen-Programm wird direkt in das \PROG – Verzeichnis des jeweiligen Konstruktionssystems gespeichert. Bild 18-3 enthält eine detaillierte Übersicht.

Zum ***Kopieren*** oder ***Duplizieren*** genügt es, das komplette Verzeichnis \Grafis\Module\[*Ko-system*]\[*Projektname*] zu kopieren. Alle zum Projekt gehörenden Dateien sind damit erfasst. Zur ***Weitergabe*** eines getesteten und freigegebenen Fachsprachenprogramms genügt es, die Datei *.cpr aus dem Verzeichnis \Grafis\[*Ko-system*]\PROG zu kopieren.

Grafis	
|--Basis_D	
| |--PROG	*.cpr-Dateien (Beispiel: `KFriedric_DA_c001_00.cpr`) Diese ausführbaren Fachsprachen-Programme enthalten alle Informationen, die zur Abarbeitung des Moduls erforderlich sind. Damit Modelle, in denen dieses Modul verwendet wurde, auf einem anderen Rechner laufen, muß diese Datei mitgeliefert werden.
|	
|	
|	
|-- ...	
|--Module	
|--Basis_D	
|--\[*Projektname*\]	Jedes Projekt erhält unter \Grafis\Module\[Ko-system]\ ein eigenes Verzeichnis, in das alle Dateien zum Projekt gespeichert werden. Das Projekt „Oberkoerper 01“ im Konstruktionssystem „Optimass“ wird als Verzeichnis „\Grafis\Module\Basis_D\Oberkoerper 01“ angelegt. Im Projektverzeichnis befinden sich folgende Dateien:

Modul.ini	Initialisierungsdatei zum Projekt
Main.qpr	Quellcode des Hauptprogrammes « Main » im RTF-Format
Main.qpt	Quellcode des Hauptprogrammes « Main » als ASCI-Text
Main.opr	Objektcode Hauptprogrammes « Main »

… und weitere Dateien *.qpr, *.qpt und *.opr eventuell vorhandener Unterprogramme.

Bild 18-3

Neues Projekt anlegen

Zur Entwicklung einer neuen Grundform wird Grafis mit dem gewünschten Konstruktionssystem gestartet. Das Programmieren mit der Fachsprache sollte in einem neuen Modell stattfinden, damit wichtige Modelle nicht versehentlich zurückgesetzt oder überschrieben werden.

Ein neues Projekt wird über das Pull-Down-Menu ***Extras*** | ***Neuer Compiler*** und anschließend ***Projekt*** | ***Neu...*** angelegt. Der Projektname darf keine Sonderzeichen (z.B. „!+-ßäöü) enthalten. Eine geeignete Bezeichnung wäre beispielsweise „Oberkoerper 01“.

Zusätzlich zum Projektnamen (=Verzeichnis für die Entwicklungsdateien, siehe Bild 18-3) ist auch ein Bezeichner für das ausführbare Fachsprachenprogramm zu vergeben. Dazu öffnet sich das Fenster „Name der Programmdatei generieren“ (Bild 18-4).

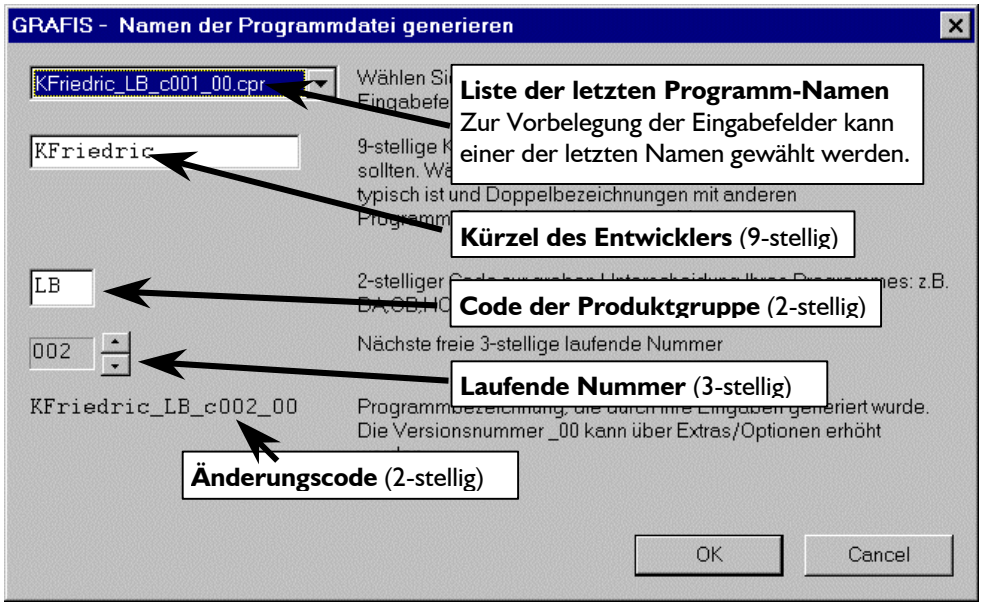


Bild 18-4

Name der Programmdatei

Der Name der Programmdatei hat eine fest vorgegebene Länge von 20 Zeichen. Gegenüber der Fachsprache der Versionen 7 und früher wurde er um 12 Zeichen verlängert. Damit sind die Programme besser unterscheidbar und eine doppelte Namensgebung wird vermieden.

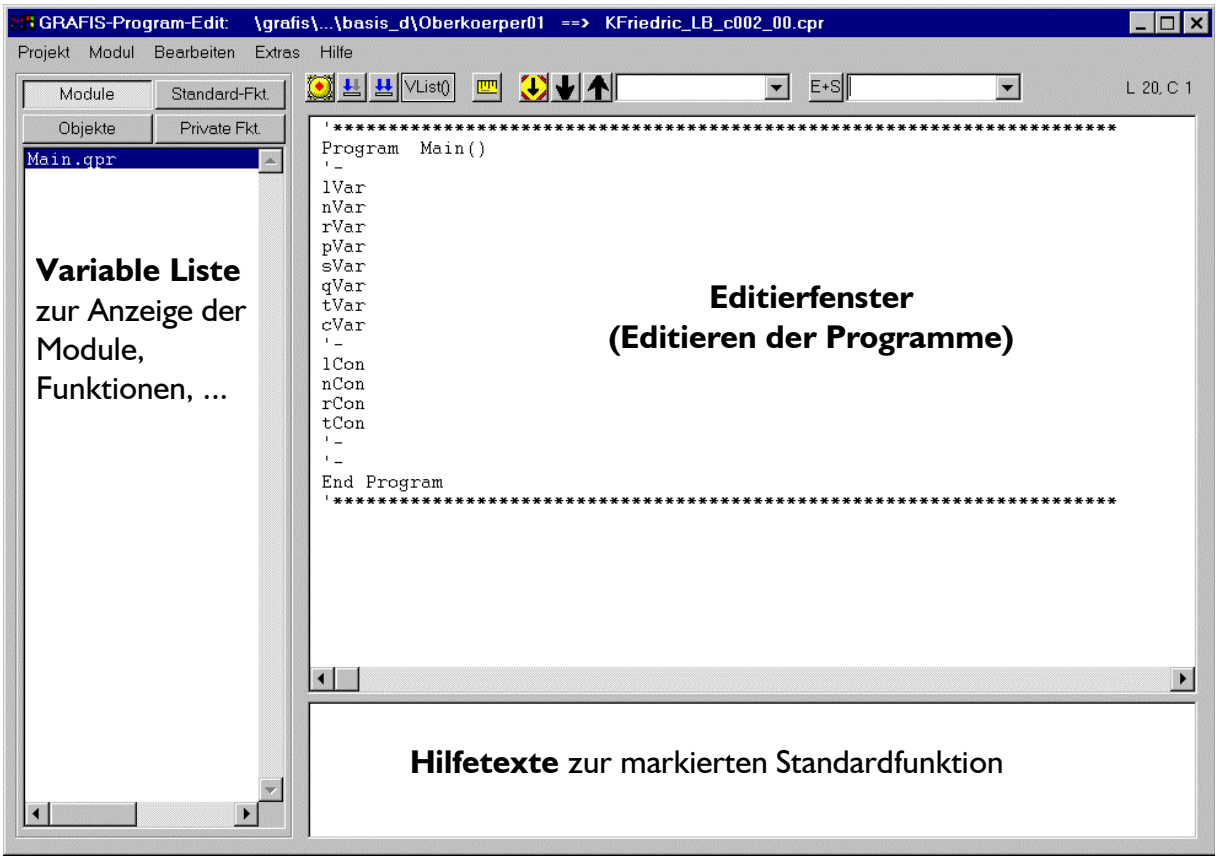


Bild 18-5

Der Name setzt sich jetzt aus folgenden Kürzeln zusammen:

KFriedric_DA_c000_00
|-- 9-stelliges ***Kürzel des Entwicklers***

Beispiele:

KFriedric	für Kerstin Friedrich
BBachmann	für Birgit Bachmann
FSBeautyW	für Friederike Sauer, Mitarbeiterin der Firma BeautyWear
RWRollerD	für Roland Wagner, Mitarbeiter der Firma Roller Design

KFriedric_DA_c000_00
|-- 2-stelliges ***Kürzel der Produktgruppe***

Beispiele:

DA	für Damenoberbekleidung
HE	für Herrenoberbekleidung
KI	für Kinder

KFriedric_DA_c000_00
|
3-stellige ***Laufende Nummer***

KFriedric_DA_c000_00
|
2-stelliger ***Änderungscode***

Nur für das erste Projekt des Entwicklers ist das Fenster komplett auszufüllen. Dazu gehören insbesondere das Kürzel des Entwicklers und das Kürzel der Produktgruppe. Die Laufende Nummer wird von Grafis automatisch vorgeschlagen und sollte nur bei Bedarf verändert werden.

Der Änderungscode muß höher gesetzt werden, wenn ein bereits ausgeliefertes/ benutztes Programm überarbeitet werden soll. Der Änderungscode wird in der Projektoberfläche über *Extras | Optionen* erhöht.

Für weitere Projekte kann aus der Liste der letzten Programmnamen ein geeigneter ausgewählt und bei Bedarf angepasst werden.

Die Projektoberfläche

Die drei größten Bereiche der Projektoberfläche (Bild 18-5) sind

- das Editierfenster zur Eingabe der Programme,
- die Variable Liste zur Anzeige vorhandener Module, Funktionen, ... und
- ein Feld mit Hilfetexten zur markierten Standardfunktion.

Das Programm **Main** wird als „leeres“ Programm in der erforderlichen Struktur sofort angelegt, siehe Bild 18-5. Alle Tastatureingaben und die Button über dem Editierfenster wirken nur auf den Programmtext. Die Variable Liste und das Feld mit Hilfetexten sind Anzeigebereiche ohne Eingabemöglichkeit.

Editierfunktionen

Zum Editieren sind nutztbar:

←→↑↓	Kursor nach links, rechts, oben, unten
↵ (Enter)	Zeilenwechsel
Pos 1	Kursor an Zeilenanfang
Ende	Kursor an Zeilenende
Entf	nächstes/markiertes Zeichen löschen
Ctrl+Pos 1	Kursor an Programmanfang
Ctrl+Ende	Kursor an Programmende
rechte Maustaste	Öffnen des Kontextmenü = Pull-Down-Menü *Bearbeiten*

Analog zu anderen Editoren ist das Markieren einer oder mehrerer Zeilen durch Anklicken am linken Zeilenrand möglich. Markierte Zeilen werden mit gedrückter linker Maustaste verschoben oder bei zusätzlich gedrückter Strg/Ctrl-Taste kopiert. Weitere Funktionen zum Editieren befinden sich im Pull-Down-Menü *Bearbeiten*, welches auch mit der rechten Maustaste als Kontextmenü geöffnet wird.

Button Compilieren und Testen

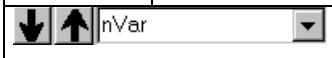
Die häufigsten Funktionen bei der Programmentwicklung befinden sich auf den Button oberhalb des Editierfensters.

Der erste Block von Button enthält Funktionen zum Compilieren und Testen des Programms:

	Compilieren (Syntaxprüfung und erste Übersetzung)
	Bilden & Testen des Programms im Probelauf
	Bilden & Testen des Programms mit Gradieren
	Ein-/Ausschalter für VList-Einträge; Bei eingeschaltetem VList()-Button, wird das Programm beim Bilden & Testen an den VList-Haltepunkte angehalten. Die Werte der angegebenen Variablen erscheinen in der Variablen Liste.
	Bildschirmanzeige des Ergebnisses vom letzten Bilden & Testen des Programms

Button Suchen und Ersetzen

Der zweite Block von Button enthält Funktionen zum Suchen und Ersetzen von Zeichen:

	Markieren aller Zeichen im Programm, die gleich dem Zeichen rechts im Feld sind
	Suchen der Zeichen rechts im Feld
	Ersetzen des markierten Zeichens durch den Begriff rechts von 

Markieren Sie den Suchbegriff, z.B. p01, im Programm. Er erscheint automatisch im Suchfeld rechts neben  , in das der Begriff auch eingegeben werden kann. Anklicken von  oder  markiert die nächste gefundene Zeichenkette.

Für das Ersetzen von z.B. p01 durch p02 ist folgende Vorgehensweise zu empfehlen:

⇒ Markieren des Suchbegriffs im Programm,

⇒ Eingabe des Ersatzbegriffes (hier: p02) rechts neben  ,

⇒ Anklicken von  , sofern der markierte Begriff durch den Ersatzbegriff ersetzt werden soll.

Die Variable Liste

Die Button über der Variablen Liste „Module“, „Innere Fkt.“, usw. wirken wie Karteikartenreiter. Nach Anklicken eines dieser Button wird in der Variablen Liste angezeigt:

Button	**Inhalt der Variablen Liste**
Module	alle Programm-Module (*.qpt-Dateien) zum aktuellen Projekt
Innere Fkt.	alle Inneren Funktionen
Externe Fkt.	alle Externen Funktionen des aktuellen Projektes (aus allen Programm-Modulen)
Objekte	die ausgegebenen Objekte (Punkte + Linien) mit den Daten (o-Objektnummer, ty-Objekttyp, po-Pos-Nummer)

Die Automatische Formatierung

Jedes Compilieren, das mit  oder <F4> gestartet wird, führt neben der Syntaxprüfung auch eine automatische Formatierung des Programmtextes durch. Bei der automatischen Formatierung werden Befehlsworte blau gekennzeichnet, Kommentare erscheinen grün.

Befehlszeilen sind grundsätzlich um zwei Zeichen eingerückt; in Schleifen jeweils um zwei Zeichen weiter. Der erste Buchstabe des Variablennamens ist zwingend klein und der zweite Buchstabe groß geschrieben.

Zur Vereinfachung der Eingabe von Kommentarzeilen gilt folgende zusätzliche Regelung:

1. ***Befindet sich ein einzelner Anführungsstrich in der 1. Spalte, dann füllt das dem Anführungsstrich folgende Zeichen die ganze Zeile.***

```
'-  wird...
'-----------------------------------
'*  wird...
'***********************************
```

2. ***Befindet sich ein einzelner Anführungsstrich in der zweiten oder einer folgenden Spalte, dann wird der Text rechtsbündig formatiert.***

```
 '---Initialisierung  wird...
                    '---Initialisierung
```

```
'Ausgabe der Punkte  wird...
                'Ausgabe der Punkte
```

3. Ein einzelner Anführungsstrich mit nachfolgendem Leerzeichen bewirkt, daß der Text unverändert bleibt.

```
' ---Initialisierung  bleibt...
' ---Initialisierung
  ' Ausgabe der Punkte  bleibt...
  ' Ausgabe der Punkte
```

Kommentare können auch in einer Programmzeile rechts vom Befehlstext stehen. Auch dabei gelten die zweite und dritte Regel.

Hinweis: Testen Sie das automatische Formatieren und die Regeln für die Kommentierung bereits mit dem noch „leeren“ Programm Main().

18.3 Regeln der Programmierung

Grundregeln

- ✓ Ein Programm ist ***zeilenweise*** aufgebaut.
- ✓ Jede Zeile enthält eine ***Zuweisung*** oder einen ***Befehl***.
- ✓ Die ***Zeilenbreite*** sollte 64 Zeichen nicht überschreiten.
- ✓ Die ***Groß-/Kleinschreibung***, ***Zeichenabstände*** und auch eventuelle ***Leerzeichen am Zeilenanfang*** formatiert Grafis automatisch beim Compilieren.
- ✓ Der Anführungsstrich ' kennzeichnet den folgenden Text als ***Kommentar***, der nicht abgearbeitet wird.
- ✓ Das Zeichen „&“ in der ersten Spalte kennzeichnet eine ***Fortsetzungszeile***.
- ✓ Das Modul Main() muß in jedem ***Projekt*** enthalten sein und das Programm Main() enthalten.
- ✓ Jedes Projekt hat genau ein ***Programm*** mit dem Namen Main(). Dieses Programm wird beim Aufruf als erstes abgearbeitet.
- ✓ Jedes Projekt kann beliebig viele ***Innere Funktionen*** und auch beliebig viele ***Externe Funktionen*** enthalten. Die Inneren Funktionen gehören zum Grafis-Lieferumfang. Externe Funktionen werden vom Anwender programmiert.
- ✓ Jedes Programm beginnt mit „`Program Main()`“ und endet mit „`End Program`“.
- ✓ Jede Funktion beginnt mit „`Function xXxx()`“ und endet mit „`End Function`“.

Variablen

In Grafis werden Variablen verschiedener Typen eingesetzt. Der Variablen-Name kann bis zu 64 Zeichen lang sein, wobei das erste Zeichen den Variablentyp kennzeichnet. Variablen können erst dann verwendet werden, wenn sie am Beginn des Programms oder der Funktion deklariert wurden. Bei der Deklaration der Variablen wird Speicherplatz reserviert und genullt. Die Variable bleibt dann bis zum Ende der Funktion oder des Programms verfügbar.

Es gibt folgende ***Variablentypen***:

Typ	Erläuterung	Beispiel
lXxx	logische Variable, die den Wert True oder False annehmen kann	lFrage
nXxx	Nummer (ganze Zahl), die einen Wert zwischen $-2*10^9$ und $+2*10^9$ (2.000.000.000) annehmen kann	nNum
rXxx	reelle Zahl, auf 6 Nachkommastellen genau	r01
pXxx	Punkt mit X- und Y-Koordinate	pAe
sXxx	Strecke mit Anfangs- und Endpunkt	sSaum
qXxx	Polygonzug / Kurve / Linienzüge (q steht für que – engl. Schlange, Folge)	qArm
tXxx	Texte mit bis zu 10.000 Zeichen	tHilfe
cXxx	Container	cBox

Alle verwendeten Variablen müssen im Programm-/ Funktionskopf deklariert werden. Die ***Deklarationszeilen*** beginnen mit `lVar` für logische Variablen, mit `nVar` für ganzzahlige Variablen und so weiter. Für jeden Variablentyp kann es mehrere Deklarationszeilen geben.

Beispiel:

```
nVar nIst1,nIst2,nIndex
```

Die ***Werte der Variablen*** werden über Zuweisungszeilen gesetzt.

Die Variablentypen l (logisch), n (ganzzahlig), r (reel) und t (Text) können am Programm-/ Funktionsanfang auch als ***Konstanten*** definiert werden. Die Definitionszeilen für Konstanten beginnen mit `lCon` für logische Variablen, mit `nCon` für ganzzahlige Variablen und so weiter.

Beispiel:

```
nCon nIst1=1,nIst2=2
```

Konstanten dürfen nicht gleichzeitig als Variablen deklariert sein.

Variablen und Konstanten gelten nur innerhalb des Programms bzw. der Funktion, in der sie deklariert wurden.

Alle während der Programmierung neu verwendeten Variablen werden beim Compilieren automatisch in den Deklarationszeilen nachgetragen, vorausgesetzt, es gibt mindestens eine (auch „leere“) Deklarationszeile für diesen Variablentyp.

Zuweisung

Das Zeichen „=“ steht in allen Programmiersprachen für eine Zuweisung. Im Unterschied zur Gleichung in der Mathematik bedeutet es hier:

Der Wert des Ausdrucks rechts vom „=“ wird der Variablen links vom „=“ zugewiesen. Links vom „=“ muß deshalb eine Variable stehen.

Die folgende Zeile wäre als mathematische Gleichung falsch. Als Zuweisungszeile beim Programmieren hat sie die nachfolgende Bedeutung:

```
nZahl=nZahl+2
```

Die Variable `nZahl` muß zunächst am Programmkopf deklariert sein. Beim Abarbeiten dieser Zeile wird zunächst der Ausdruck rechts vom „=“ berechnet und dann auf die Variable links vom „=“ geschrieben. Hat `nZahl` vor dem Abarbeiten der Zeile den Wert 5, dann ergibt der Ausdruck rechts vom „=“ den Wert 7. Nach Abarbeiten der Zeile hat sich somit der Wert von `nZahl` um 2 erhöht.

Befehl / Anweisung

Mit Befehlen / Anweisungen werden beim Programmieren Operationen ausgeführt, die auf ein oder mehrere Objekte wirken können. In Grafis gibt es Befehle / Anweisungen zum Verschieben, Drehen, Spiegeln oder zur Bildschirmausgabe eines oder mehrerer Objekte. Befehle / Anweisungen beginnen im Unterschied zur Zuweisung sofort mit dem Befehlswort.

Innere Funktionen

Innere Funktionen sind vorbereitete Funktionen, die zum Grafis - Lieferumfang gehören. Innere Funktionen, die einen Wert liefern, werden in Berechnungen verwendet. Innere Funktionen, die eine Operation ausführen, werden in Befehlszeilen / Anweisungen eingesetzt. Der Umfang der Inneren Funktionen reicht aus, um alle üblichen Schritte bei der Konstruktion eines Schnittes programmieren zu können.

Nach dem Öffnen eines Projektes und dem Anklicken des Button „Innere Fkt.“ (über der Variablen Liste) werden in der Variablen Liste alle Inneren Funktionen angezeigt. Anklicken einer Funktion markiert sie. Gleichzeitig erscheinen unter dem Editierfenster die Hilfetexte zur markierten Funktion. Doppelklick auf die Funktion übernimmt sie ins Programm.

Das erste Zeichen des Funktionsnamens von Inneren Funktionen, die einen Wert liefern, ist ein Typkennzeichen für den gelieferten Wert. Die Typen sind identisch mit den Variablentypen. Die Funktion **r**`G()` liefert einen reelen Wert. Die Funktion **p**`PRiLng()` liefert einen Punkt.

Zuweisen von Werten

Deklarierte Variablen werden durch folgende Anweisungen mit einem Wert belegt.

Logische Variable

```
lFrage1=False
lFrage2=True
```

Nummer / Ganzzahlige Variable

```
nIndex=1
```

Mathematische Berechnungen (Addition, Subtraktion, Multiplikation, Division) aus Zahlen, ganzzahligen/reellen Variablen sowie ganzzahligen/reellen Funktionen sind möglich. Liefert der Ausdruck rechts vom „=“ keinen ganzzahligen Wert, dann wird auf die nächste ganze Zahl gerundet.

Reele Variable

```
rAbstand=920*2/3+14
```

Analog zu ganzzahligen Variablen sind auch hier mathematische Berechnungen möglich. Das Ergebnis wird jedoch nicht gerundet.

Punkt

```
p00=pXY(0,0)
```

Punkte werden unter Verwendung der Inneren Funktionen gesetzt. Auch das Kopieren eines Punktes mit `p31=p30` ist möglich.

Strecke

```
sSaum=sPP(p31,p42)
```

... analog Punkt

Zusätzlich kann mit `sSaum=-sSaum` die Orientierung der Strecke geändert werden.

Kurve

```
qArm=qSpline(p01,r01,p02,r02)
```

... analog Strecke

Text

```
tInfo="Mein erstes Programm."
```

Der Text muß immer zwischen Anführungszeichen stehen.

18.4 Programm Gradierbares Rechteck

Gradierbares Rechteck

Ein gradierbares Rechteck (Breite: Brustumfang, Höhe: Körperlänge) soll nun konstruiert werden (Bild 18-6). Folgende Zeilen führen zum Ziel:

```
'*************************************
*
  Program Main()
'----- Programm: Gradierbares Rechteck
'------------------ Deklarationszeilen
  lVar
  nVar
  rVar rBreite,rHoehe
  pVar p00,p01,p02,p03
  sVar
  qVar
  tVar
  cVar
'-------------------------- Konstanten
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  tCon
'----------Zuweisungen / Befehlszeilen
  p00= pXY(0,0)
  rBreite  = rG(1)
  rHoehe= rG(3)
  p01= pPRiLng(p00,rRe,rBreite)
  p02= pPRiLng(p01,rOb,rHoehe)
  p03= pPRiLng(p02,rLi,rBreite)
  AusP(p00,p01,p02,p03)
  AusQ(p00+p01,p01+p02)
  AusQ(p02+p03,p03+p00)
```

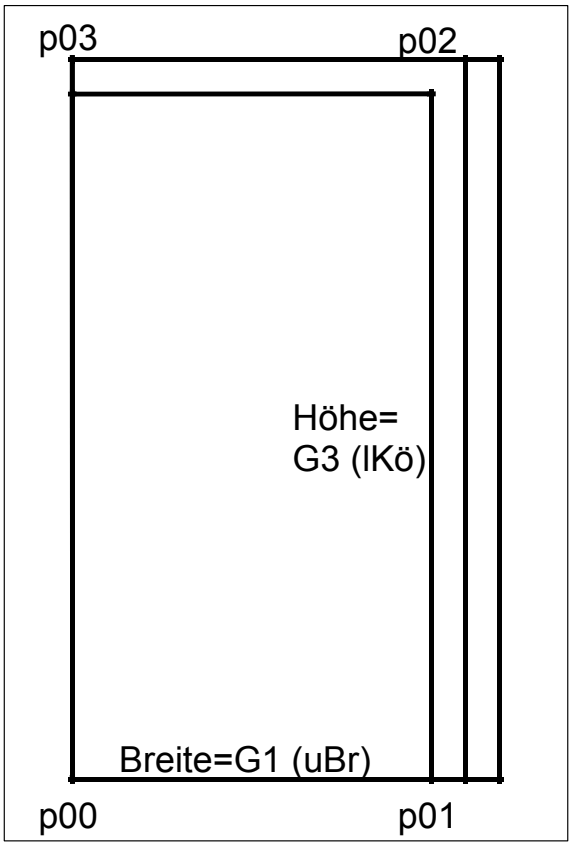


Bild 18-6

```
'------------------------ Programmende
  End Program
'************************************
```

Die Deklarationen der neuen Variablen `rBreite` und `rHoehe` wird automatisch beim ersten Compilieren durchgeführt. Es ist nicht nötig, die Variablen selbst in die Deklarationszeilen einzutragen.

Richtungen

Richtungen werden in Winkel-Graden angegeben. Ein Punkt wird nach links abgetragen, wenn für die Richtung der Wert 180 gesetzt ist. Alle Winkelangaben beziehen sich auf die positive X-Achse und entgegen dem Uhrzeigersinn (Bild 18-7). Fällt es Ihnen schwer, sich die Richtungen in Winkel-Grad vorzustellen, dann sollten Sie mit Richtungskonstanten arbeiten, z.B. `rRe=0, rLi=180, rOb=90, rUn=270`; siehe auch Programm-Beispiel „Gradierbares Rechteck“.

Die Funktionen `pXY(),rG(),pRiLng()`

Die Zeilen im Block „Zuweisungen / Befehlszeilen“ haben folgende Bedeutung:

```
p00= pXY(0,0)
```

Die Variable `p00` wird mit Hilfe der Inneren Funktion `pXY()` belegt. Die Parameter nach der öffnen-

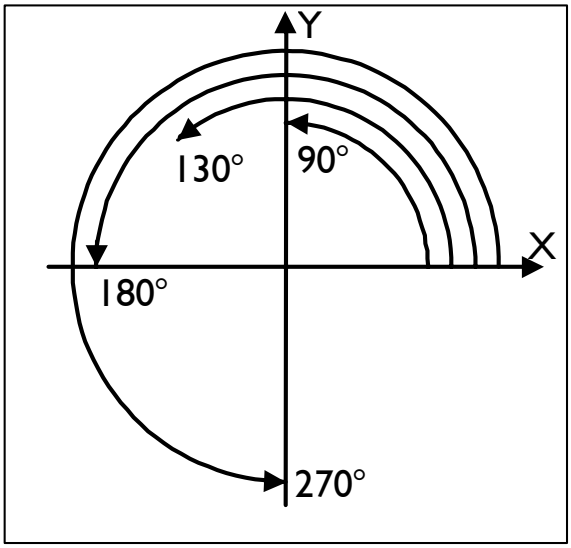


Bild 18-7

den Klammer geben die X- und die Y-Koordinate des Punktes an. In diesem Fall sind beide Koordinaten Null. Damit ist `p00` der Nullpunkt.

```
rBreite  = rG(1)
rHoehe= rG(3)
```

Den neuen Variablen `rBreite` und `rHoehe` werden Werte zugewiesen, die mit der Inneren Funktion `rG(n)` berechnet wurden. ***Die Funktion `rG(n)` ermittelt den n-ten Größenwert der Maßtabelle.*** Mit `rG(1)` wird der erste Wert der Maßtabelle (in Damen_5 und Basis_D: Brustumfang) und mit `rG(3)` der dritte Wert der Maßtabelle (in Damen_5 und Basis_D: Körperlänge) übergeben.

```
p01= pPRiLng(p00,rRe,rBreite)
```

Dem neuen Punkt `p01` wird das Ergebnis von `pPRiLng(p00,rRe,rBreite)` zugewiesen. `pPRiLng()` berechnet einen neuen Punkt, der ausgehend vom Punkt `p00` in Richtung `rRe` und im Abstand `rBreite` liegt.

In der Parameterliste von Inneren Funktionen können statt der Variablen auch Funktionen gleichen Typs und für reele/ganzzahlige Parameter auch Zahlen stehen. Die folgenden Zeilen haben daher gleiche Bedeutung:

```
p01= pPRiLng(p00,rRe,rBreite)
p01= pPRiLng(pXY(0,0),rRe,rG(1))
p01= pPRiLng(p00,0,rBreite)
```

In den Zeilen

```
p02= pPRiLng(p01,rOb,rHoehe)
p03= pPRiLng(p02,rLi,rBreite)
```

wird der Punkt `p02` ausgehend von `p01` abgetragen und zwar nach oben mit dem Abstand der Höhe des Rechteckes. Analoges gilt für `p03`.

Ausgabe der Objekte, Programm testen

```
AusP(p00,p01,p02,p03)
```

gibt die Eckpunkte des Rechteckes auf dem Bildschirm aus.

```
AusQ(p00+p01,p01+p02)
AusQ(p02+p03,p03+p00)
```

gibt die Verbindungslinien zwischen den Eckpunkten als einzelne Linien auf dem Bildschirm aus.

Nach Eingabe der Programmzeilen und Compilieren ist das Programm zu Bilden&Testen. Das Ergebnis erscheint nach . Mit der rechten Maustaste öffnet sich wieder die Programmier-Oberfläche. Mit statt wird das Programm nicht nur mit der Modellgröße, sondern in allen Größen der Gradiertabelle berechnet und nach auch dargestellt. Damit sich die Höhe des Gradierbaren Rechteckes ändert, müssen auch lange/ kurze Größen bzw. individuelle Größe eingetragen sein

18.5 Programm Bündchenkragen

Ein Bündchenkragen gemäß Bild 18-8 soll unter Verwendung folgender X-Werte programmiert werden.

X	Bezeichnung	Schritt	Wert
1	Höherstellung HM	p1⇒p2	35mm
2	Kragenumfaltbreite	p2⇒p4	20mm
3	Kragenbreite HM	p4⇒p5	40mm
4	Kragenspitze (X) zu p3	p3⇒p6	40mm
5	Kragenspitze (Y) zu p3	p6⇒p7	45mm
6	Winkel Umfalt+Ansatzlinie	in p3	90°
7	Winkel Kragenaußenlinie	in p7	80°

Verwendung von X-Werten

X-Werte müssen im Programmkopf direkt nach den Deklarationszeilen für Variablen und Konstanten definiert werden. Für die Definition der X-Werte gilt:

- ✓ Mit der Zeile
  ```
  XTitel(".............")
  ```
  wird ein Programmname übergeben, der später in der X-Wert-Liste der Grundkonstruktion erscheint. Daran erkennt der Anwender, zu welcher Grundkonstruktion diese X-Werte gehören. Der Titel darf 50 Zeichen lang sein.
- ✓ Die X-Wertangaben müssen am Beginn des Programmes mit folgender Befehlsstruktur stehen.
  ```
  Defx(1,".....",10.0)
  Defx(2,".....",12.5)
      | |           |
      | |           |_  Standardwert
      | |_____________  Kommentartext
      |_______________  Laufende Nummer
  ```
- ✓ Die Standardwerte dürfen nur eine Nachkommastelle haben und müssen im Wertebereich -3200. <= Wert <= 3200. liegen.
- ✓ Die laufende Nummer muß mit **1** beginnen und lückenlos ansteigen.
- ✓ Jedem X-Wert können größenbezogene Werte zugeordnet werden. Die Definitionszeile des X-Wertes wird dafür wie folgt erweitert:
  ```
  Defx(3,"Zugabe zu RL",0,
  &  "_36",3,"_46",4,"_036",1,"_046",2)
       |    |
  ```
  Größe Wert
  (_ steht für eine Standardmaßtabelle!)
  Das Zeichen & steht für eine Folgezeile. Die X-Wert-Definitionszeile kann mehrere Folgezeilen haben.

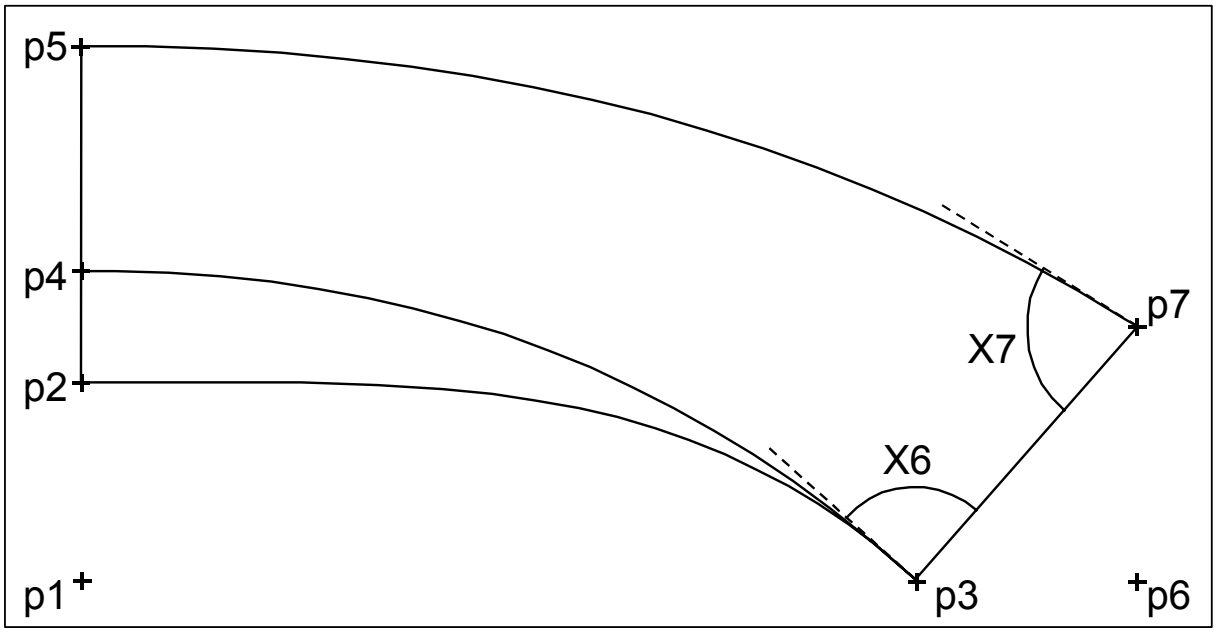


Bild 18-8

Der Kennung „Defx“ mit Laufender Nummer, Bezeichner und Standardwert könne auch größenbezogene X-Wert-Zuweisungen folgen. In Angabeblöcken wird einem Größenbezeichner ein Wert zugeordnet, den der X-Wert bei dieser Größe annehmen soll. Die Größenbezeichner für Standardmaßtabellen müssen mit "_" beginnen!

Programmieren der Punkte

Grundsätzlich wird ein Programm in Teilschritten erarbeitet und nach jedem Teilschritt getestet. Erst wenn der Teilschritt erfolgreich war, sollte fortgesetzt werden. Ein erster Teilschritt bei der Programmierung des Bündchenkragens ist das Programmieren der Punkte der Hinteren Mitte und anschließend der Punkte an der Kragenecke.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar
  pVar p1,p2,p4,p5
  sVar
  qVar
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  rCon
  tCon
'----------------- X-Wert-Definitionen
  XTitel("Bündchenkragen")
  Defx(1,"Höherstellung HM",35)
  Defx(2,"Kragenumfaltbreite",20)
  Defx(3,"Kragenbreite HM",40)
  Defx(4,"Kragenspitze(X) zu p3",40)
  Defx(5,"Kragenspitze(Y) zu p3",45)
  Defx(6,"Wi Ansatz+Umfaltl.in p3",90)
  Defx(7,"Wi Außenlinie in p7",80)
'---------------------- Punkte der HM
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rOb,rX(2))
  p5 = pPRiLng(p4,rOb,rX(3))
'--------------------- Punkte ausgeben
  AusP(p1,p2,p4,p5)
'------------------------------------
  End Program
'************************************
```

Die Angaben in der Zeile `pVar` trägt Grafis nach einem Compilieren automatisch ein. Die Zeile `rCon` enthält wiederum die Vorbelegung für die Hauptrichtungen. Danach schließt sich ein Block mit der Definition der X-Werte an. Für die erste Kommentarzeile genügt es,

```
'- X-Wert-Definitionen
```

einzugeben. Die restlichen Zeichen fügt Grafis beim automatischen Formatieren ein. Die X-Werte werden in den darauffolgenden Zeilen fortlaufend definiert und enthalten keine größenbezogenen X-Werte.

Nach der Definition der X-Werte werden die ersten Punkte konstruiert.

```
  p1 = pXY(0,0)
```

... definiert den Punkt p1 mit den Koordinaten (0,0). p1 ist damit der Nullpunkt der Konstruktion.

```
  p2 = pXY(0,rX(1))
```

... definiert einen Punkt `p2` mit den Koordinaten `(0,rX(1))`, wobei `rX(1)` den Wert des ersten X-Wertes übergibt. `p1` liegt damit um die „Höherstellung HM“ nach oben versetzt.

```
  p4 = pPRiLng(p2,rOb,rX(2))
```

… definiert einen Punkt `p4`, der von `p2` nach oben im Abstand `rX(2)` –dem zweiten X-Wert- abgetragen wird.

```
  p5 = pPRiLng(p4,rOb,rX(3))
```

… definiert einen Punkt `p5`, der von `p4` nach oben im Abstand `rX(3)` –dem dritten X-Wert- abgetragen wird.

Die Punkte der Hinteren Mitte liegen damit programmintern vor. Sie müssen noch auf dem Bildschirm ausgegeben werden. Dazu dienen die Zeilen

```
'------------------ Punkte ausgeben
  AusP(p1,p2,p4,p5)
```

Diesen ersten Teilschritt sollten Sie mit  ,  und  zunächst gründlich testen. Es erscheinen nur die Punkte der Hinteren Mitte auf dem Bildschirm. Messen Sie die Abstände zwischen den Punkten und gegebenenfalls auch deren Koordinaten. Mit der rechten Maustaste kehren Sie wieder in die Programmier-Oberfläche zurück. Speichern Sie das Projekt über *Projekt* | *Speichern*.

Im nächsten Teilschritt werden die Punkte an der Kragenecke konstruiert. Empfehlenswert ist, die Bildschirmausgaben in einem Block am Ende des Programms anzuweisen. Daher werden die nächsten Programmzeilen direkt vor „`Punkte ausgeben`“ eingefügt. Alle Ergänzungen sind hervorgehoben.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar
  pVar p1,p2,p3,p4,p5,p6,p7
  sVar
  qVar
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  rCon rKgLng=150
  tCon
'----------------- X-Wert-Definitionen
  XTitel("Bündchenkragen")
  Defx(1,"Höherstellung HM",35)
```

```
  Defx(2,"Kragenumfaltbreite",20)
  Defx(3,"Kragenbreite HM",40)
  Defx(4,"Kragenspitze(X) zu p3",40)
  Defx(5,"Kragenspitze(Y) zu p3",45)
  Defx(6,"Wi Ansatz+Umfaltl.in p3",90)
  Defx(7,"Wi Außenlinie in p7",80)
'----------------------- Punkte der HM
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rOb,rX(2))
  p5 = pPRiLng(p4,rOb,rX(3))
'-------------------- Eckpunkt p3 (VM)
  p3 = pXY(rKgLng,0)
  p6 = pPRiLng(p3,rRe,rX(4))
  p7 = pPRiLng(p6,rOb,rX(5))
'-------------- Punkte+Linien ausgeben
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(p2+p5)
  AusQ(p3+p7)
'-------------------------------------
  End Program
'*************************************
```

Die Kragenbreite soll in diesem Beispiel noch fest vorgegeben sein. In Abschnitt 19.2 wird erarbeitet, welche Anweisungen für eine Automatische Längenanpassung des Kragens an das Halsloch erforderlich sind. Die Kragenlänge `rKgLng` wird in der Zeile

```
  rCon rKgLng=150
```

als Konstante von 150mm festgelegt. Direkt vor der Ausgabe der Punkte und Linien wurde folgender Block ergänzt:

```
'----------------- Eckpunkt p3 (VM)
  p3 = pXY(rKgLng,0)
```

... definiert den Punkt `p3` mit den Koordinaten `(rKgLng,0)`. `p3` liegt damit im Abstand der Kragenlänge rechts vom Nullpunkt.

```
  p6 = pPRiLng(p3,rRe,rX(4))
```

… definiert einen Punkt `p6`, der von `p3` nach rechts im Abstand `rX(4)` –dem vierten X-Wert- abgetragen wird

```
  p7 = pPRiLng(p6,rOb,rX(5))
```

… definiert einen Punkt `p7`, der von `p6` nach oben im Abstand `rX(5)` –dem fünften X-Wert-. abgetragen wird

In der Zeile zur Ausgabe der Punkte wurden die neuen Punkte `p3`, `p6` und `p7` ergänzt.

```
  AusP(p1,p2,p3,p4,p5,p6,p7)
```

Mit den Zeilen

```
  AusQ(p2+p5)
  AusQ(p3+p7)
```

wird auch die Hintere Mitte als Verbindung zwischen den Punkten `p2` und `p5` sowie die Linie an der Kragenecke zwischen `p3` und `p7` sichtbar. Mit dem Ausgabebefehl `AusQ()` können Linien und Kurven zur Bildschirmausgabe angewiesen werden. Statt der Variablen ist auch die Angabe von Strecken-Funktionen erlaubt.

Testen und prüfen Sie diesen Teilschritt mit  ,  und  . Speichern Sie das Projekt.

Richtungen und Winkel berechnen

Richtungs- und Winkelangaben

Richtungen werden unter anderem beim Abtragen von Punkten in einer Richtung sowie bei der Konstruktion von Kurven benötigt. In der neuen Fachsprache erfolgen Richtungsangaben grundsätzlich als reele Zahlenwerte in Grad.

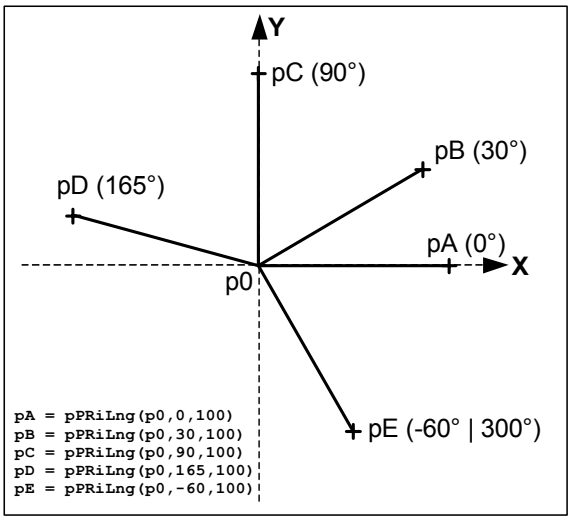


Bild 18-9

Die Punkte `pA` bis `pE` aus Bild 18-9 sind wie folgt programmierbar, wobei der Abstand zu p0 jeweils 100mm betragen soll:

```
  pA = pPRiLng(p0,0,100)
  pB = pPRiLng(p0,30,100)
  pC = pPRiLng(p0,90,100)
  pD = pPRiLng(p0,165,100)
  pE = pPRiLng(p0,-60,100)
```

Statt der Zahlenangaben in Grad könnten auch reele Variable als Parameter eingetragen werden.

Richtungen berechnen

Eine Richtung kann bestimmt werden als

- Richtung vom ersten zum zweiten Punkt mit `rRiPP(p,p)`,
- Richtung einer Strecke `rRiS(s)`,
- Richtung einer Kurve im Anfangs- bzw. Endpunkt `rRiQanf(q)` bzw. `rRiQend(q)` oder
- Richtung einer Kurve in einem Kurvenpunkt `rRiQP(q,p)`.

Aus mathematischer Sicht ist die Richtung gleichzusetzen mit einem Vektor. Erst wenn der Vektor mit einem Punkt verknüpft wird, entsteht eine Gerade.

Die Richtung des Punktes `pB` bezogen auf `p0` (Bild 18-9) lässt sich wie folgt berechnen:

```
  rB = rRiPP(p0,pB)
```

Nach Abarbeiten dieser Zeile hat `rB` den Wert 30.

Winkel berechnen

Ein Winkel berechnet sich als

- Winkel, bestimmt durch drei Punkte, mit `rWiPPP(p,p,p)` (Anfangs-, Dreh- und Endpunkt) oder
- Winkel zwischen zwei Strecken `rWiSS(s,s)`.

Für die Punkte gemäß Bild 18-9 ergeben die Funktionsaufrufe in der linken Spalte die Werte in der rechten Spalte.

Aufruf	Ergebnis
`rWiPPP(pA,p0,pB)`	+30
`rWiPPP(pB,p0,pA)`	-30
`rWiPPP(pD,p0,pE)`	+135
`rWiPPP(pD,p0,pC)`	-75
`rWiPPP(pE,p0,pA)`	+60

Der erste Parameter in `rWiPPP(p,p,p)` bestimmt den ersten Schenkel des Winkels. Bezogen auf diesen Schenkel bestimmt sich die Drehrichtung (positiver oder negativer Drehwinkel).
Gleiches gilt für die Funktion `rWiSS(s,s)`, bei der die Schenkel des Winkels zuvor als Strecken definiert werden müssen.

Die Kurvenvariante Spline

Eine Kurve in der Variante als Spline kann durch beliebig viele Stützpunkte verlaufen. In diesen Stützpunkten können auch Richtungen für den Kurvenverlauf angegeben werden. Wie bei einem Lineal aus Stahl, biegt sich die Kurve so, daß alle Bedingungen mit möglichst geringer Biegeenergie erfüllt werden.
Zur Definition einer Spline muß mindestens ein Anfangs- und ein Endpunkt angegeben werden. Die einfachste Variante mit

```
q1=qSpline(pA,pE)
```

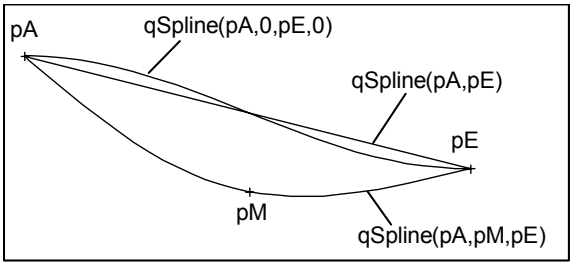


Bild 18-10

bestimmt eine Spline von pA nach pE. In diesen Punkten kann die Spline eine beliebige Richtung annehmen und wird daher als Strecke (Bild 18-10) erscheinen.
Mit der Zeile

```
q2 = qSpline(pA,0,pE,0)
```

wird die Kurve zusätzlich gezwungen, in den Punkten pA und pE mit Richtung 0° waagerecht nach rechts zu verlaufen. Für eine umgekehrte Kurvenrichtung wäre zu schreiben

```
q2 = qSpline(pE,180,pA,180)
```

Jede Kurve hat eine Richtung!

Mit der Zeile

```
q3 = qSpline(pA,pM,pE)
```

entsteht eine Kurve durch drei Punkte, wobei die Richtungen in den Punkten nicht vorgegeben sind.

Kragenansatz- und Kragenumfaltlinie als Spline mit Richtungsangaben konstruieren

Die Kragenansatz- und Kragenumfaltlinien sollen als Spline konstruiert werden. Der Anfangspunkt beider Kurven ist `p3`. Beide Kurven sollen im Winkel X6 beginnen, bezogen auf die Verbindung von `p3` nach `p7`. Dazu ist zunächst die Richtung von `p3` nach `p7` zu ermitteln (Bild 18-11).

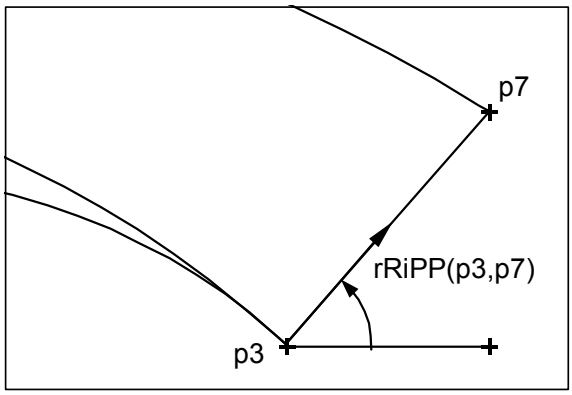


Bild 18-11

Die berechnete Richtung ist noch um den vorgegebenen Winkel zu drehen (Bild 18-12).

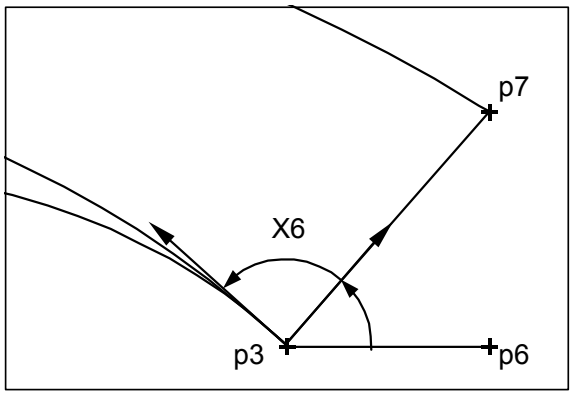


Bild 18-12

Die folgenden Zeilen führen zum Ziel:

```
rWi3  = rRiPP(p3,p7)
rWi3  = rWi3+rX(6)
q1 = qSpline(p3,rWi3,p2,rLi)
q2 = qSpline(p3,rWi3,p4,rLi)
```

Für die Kragenaußenlinie ist zu berücksichtigen, daß der Winkel in `p7` im Kragen innen angegeben wird. Die Richtung der Kurve in `p7` kann entweder mit

```
rWi7  = rRiPP(p7,p3)-rX(7)
```

oder mit

```
rWi7  = rRiPP(p3,p7)-180-rX(7)
```

berechnet werden. Nach

```
q3 = qSpline(p7,rWi7,p5,rLi)
```

ist die Kragenaußenlinie gebildet, jedoch noch nicht auf dem Bildschirm ausgegeben. Die Ausgabeanweisung für die drei Kurven lautet

```
AusQ(q1,q2,q3)
```

Das Programm für einen Bündchenkragen mit (noch) vorgegebener Kragenlänge ist fertig :

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar rWi3,rWi7
  pVar p1,p2,p3,p4,p5,p6,p7
  sVar
  qVar q1,q2,q3
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rLi=180,rOb=90,rUn=270
  rCon rKgLng=150
  tCon
'----------------- X-Wert-Definitionen
  XTitel("Bündchenkragen")
  Defx(1,"Höherstellung HM",35)
  Defx(2,"Kragenumfaltbreite",20)
  Defx(3,"Kragenbreite HM",40)
  Defx(4,"Kragenspitze(X) zu p3",40)
  Defx(5,"Kragenspitze(Y) zu p3",45)
  Defx(6,"Wi Ansatz+Umfaltl.in p3",90)
  Defx(7,"Wi Außenlinie in p7",80)
'----------------------- Punkte der HM
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rOb,rX(2))
  p5 = pPRiLng(p4,rOb,rX(3))
'-------------------- Eckpunkt p3 (VM)
  p3 = pXY(rKgLng,0)
  p6 = pPRiLng(p3,rRe,rX(4))
  p7 = pPRiLng(p6,rOb,rX(5))
'------------------ Kragenansatzlinie
  rWi3  = rWiPPP(p6,p3,p7)
  rWi3  = rWi3+rX(6)
  q1 = qSpline(p3,rWi3,p2,rLi)
  q2 = qSpline(p3,rWi3,p4,rLi)
'------------------- Kragenaußenlinie
  rWi7  = rRiPP(p7,p3)-rX(7)
  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
'************************************
```

18.6 Programm Rock

Die Grundkonstruktion Rock gemäß Bild 18-13 soll unter Verwendung der angegebenen X-Werte programmiert werden. Die Programmerstellung erfolgt in vier Teilschritten. Zu jedem Teilschritt gehören die Konstruktionsschritte (Tabelle), eine Abbildung und das Programm bis zum abgebildeten Zustand. Die Teilschritte sollten zunächst selbst erarbeitet und dann mit dem vorbereiteten Programmtext verglichen werden.

Bei der Erarbeitung des Rockes werden Sie möglicherweise folgende Fragen haben:

Was mache ich bei Fehlermeldungen ?
Wie finde ich die passende Funktion ?
Was ist bei der <u>Freigabe</u> eines Programmes zu berücksichtigen ?
Was ist bei der <u>Änderungen</u> / <u>Korrektur</u> eines Programmes zu berücksichtigen ?

Die Antworten auf diese Fragen finden Sie im letzten Abschnitt 18.6 dieses Kapitels.

X	Bezeichnung	Wert
1	Rocklänge ab Taille	600mm
2	Zugabe zum halben Hüftumfang	10mm
3	Zugabe zum halben Taillenumfang	10mm
4	Verschiebung der SN nach vorn	0mm
5	Höherstellung Seitennaht	10mm
6	Abnäherlänge Vorderrock	90mm
7	Abnäherspitze ab Hüftlinie im HR	35mm

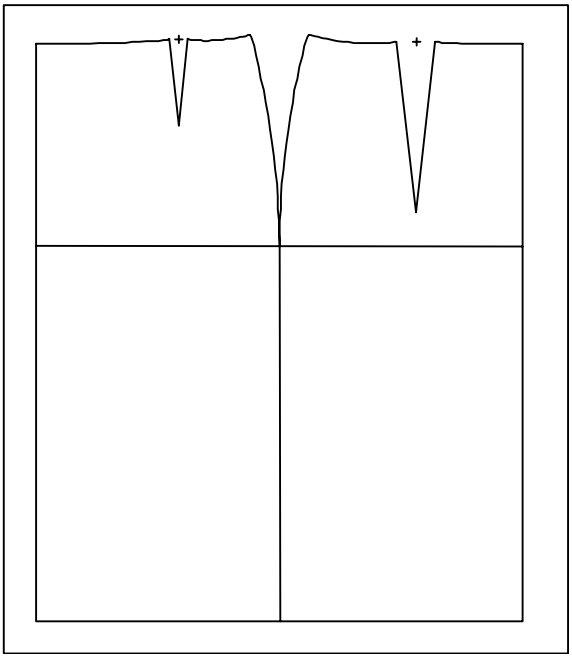


Bild 18-13

1.Teilschritt: Punkte der Hinteren Mitte, der Vorderen Mitte und der Seitennaht konstruieren (Bild 18-14)

von	bis	Richtg	Abstand
01	02	⇓	G10 (Hüfttiefe)
01	03	⇓	X1 (Rocklänge ab Taille)
01	05	⇐	G2/2+X2 (halber Hüftumfang + Zugabe)
02	04	⇐	G2/2+X2
03	06	⇐	G2/2+X2
02	07	⇐	½ Abstand p02⇔p04 + X4
01	08	⇐	½ Abstand p02⇔p04 + X4
03	09	⇐	½ Abstand p02⇔p04 + X4

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar rZ
  pVar p01,p02,p03,p04,p05,p06,p07,
&      p08,p09
  sVar
  qVar
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rOb=90,rLi=180,rUn=270
  tCon
'-------------- Definition der X-Werte
  XTitel("Rock")
  Defx(1,"Rocklänge ab Taille",600)
  Defx(2,"Zugabe 1/2 Hüftumfang",10)
  Defx(3,"Zugabe 1/2Taillenumfang",10)
  Defx(4,"Verschiebung SN n. vorn",0)
  Defx(5,"Höherstellung SN",10)
  Defx(6,"Abnäherlänge Vorderrock",90)
  Defx(7,"Abnspitze ab Hüftl. HR",35)
'----------------------- Punkte der HM
  p01 = pXY(0,0)
  p02 = pXY(0,-rG(10))
  p03 = pXY(0,-rX(1))
'----------------------- Punkte der VM
  rZ  = rG(2)/2+rX(2)
  p05 = pPRiLng(p01,rLi,rZ)
  p04 = pPRiLng(p02,rLi,rZ)
  p06 = pPRiLng(p03,rLi,rZ)
'----------------------- Punkte der SN
  rZ  = rAbstPP(p02,p04)/2+rX(4)
  p07 = pPRiLng(p02,rLi,rZ)
  p08 = pPRiLng(p01,rLi,rZ)
  p09 = pPRiLng(p03,rLi,rZ)
```

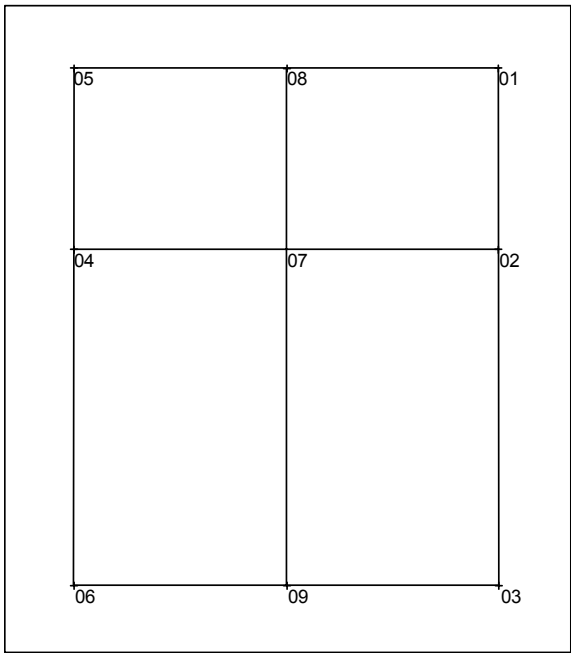


Bild 18-14

```
'------------------ Ausgabe der Punkte
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09)
'------------------ Ausgabe der Linien
  AusQ(p01+p03)
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
'------------------------------------
  End Program
'************************************
```

2.Teilschritt: Mehrweite berechnen & verteilen (Bild 18-15)

Mehrweite *Mw=(G2/2+X2)-(G4/2+X4)*
Anteil Seitennaht *3/6 Mehrweite*
Anteil Hinterrock *2/6 Mehrweite*
Anteil Vorderrock *1/6 Mehrweite*

von	bis	Richtg	Abstand
08	08	⇑	X5 (Höherstellung Seitennaht)
08	10	⇒	½ * 3/6 * *Mehrweite*
08	11	⇐	½ * 3/6 * *Mehrweite*
01	12	⇐	½ Abstand p01⇔p10
12	12	⇑	¼*X5 (Taillenringerhöhung)
12	13	⇒	½ * 2/6 * *Mehrweite*
12	14	⇐	½ * 2/6 * *Mehrweite*
05	15	⇒	2/3 Abstand p05⇔p11
15	15	⇑	½ *X5 (Taillenringerhöhung)
15	16	⇒	½ * 1/6 * *Mehrweite*
15	17	⇐	½ * 1/6 * *Mehrweite*

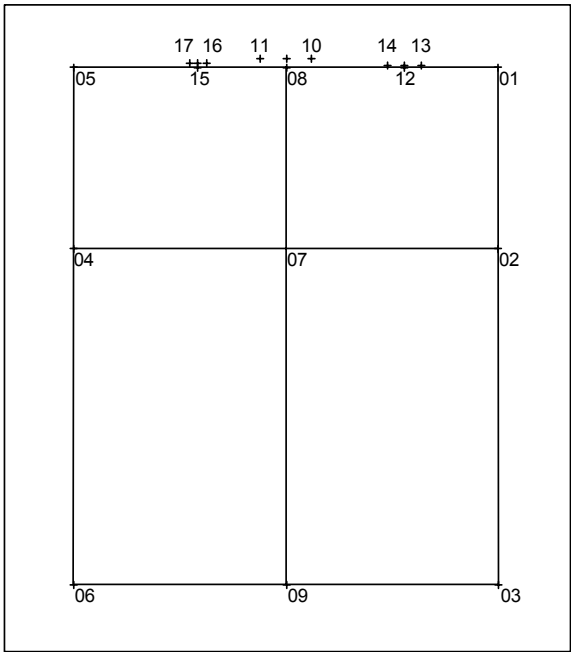


Bild 18-15

```
'************************************
  Program Main()
  lVar
  nVar
  rVar rZ,rMw,rSn,rHr,rVr
```

```
  pVar p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17
  sVar
  qVar
  tVar
  cVar
'-------------------------------------
  lCon
  nCon
  rCon rRe=0,rOb=90,rLi=180,rUn=270
  tCon
'-------------- Definition der X-Werte
  XTitel("Rock")
  Defx(1,"Rocklänge ab Taille",600)
  Defx(2,"Zugabe 1/2 Hüftumfang",10)
  Defx(3,"Zugabe 1/2Taillenumfang",10)
  Defx(4,"Verschiebung SN n. vorn",0)
  Defx(5,"Höherstellung SN",10)
  Defx(6,"Abnäherlänge Vorderrock",90)
  Defx(7,"Abnspitze ab Hüftl. HR",35)
'----------------------- Punkte der HM
  p01 = pXY(0,0)
  p02 = pXY(0,-rG(10))
  p03 = pXY(0,-rX(1))
'----------------------- Punkte der VM
  rZ  = rG(2)/2+rX(2)
  p05 = pPRiLng(p01,rLi,rZ)
  p04 = pPRiLng(p02,rLi,rZ)
  p06 = pPRiLng(p03,rLi,rZ)
'----------------------- Punkte der SN
  rZ  = rAbstPP(p02,p04)/2+rX(4)
  p07 = pPRiLng(p02,rLi,rZ)
  p08 = pPRiLng(p01,rLi,rZ)
  p09 = pPRiLng(p03,rLi,rZ)
'----------------- Mehrweite aufteilen
  rMw = (rG(2)/2+rX(2))
&       -(rG(4)/2+rX(3))
                  'auf halbes Erzeugnis
  rSn = 3/6*rMw       'Anteil in die SN
  rHr = 2/6*rMw       'Anteil in den HR
  rVr = 1/6*rMw       'Anteil in den VR
'--------- SN an der Taille einstellen
  p08 = pPRiLng(p08,rOb,rX(5))
  p10 = pPRiLng(p08,rRe,rSn/2)
  p11 = pPRiLng(p08,rLi,rSn/2)
'----------------------- Abhäher im HR
  rZ  = rAbstPP(p01,p10)/2
  p12 = pPRiLng(p01,rLi,rZ)
  p12 = pPRiLng(p12,rOb,rX(5)/4)
  p13 = pPRiLng(p12,rRe,rHr/2)
  p14 = pPRiLng(p12,rLi,rHr/2)
'----------------------- Abhäher im VR
  rZ  = rAbstPP(p11,p05)*2/3
  p15 = pPRiLng(p05,rRe,rZ)
  p15 = pPRiLng(p15,rOb,rX(5)/2)
  p16 = pPRiLng(p15,rRe,rVr/2)
  p17 = pPRiLng(p15,rLi,rVr/2)
'------------------ Ausgabe der Punkte
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17)
'------------------ Ausgabe der Linien
  AusQ(p01+p03)
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
  End Program
'*************************************
```

3.Teilschritt: Abnäher einzeichnen (Bild 18-16)

von	bis	Richtg	Abstand
12	12a	Lot	Lot von p12 auf Strecke p02⇔p07
12a	12b	⇑	X7
15	15a	⇓	X6
			Abnäher einzeichnen

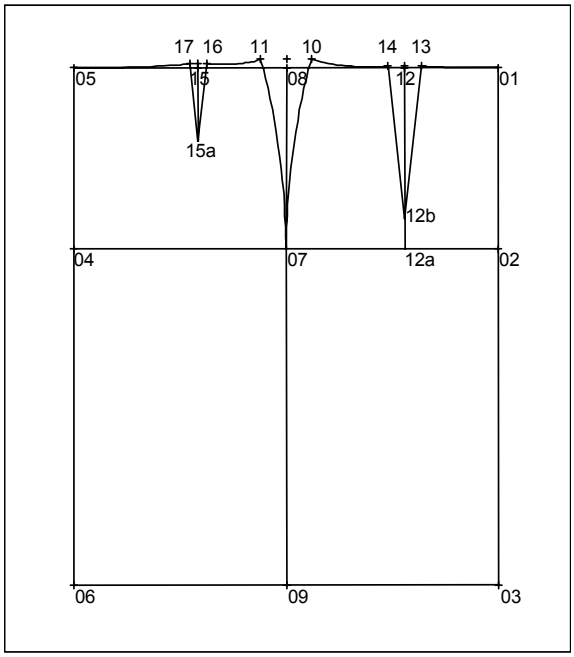


Bild 18-16

```
'*************************************
  Program Main()
'-------------------------------------
  lVar
  nVar
  rVar rZ,rMw,rSn,rHr,rVr
  pVar p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a
  sVar sZ
  qVar
  tVar
  cVar
'-------------------------------------
  lCon
  nCon
  rCon rRe=0,rOb=90,rLi=180,rUn=270
  tCon
'-------------- Definition der X-Werte
  ... wie vorher ...
'----------------------- Abnäher im VR
  rZ  = rAbstPP(p11,p05)*2/3
  p15 = pPRiLng(p05,rRe,rZ)
  p15 = pPRiLng(p15,rOb,rX(5)/2)
  p16 = pPRiLng(p15,rRe,rVr/2)
  p17 = pPRiLng(p15,rLi,rVr/2)
'------------------- Abnäherspitze HR
  sZ  = sPP(p02,p07)
  p12a= pLotPS(p12,sZ)
  p12b= pPRiLng(p12a,rOb,rX(7))
'------------------- Abnäherspitze VR
  p15a= pPRiLng(p15,rUn,rX(6))
'------------------ Ausgabe der Punkte
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a)
'------------------ Ausgabe der Linien
  AusQ(p01+p03)
```

```
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
  AusQ(p12b+p13)
  AusQ(p12b+p14)
  AusQ(p15a+p16)
  AusQ(p15a+p17)
'-----------------------------------
  End Program
'***********************************
```

Die Kurvenvariante Kreisbogenkurve

Dem Kurventyp Kreisbogenkurve liegen verzerrte, entartete Kreisbögen zugrunde. Ein wesentlicher Unterschied zum Kurventyp Spline ist, daß eine Kreisbogenkurve keine Wendepunkte (Bild 18-17) annehmen kann. Eine Kurvenform gemäß Bild 18-17 kann nur mit dem Kurventyp Spline konstruiert werden.

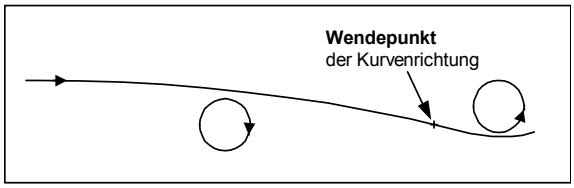


Bild 18-17

Für den Kurventyp Kreisbogenkurven gibt es drei Definitionsvarianten, die bei vergleichbaren Parametern auch die gleiche Kurvenform liefern. ***Kreisbogenkurven ergeben relativ gering ausgeformte Kurven. Sie sind besonders für Hüft- und Taillenkurven geeignet. Sollte die Kurvenform mit einer der Kreisbogenvarianten nicht befriedigend sein, so ist die einzige Alternative eine Kurve vom Typ Spline.*** Die Kurve in Bild 18-18 wurde jeweils mit einer der drei Definitionsvarianten konstruiert:

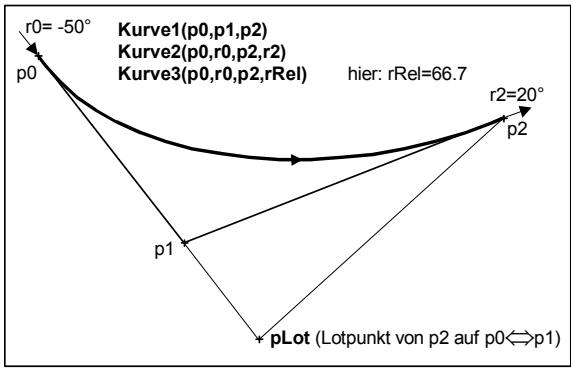


Bild 18-18

Kurve1(pA,pR,pE)
Die Kurve wird von `pA` nach `pE` gebildet. Als Parameter sind

- der Anfangspunkt `pA`,
- der Richtpunkt `pR` und
- der Endpunkt `pE` anzugeben.

Mit dem Richtpunkt `pR` werden die Richtungen der Kurve in `pA` und `pE` bestimmt. In `pA` hat die Kurve die Richtung `pA⇒pR` und in `pE` hat sie die Richtung `pR⇒pE`. Die Kurve schmiegt sich dadurch an die Linien `pA⇒pR` und `pR⇒pE` an.

Kurve2(pA,rA,pE,rE)
Die Kurve wird von `pA` nach `pE` gebildet. Als Parameter sind

- der Anfangspunkt `pA`,
- die Richtung im Anfangspunkt `rA`,
- der Endpunkt `pE` und
- die Richtung im Endpunkt `rE` anzugeben.

Mit den Richtungen im Anfangs- und Endpunkt ergibt sich der Richtpunkt der ersten Definitionsvariante.

Kurve3(pA,rA,pE,rRel[,rE])
Die Kurve wird von `pA` nach `pE` gebildet. Als Parameter sind

- der Anfangspunkt `pA`,
- die Richtung im Anfangspunkt `rA`,
- der Endpunkt `pE`,
- ein Relativwert für die Kurvenform `rRel` und
- optional als Rückgabewert die Richtung im Endpunkt `rE` anzugeben.

Die Richtung der Kurve im Endpunkt berechnet sich aus dem Relativwert durch folgende Regel:
Vom Endpunkt wird das Lot auf die Strecke vom Anfangspunkt mit der Anfangsrichtung gefällt. Der Abstand `pA⇔`Richtpunkt (analog dem Typ Kurve1) berechnet sich aus `rRel/100*`Abstand `pA⇔`Lotpunkt.
Mit dem Wert `rRel` wird indirekt die Richtung im Endpunkt eingestellt. Die Kurvenform kann damit sehr feinfühlig geändert werden. Sie ist jedoch ungeeignet, wenn im Anfangs- und Endpunkt vorgegebene Richtungen einzuhalten sind.

4.Teilschritt: Seitennaht und Taillenlinien auszeichnen

Für die Konstruktion der Seitennaht wird die Konstruktionsvariante „Kurve3“ benutzt, da die Richtung der Seitennaht an der Taille noch beliebig ist. Der Hüftbogen kann dadurch mit dem Parameter `rRel` in einer optimalen Form eingestellt werden. Der Hüftbogen im Vorderrock entsteht mit

```
qSn_vr = qKurve3(p07,rOb,p11,60)
```

Verändern Sie den Zahlenwert `60` in Schritten von `5` und stellen nach  ,  (auch  mit mehreren Größen) und  einen schönen Hüftbogen ein.

Nach Spiegeln an p07⇔p08 entsteht der Hüftbogen im Hinterrock.

```
qSn_hr = qSn_vr
Spgl(sPP(p07,p08):qSn_hr)
```

Vor dem Spiegeln wird zunächst auf die neue Kurvenvariable qSn_hr umgespeichert. Mit

```
Spgl(sPP(p07,p08):qSn_hr=qSn_vr)
```

würde direkt in der Spiegelfunktion umgespeichert.

Die Taillenlinien sollen jeweils rechtwinklig zur Seitennaht, zu den Abnäherlinien sowie zur Vorderen und Hinteren Mitte verlaufen. Bevor die jeweiligen Abschnitte der Taillenlinie gebildet werden können, muß die Richtung der Taillenlinie im Anfangs-/ Endpunkt berechnet werden.

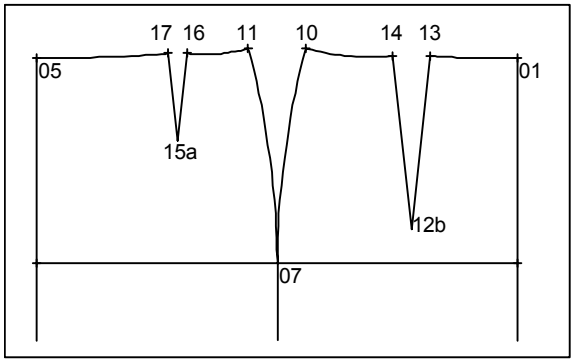


Bild 18-19

Die Taillenlinie ab Hinterer Mitte beginnt am Punkt p01 mit waagerechter Richtung nach links (180°) und endet in p13 senkrecht auf der Richtung p12b⇒p13 (Bild 18-19). Mit rRiPP(p12b,p13) wird die Richtung berechnet und mit +90 um 90° im mathematisch positiven Drehsinn gedreht.

```
rRi13  = rRiPP(p12b,p13)+90
```

Das Teilstück der Taillenlinie ab Hinterer Mitte kann dann konstruiert werden mit

```
qTa_hr1= qKurve2(p01,rLi,p13,rRi13)
```

Die Taillenlinie soll senkrecht in die Seitennaht einlaufen. Mit rRiQend(qSn_hr)wird dazu zunächst die Richtung im Endpunkt der Seitennaht berechnet. Mit +90 wird die Richtung wiederum um 90° in mathematisch positiver Richtung gedreht und ergibt die gesuchte Richtung der Taillenlinie im Endpunkt rRi10.

Das Teilstück der Taillenlinie ab Abnäher HR zur Seitennaht kann dann konstruiert werden mit

```
qTa_hr2=qKurve2(p14,rRi14,p10,rRi10)
```

Analog schließt sich die Konstruktion der Taillenlinien im Vorderrock an.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar rZ,rMw,rSn,rHr,rVr,
&      rRi13,rRi14,rRi10,
&      rRi17,rRi16,rRi11
  pVar p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a
  sVar sZ
  qVar qSn_vr,qSn_hr,
&      qTa_hr1,qTa_hr2,
&      qTa_vr1,qTa_vr2
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRe=0,rOb=90,rLi=180,rUn=270
  tCon
'-------------- Definition der X-Werte
  ... wie vorher ...
'-------------------- Abnäherspitze VR
  p15a= pPRiLng(p15,rUn,rX(6))
'----------- SN auszeichnen & spiegeln
  qSn_vr = qKurve3(p07,rOb,p11,60)
  qSn_hr = qSn_vr
  Spgl(sPP(p07,p08):qSn_hr)
'--------- Taillenlinie HR auszeichnen
  rRi13  = rRiPP(p12b,p13)+90
  qTa_hr1= qKurve2(p01,rLi,p13,rRi13)
  rRi14  = rRiPP(p12b,p14)+90
  rRi10  = rRiQend(qSn_hr)+90
  qTa_hr2=qKurve2(p14,rRi14,p10,rRi10)
'--------- Taillenlinie VR auszeichnen
  rRi17  = rRiPP(p15a,p17)+90
  qTa_vr1= qKurve2(p05,rRe,p17,rRi17)
  rRi16  = rRiPP(p15a,p16)+90
  rRi11  = rRiQend(qSn_vr)+90
  qTa_vr2=qKurve2(p16,rRi16,p11,rRi11)
'------------------ Ausgabe der Punkte
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a)
'------------------ Ausgabe der Linien
  AusQ(p01+p03)
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
  AusQ(p12b+p13)
  AusQ(p12b+p14)
  AusQ(p15a+p16)
  AusQ(p15a+p17)
  AusQ(qSn_vr,qSn_hr,qTa_vr1,qTa_vr2,
&      qTa_hr1,qTa_hr2)
'------------------------------------
  End Program
'************************************
```

18.7 Allgemeine Hinweise

Mit den bisher vorgestellten Befehlen und Funktionen können bereits die Mehrzahl aller Grundkonstruktionen in ein Fachsprachenprogramm umgesetzt werden. Eine Übersicht über alle verfügbaren Funktionen befindet sich in der Grafis-Hilfe.

Wie finde ich die passende Funktion ?

Ermitteln Sie zunächst, von welchem Variablentyp das Ergebnis sein muß. Wird ein Punkt gesucht, dann kommen nur Funktionen in Frage, die mit „p“ beginnen, für Strecken nur die Funktionen mit „s“ und so weiter. In der Regel folgt im Funktionsnamen ein Kürzel für die Ergebnisart

Kürzel	Ergebnisart	Beispiel
Wi	Winkel	rWiSS(s,s)
Ri	Richtung	rRiPP(p,p)
Lng	Gesamtlänge	rLngQ(q)
Tlng	Teillänge	rTlngSP(s,p)
Rlng	relative Länge	rRlngSP(s,p)
Lot	Lot	pLotPS(p,s)
Tang	Tangens	pTangPQ(p,q)
…	…	…

und anschließend die benötigten Parameter als Großbuchstabe.

Was mache ich bei Fehlermeldungen ?

Es sind zwei Fehlerarten zu unterscheiden

- ***Syntaxfehler*** = Fehler in der Schreibkonvention („Rechtschreibfehler“) und
- ***logische Fehler***, die beim Abarbeiten des Programms auftreten.

Syntaxfehler werden beim Compilieren unter Angabe der betreffenden Zeile und eines Hinweises gemeldet. Als Syntaxfehler werden unter anderem das Fehlen öffnender / schließender Klammern, unbekannte Funktionsaufrufe oder falsche Parametertypen in Funktionsaufrufen gemeldet. Syntaxfehler können in der Regel schnell behoben werden.

Ein logischer Fehler liegt vor, wenn das Programm nicht das erwartete Ergebnis liefert. ***Logische Fehler sind leichter zu finden, wenn das Programm in kleinen Teilschritten entwickelt und jeder Teilschritt gründlich getestet wird (auch in kleinen/großen Größen).*** In diesem Fall ist der Fehler im letzten Teilschritt zu suchen. Bei langen Programmen ist es durchaus sinnvoll, die Punkte, Linien und Kurven von einem bestimmten Zwischenstand auszudrucken und zu beschriften, analog den Bildern 18-14, 18-15, 18-16 und 18-19. Zur Fehlersuche hier noch einige Tips:

- Der Wert einer Variablen beliebigen Typs kann mit dem Befehl `VList()` geprüft werden. Mit der Zeile
  ```
  rRi11  = rRiQend(qSn_vr)+90
  VList(rRi11)
  ```
  wird nach dem nächsten  oder  der Wert der Variablen `rRi11` in der Variablen Liste angezeigt.
- Die Zeile, in der eine Variable mit dem aktuellen Wert belegt wurde, ermitteln Sie, indem die Variable in der aktuellen Zeile markiert und dann mit  rückwärts gesucht wird.
- Zur Identifikation eines gesuchten Punktes `pW` geben Sie eine Strecke vom Nullpunkt zum gesuchten Punkt aus: `AusQ(pXY(0,0)+pW)`
- Wird ein Punkt als Schnittpunkt zwischen Kreis und Strecke gebildet, geben Sie temporär den Kreis und die Strecke aus und beobachten dann das Ergebnis auch in kleinen/großen Größen. Mit einem „'“ vor dieser temporären Ausgabe wird die Zeile zur Kommentarzeile.
- Während der Compilierung wird gefragt, ob eine nicht deklarierte Variable neu deklariert werden soll. Prüfen Sie bei jeder Abfrage, ob die Variable tatsächlich neu verwendet wurde oder nur durch einen Schreibfehler „neu entstanden“ ist.
- Nach dem erfolgreichen Compilieren erscheint im unteren Hinweisfeld eine Angabe, welche Variablen unbenutzt sind. Unbenutzte Variablen sind bei ordentlicher Programmierung oft der Hinweis auf eine Verwechslung.

Was ist bei der Freigabe eines Programmes zu berücksichtigen ?

Vor der Freigabe eines Programmes sollte abschließend geprüft werden, ob

- das Programm in allen Größen, auch in extremen kleinen/ großen/ individuellen, fehlerfrei funktioniert.
- alle X-Werte eingebunden sind, richtig verrechnet werden und korrekt kommentiert sind. Eine „Zugabe zum Taillenumfang“ darf nicht als „Zugabe zum halben Taillenumfang“ wirken. Ein positiver Wert bei „Ausstellung Seitennaht“ darf nicht zum Einstellen der Seitennaht führen.
- nur Objekte (Punkte, Linien, Kurven) ausgegeben werden, die nötig sind. Objekte, die der Anwender nicht benötigt, sollten auch nicht ausgegeben werden.
- die Linienlängen korrekt sind. In der Rock-Konstruktion des vorhergehenden Abschnittes sollten beispielsweise die Längen der Seitennähte in Vorder- und Hinterrock verglichen und die Summe der Taillenlinien nachgemessen werden; auch in anderen Größen.

Nach der Freigabe des Programmes ist die Programmdatei *.cpr in die Holen-Liste einzutragen und eine Infomaske zu erstellen.

Weiterhin erstellen Sie eine Dokumentation mit unter anderem folgendem Inhalt:

- ein Ausdruck der Konstruktion, in dem die Objekte beschriftet werden. Es sollten alle verwendeten Objekten ausgegeben werden, auch Hilfspunkte und –linien, die in der freigegebenen Konstruktion nicht erscheinen.
- ein Ausdruck des Programmes,
- das Programm als Datei und
- eine Kopie der Konstruktionsbeschreibung.

Was ist bei der Änderungen / Korrektur eines Programmes zu berücksichtigen ?

Korrekturen in freigegebenen Programmen müssen sehr umsichtig durchgeführt werden, da Modelle, die aus diesem Programm entwickelt wurden, immer wieder darauf zurückgreifen.

Vor allen Änderungen an freigegebenen Programmen, muß der Änderungscode in der Projektoberfläche über Extras | Optionen erhöht werden! Dies gilt insbesondere im Fall von Änderungen bei der Objektausgabe.

Zur Erläuterung soll an dieser Stelle das Protokollprinzip von Grafis erläutert werden. Jeder Ausgabebefehl eines Fachsprachenprogrammes übergibt Objekte (Punkte, Linien) an das Grafis-Protokoll. Die Objekte erhalten in der Reihenfolge ihrer Übergabe eine Pos-Nummer. Die Pos-Nummer ist ein Identifikator für die Objekte des Grafis-Protokolls. Der Ausgabebefehl

```
AusQ(qSaum,qInnenbein,qSchritt)
```

übergibt die Saum-, Innenbein- und Schrittnaht an das Grafis-Protokoll, das diesen Linien die laufenden Pos-Nummern 1, 2 und 3 zuordnet. Wird jetzt im Protokollbetrieb eine Parallele an die Innenbeinnaht konstruiert, bezieht sich dieser Protokollschritt auf das Objekt mit Pos-Nummer 2.

Wird später am Fachsprachenprogramm die Ausgabezeile geändert in

```
AusQ(qInnenbein,qSaum,qSchritt)
```

und im Modell ein Probelauf durchgeführt, erscheint die Parallele statt an der Innenbeinnaht an der Saumlinie. Diese Änderung führt nur bei Modellen zu Fehlern, die mit dem Fachsprachenprogramm vor der Änderung entwickelt wurden.

Die Objekte müssen immer in gleicher Reihenfolge bei gleicher Objektart ausgegeben werden, unabhängig von der Größe und von den X-Werten. Ausgabeanweisungen innerhalb von IF-ENDIF-Strukturen sollten daher vermieden werden.

Auch vor der Korrektur an Zahlen oder Formeln sollte der Änderungscode erhöht werden. Ein Anwender Ihres Programmes könnte die Grundform bereits durch Konstruktionsschritte korrigiert haben. Diese Konstruktionsschritte werden später auch mit dem geänderten Programm ausgeführt.