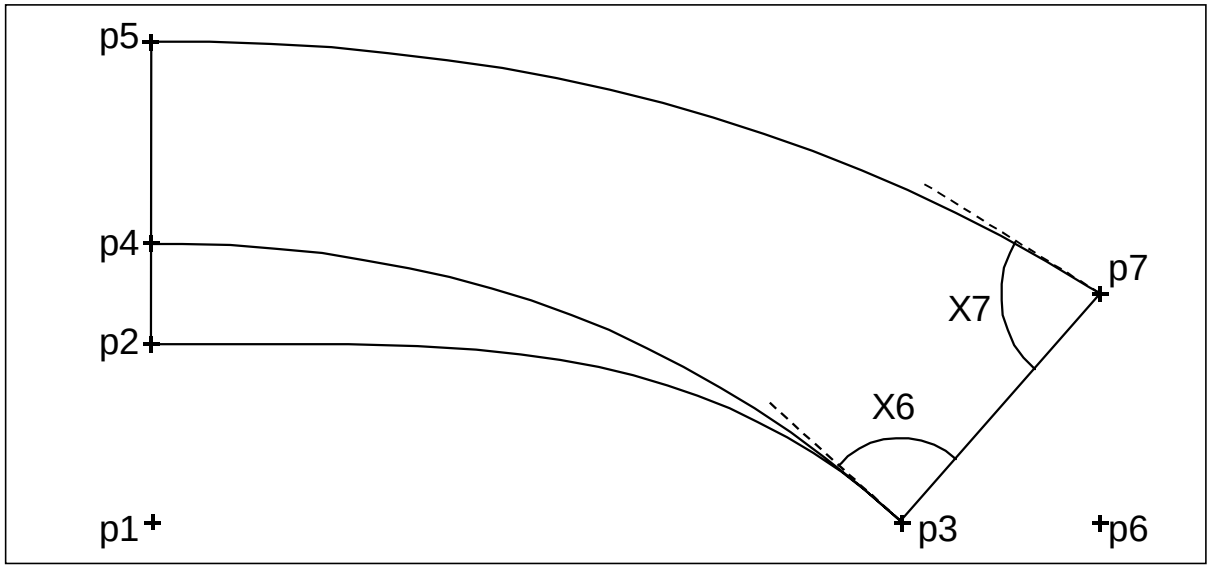
Chapter 18 „Programming Language I“

Content

The Grafis Programming Language is used for development of basic blocks and construction components. The individual steps for generation of a basic block are entered as text. Basic blocks should be developed in the programming language if a company-specific fit or a complete component solution is to be developed. It is to be considered that the programming language is an abstract form of pattern development. Apart from excellent pattern construction knowledge and experience in the application of Grafis, considerable familiarisation time is required. For development of production patterns from prepared, adjusted basic shapes, the functions of the Grafis dialogue are the more appropriate tools.

```
'************************************
  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 rRi=0,rLe=180,rUp=90,rDo=270
  rCon rClLng=150
  tCon
'------------------- x value definitions
  XTitel("collar band")
  Defx(1,"raise CB",35)
  Defx(2,"collar fall width",20)
  Defx(3,"collar width CB",40)
  Defx(4,"collar point(x) to p3",40)
  Defx(5,"collar point(y) to p3",45)
  Defx(6,"ang neck+foldline in p3",90)
  Defx(7,"ang collar edge in p7",80)
'--------------------------- points at CB
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rUp,rX(2))
  p5 = pPRiLng(p4,rUp,rX(3))
'--------------------corner point p3 (CF)
  p3 = pXY(rClLng,0)
  p6 = pPRiLng(p3,rRi,rX(4))
  p7 = pPRiLng(p6,rUp,rX(5))
'------------------------------ neck line
  rWi3  = rWiPPP(p6,p3,p7)
  rWi3  = rWi3+rX(6)
  q1 = qSpline(p3,rWi3,p2,rLe)
  q2 = qSpline(p3,rWi3,p4,rLe)
'---------------------------- collar edge
  rWi7  = rRiPP(p7,p3)-rX(7)
  q3 = qSpline(p7,rWi7,p5,rLe)
'------------------ output points + lines
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(p2+p5)
  AusQ(p3+p7)
  AusQ(q1,q2,q3)
'------------------------------------
  End Program
'************************************
```



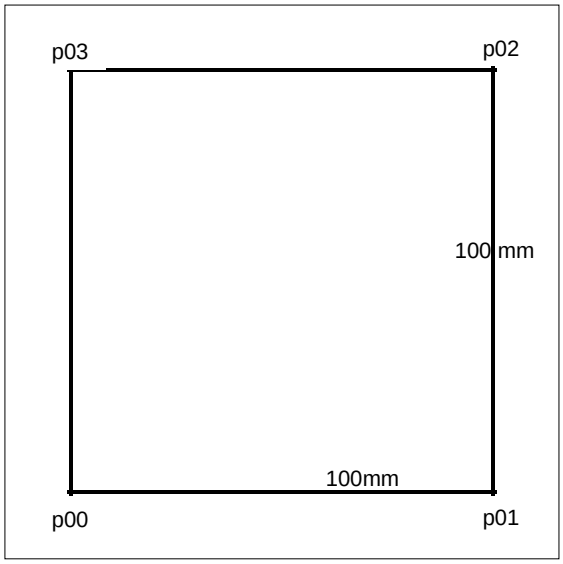
18.1 A simple program: square

Introductory notes

From version 8 onwards, the new programming language is part of the Grafis package. It is a compiler-oriented programming language. The programs are no longer processed, interpretively but in machine-similar code. The processing of the programming language program is therefore, significantly faster. The new programming language contains many tools common in other programming languages; adapted to the requirements of pattern construction. The use of sub-programs is also possible, so that repeated calculations can be saved as sub-programs. The clarity, in particular for the text display and the syntax check have become better and more accurate compared with the previous programming language. Users who have worked with the previous programming language will, at first, find the current language more complex. With further observation, it will become clear that the new techniques allow for a shorter and more transparent programming.

Square

In the first project a square (Picture 18-1) is to be constructed which will later be transformed into the shape of a house as in Picture 18-2.



Picture 18-1

Start the new project „square“ as follows:

⇒ *Extras | New Compiler*
⇒ *Project | New...*
⇒ Project name: square
⇒ enter 9-digit code for your name (according to Picture 18-4)
⇒ enter 2-digit code for the product group, e.g. „TB“ for Textbook exercises
⇒ With <OK>, the project is created.

The structure of the program `Main()` has been set up in the edit window (Picture 18-5). Edit the program as follows:

```
'************************************
Program Main()
'- Program: square
'- Declaration lines
lVar
nVar
rVar
pVar
sVar
qVar
tVar
cVar
'- Constants
lCon
nCon
rCon
tCon
'- Allocations / Instructions
'- Program end
End Program
'************************************
```

After a first compilation with the button  or <F4>, the program thus far has been automatically formatted:

```
'************************************
  Program Main()
'-------------------- Program: square
'------------------- Declaration lines
  lVar
  nVar
  rVar
  pVar
  sVar
  qVar
  tVar
  cVar
'--------------------------- Constants
  lCon
  nCon
  rCon
  tCon
'---------- Allocations / Instructions
'------------------------- Program end
  End Program
'************************************
```

The construction of a square can be carried out as follows:

⇒ assign points p00 to p03
⇒ output points p00 to p03
⇒ output connecting lines between the points

The following allocations/instructions lead to the result:

```
'--------- Allocations / Instructions
  p00= pXY(0,0)
  p01= pXY(100,0)
  p02= pXY(100,100)
  p03= pXY(0,100)
```

The function `pXY()` constructs a point from the x and y co-ordinates to be entered. The x co-ordinate of point `p01` has the value 100 and the y co-ordinate the value 0.

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

With the instruction `AusP` the listed points are output to the screen. Without this instruction line, the points are assigned in the program but are not displayed on the screen.

After entry of the five lines, compile the program with  and then test it in the test run with  . Clicking  switches to the Grafis screen and the points of the square appear. The Grafis screen can be closed with the right mouse button.

With the instruction `AusQ`, individual lines, curves or polygons are output. Select „Inner Fcn“ in the variable list and click on `AusQ`. A help text for the selected function appears below the edit window. The connecting lines are output as one line, linked across the corners with

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

with the lines

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

the connecting lines are output as individual lines from corner to corner. The line does not have to be created as a variable, first. The calculation can ensue directly in the function. The lines

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

lead to the same result as the line

```
AusQ(sPP(p00,p01))  or as
AusQ(p00+p01)
```

In the first option, the line is created as variable `s1`, first and then output. In the second option, the line is created directly in the instruction. The function `sPP(p00,p01)` creates a line variable as a connection between two points to be entered.

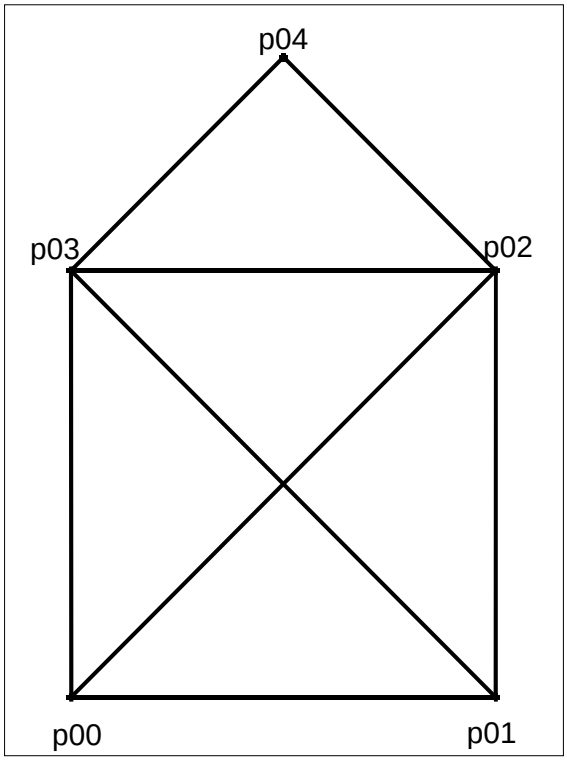
Variation „house“

The square can be transformed into a house (Picture 18-2). An additional point `p04` is to be created as the apex of the roof. The lines are to be output, continuously.

The following allocations/instructions lead to the result:

```
'------------ Allocations / Instruction
 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)
```

Save the project via *Project* | *Save* and quit the project user interface via *Project* | *End*.



Picture 18-2

18.2 Data basis and user interface

Data basis

The development of a basic block ensues in a so-called project. A project consists of

- the modules in clear text,
- the object code for the project and
- the program in the processable programming language as the result.

The project itself is saved as a directory. This directory contains the modules in clear text and the object code. The processable programming language program is saved directly in the \PROG directory of the respective construction system. Picture 18-3 contains a detailed overview.

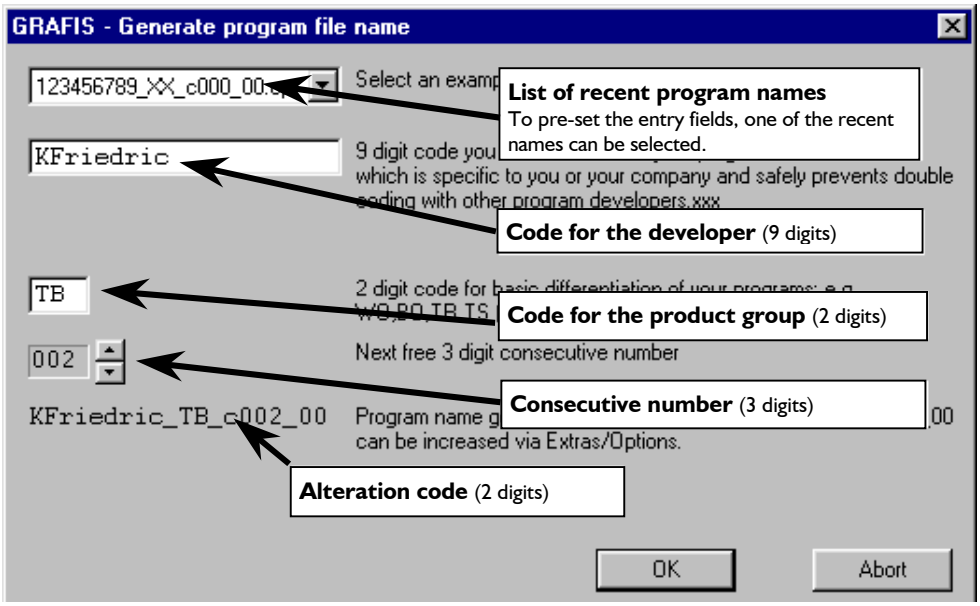
To ***copy*** or ***duplicate***, it is sufficient to copy the complete directory \Grafis\Module\[*co-system*]\[*Project name*]. All data belonging to the project are included. To ***pass on*** a tested and released programming language program, it is sufficient to copy the *.cpr file from the directory \Grafis\[*co-system*]\PROG.

Grafis	
|--Basis_D	
| |--PROG	*.cpr files (example: KFriedric_LA_c001_00.cpr) These processable programming language programs contain all information required for processing of the module. To be able to run styles in which this module was used on other computers, this file must be also be transferred.
|-- ...	
|--Module	
|--Basis_D	
|--\[*Project name*]	Each project obtains its own directory under \Grafis\Module\[co-system]\ in which all files of the project are saved. The project „Bodice 01“ in the „Optimass“ construction system is saved as directory „\Grafis\Module\Basis_D\Bodice 01“. The project directory contains the following files: ➢ Modul.ini Initialisation file for the project ➢ Main.qpr Source code of the main program « Main » in RTF format ➢ Main.qpt Source code of the main program « Main » as ASCI file ➢ Main.opr Object code of the main program « Main » … and further files *.qpr, *.qpt and *.opr for possible sub-programs.

Picture 18-3

Start new project

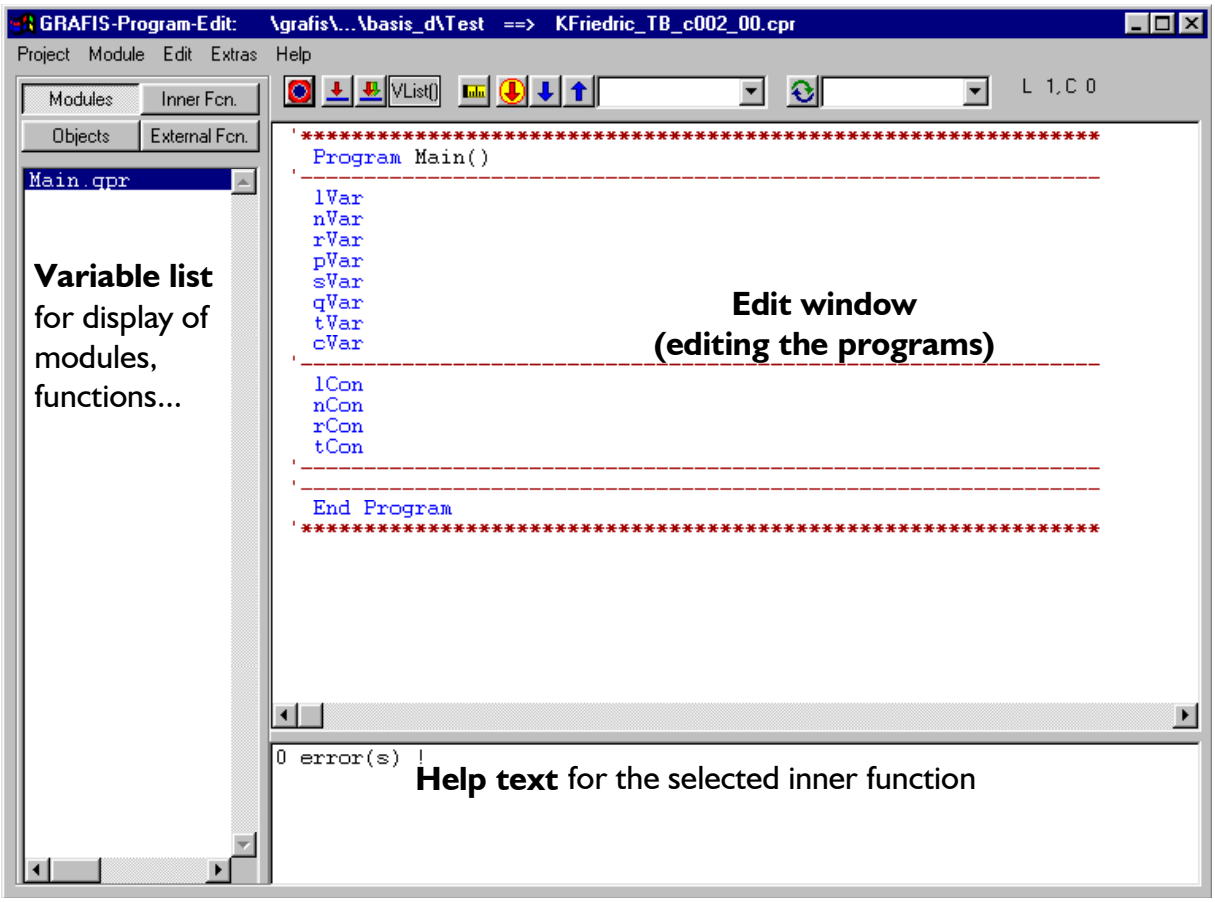
To develop a new basic shape, start Grafis with the required construction system.
Programming with the programming language should ensue in a new style so that important styles are not accidentally reset or overwritten.
A new project is started via the pull-down menu ***Extras*** | ***New Compiler*** and then, ***Project*** | ***New...***. The project name must not contain special characters (e.g. „!+-ßäöü). A suitable name would be for example „Bodice 01“.
In addition to the project name (= directory for development files, Picture 18-3), a name for the processable programming language program is to be given. The window „generate program file name“ (Picture 18-4) opens.



Picture 18-4

Name of the program file

The name of the program file has a given, pre-set length of 20 characters. It has been lengthened by 12 characters from version 7 or earlier. Thus, program can be differentiated, more easily and use of the same name can be prevented.



Picture 18-5

The name consists of the following codes:

KFriedric_LA_c000_00
|-- 9-digit ***code for the developer***

Examples:

KFriedric	for Kerstin Friedrich
BBachmann	for Betty Bachmann
FSBeautyW	for Fred Smith, employee at the company BeautyWear
RWRollerD	for Roland Williams, working for the company Roller Design

KFriedric_LA_c000_00
|-- 2-digit ***code for the product group***

Examples:

LA	for ladies wear
ME	for menswear
CH	for childrenswear

KFriedric_LA_c000_00
|
3-digit ***consecutive number***

KFriedric_LA_c000_00
|
2-digit ***alteration code***

Only for the first project of a developer, the window is to be filled in, completely. This includes in particular the developer code and the code for the product group. The consecutive number is suggested by Grafis, automatically and should only be changed if necessary.

The alteration code must be increased, if an already delivered / applied program is to be edited. The alteration code can be increased in the project user interface with *Extras | Option*.

For further projects, a relevant name can be selected from the list of recent project names and then, adapted if necessary.

The project user interface

The three largest areas of the project user interface (Picture 18-5) are

- the edit window for entry of the programs,
- the variable list for display of available modules, functions, ... and
- a window with help texts about the selected inner function.

The program **Main** is started, immediately as an „empty“ program in the required structure, see Picture 18-5. All keyboard entries and the buttons above the edit window apply only to the program text. The variable list and the window with help texts are display areas without entry option.

Edit functions

The following can be used for edit:

←→↑↓	move cursor left, right, up, down
↵ (Enter)	next line
Home	cursor at beginning of line
End	cursor at end of line
Delete	delete next/selected character
Ctrl+Home	cursor at beginning of program
Ctrl+End	cursor at end of program
right mouse button	open the context menu = pull-down menu *Edit*

Analogous to other editors, selecting one or more lines is possible by clicking at the edge of the line. Selected lines are moved with the left mouse button pressed or copied with additional pressed Ctrl key. Further edit functions can be found in the *Edit* pull-down menu which can also be opened as context menu with the right mouse button.

Buttons compile and test

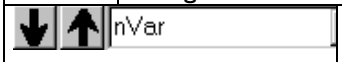
The functions most commonly used during program development can be found on the buttons above the edit window.

The fist block of buttons contains functions for compiling and testing of programs:

	compile (syntax check and first translation)
	create & test the program in test run
	create & test the program with grading
	show / hide VList entries; If the VList() button is switched on, the program is stopped at the VList stops during create & test. The values of the entered variable appear in the variable list.
	Display of the result of the last create & test of the program

Buttons search and replace

The second block of buttons contains functions to search and replace characters:

	select all characters in the program identical to the characters in the field on the right
	search the characters in the right field
	replace the selected characters with the term to the right of 

Select the search term, e.g. p01 in the program. It automatically appears in the search field to the right of  , in which the term could also be entered.

Clicking on  or  selects the next character string found.

To replace e.g. p01 with p02, the following procedure is recommended:

⇒ Select the search term in the program,

⇒ enter the replace term (here: p02) to the right of  ,

⇒ Click on  if the selected term is to be replaced with the replace term.

The variable list

The buttons „Module“, „Inner Fcn“ etc. above the variable list work like file card tags. After having clicked one of these buttons, the following is displayed in the variable list:

Button	**Content of the variable list**
Module	all program modules (*.qpt files) for the current project
Inner Fcn.	all inner functions
External Fcn.	all external functions of the current project (of all program modules)
Objects	the output objects (points + lines) with the data (o-object number, ty-object type, po-pos-number)

Automatic formatting

Each compiling started with  or <F4>, automatically formats the program text as well as checking the syntax. During automatic formatting of the program text, instructions are highlighted in blue, comments in green.

As a rule, instructions are indented by two characters; in loops by another two characters. The first letter of the variable name must be in lower case and the second letter must be in upper case.

To simplify entry of comment lines, the following rules apply:

1. ***If a single inverted comma is entered in the first column, the character following the inverted comma will fill the whole line.***

```
'-  becomes...
'-----------------------------------
'*  becomes...
'***********************************
```

2. ***If a single inverted comma is entered in the second or a following column, the text is aligned right.***

```
'---Initialisation  becomes...
                        '---Initialisation
   'output points    becomes...
                           'output points
```

3. ***One single inverted comma followed by a space result in the text being unchanged.***

```
' ---Initialisation    remains...
' ---Initialisation
```

```
' output points    remains...
' output points
```

Comments can also be entered in a program line to the right of the instruction. Here, the second and third rule apply.

Note: Test the automatic formatting and the rules for comments with the still „empty“ program Main().

18.3 Rules for programming

Ground rules

- A program is set up ***in lines***.
- Each line contains an ***allocation*** or an ***instruction***.
- The ***line width*** should not exceed 64 characters.
- ***Upper/lower case, spacing*** and possible ***indents at the beginning of the line*** are formatted by Grafis, automatically during compiling.
- The inverted comma ' indicates that the following text is a ***comment*** which is not processed.
- The character „&“ in the first column indicates that the line is a ***continuation line***.
- The module Main() must be contained in each ***project*** and contain the program Main().
- Each project contains exactly one ***program*** with the name Main(). This program is processed, first after calling.
- Each project can contain any number of ***inner functions*** and any number of ***external functions***. The inner functions are part of the Grafis package. External functions are programmed by the user.
- Each program begins with „`Program Main()`“ and ends with „`End Program`“.
- Each function begins with „`Function xXxx()`“ and ends with „`End Function`“.

Variables

In Grafis, variables of different types are used. The variable name can consist of up to 64 characters, where the first character indicates the variable type. Variables can only be used after they have been declared at the beginning of the program or the function. During declaration of variables memory is reserved and set to nil. The variable is then, available until the end or the function or program.

The following ***variable types*** are available:

Type	Explanation	Example
lXxx	logical variable, which can adapt the value True or False	lQuery
nXxx	number (whole number) with a value between $-2*10^9$ and $+2*10^9$ (2.000.000.000)	nNum
rXxx	real number, accurate to 6 decimals	r01
pXxx	point with x and y co-ordinates	pSIn
sXxx	line with start and end point	sHem
qXxx	polygon / curve / line sequence (q stands for queue)	qArm
tXxx	text with up to 10.000 characters	tHelp
cXxx	container	cBox

All variables used must be declared in the program / function header. The ***declaration lines*** begin with `lVar` for logical variables, with `nVar` for whole number variables etc. For each variable type, a number of declaration lines may be entered.
Example:

```
nVar nIs1,nIs2,nIndex
```

The ***values of the variables*** are set via allocation lines.

The variable types l (logical), n (whole number), r (real) and t (text) can also be defined as ***constants*** at the beginning of the program/function. The definition lines for constants start with `lCon` for logical variables, with `nCon` for whole number variables etc.
Example:

```
nCon nIs1=1,nIs2=2
```

Constants may not be declared as variables at the same time.

Variables and constants apply only within the program or function in which they were declared.

All new variables used during programming, are automatically entered into the declaration lines during compiling, provided a minimum of one (also empty) declaration line is available for this variable type.

Allocation

The character „=“ stands for allocation in all programming languages. As opposed to an equation in mathematics, here it means:

The value of the term to the right of „=“ is assigned with the variable to the left of „=“. Therefore, to the left of „=“ must be a variable.

The following line would be incorrect as a mathematical equation. As an allocation for programming it has the following meaning:

```
nNumber=nNumber+2
```

The variable `nNumber` must first be declared in the program header. When processing this line, the term to the right of „=“ is calculated, first and then, assigned to the variable on the left of „=“. If `nNumber` has the value 5 before processing the line, the result of the term to the right of „=“ has the value 7. After processing the line, the value of `nNumber` has been increased by 2.

Instructions

With instructions, operations are carried out during programming which can effect one or more objects. In Grafis, instructions for move, rotate, flip or screen output are available for one or more objects. Instructions begin with a command word as opposed to allocations.

Inner functions

Inner functions are prepared functions which are part of the Grafis package. Inner functions which deliver a value are used in calculations. Inner functions which carry out an operation are used in instruction lines. The range of inner functions is sufficient to program all steps common in pattern construction.

After having opened a project and having clicked the button „Inner Fcn.“ (above the variable list), all inner functions are displayed in the variable list. Clicking on a function highlights it. At the same time, a help text on the selected function appears below the edit window. Double-click on the function inserts it into the program.

The first character of the name of a inner function which delivers a value, is a code for the type of the delivered value. The types are identical to the variable types. The function `rG()` delivers a real value. The function `pPRiLng()` delivers a point.

Allocation of values

Declared variables are allocated a value with the following instructions:

Logical variable

```
lQuery1=False
lQuery2=True
```

Number / whole number variable

```
nIndex=1
```

Mathematical calculations (addition, subtraction, multiplication, division) of numbers, whole number / real variables and whole number / real functions are possible. If the term on the right of the „=“ does not deliver a whole number value, it is rounded to the nearest whole number.

Real variable

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

Here, analogous to whole number variables, mathematical calculations are also possible. However, the result is not rounded.

Point

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

Points are set with the use of the inner functions. Copying a point with `p31=p30` is also possible.

Line

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

... analogous point

In addition, with `sHem=-sHem` the orientation of the line can be altered.

Curve

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

... analogous line

Text

```
tInfo=„My first program.“
```

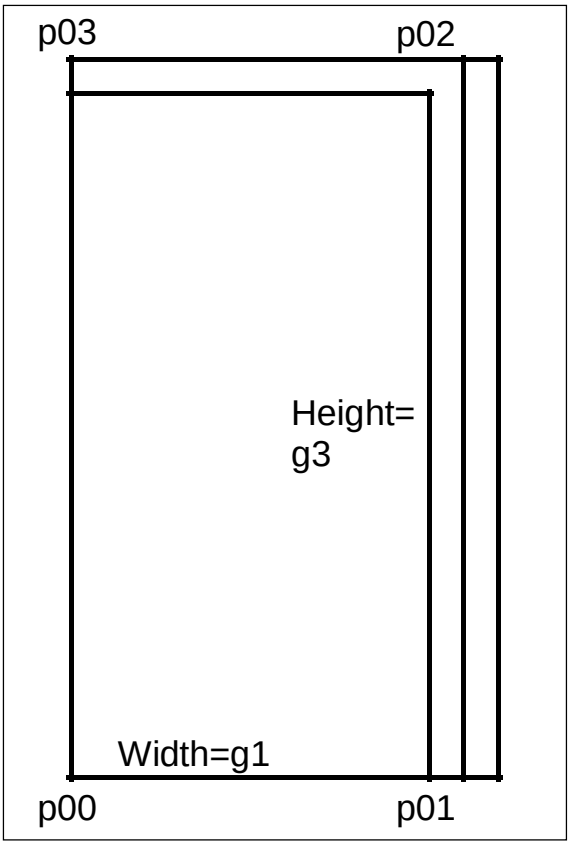
The text must always be placed between quotation marks.

18.4 Program: gradeable rectangle

Gradeable rectangle

A gradeable rectangle (width: bust girth, height: body height) is to be constructed (Picture 18-6). The following lines generate the result:

```
'************************************
  Program Main()
  '-------- Program: gradeable rectangle
  '------------------- declaration lines
  lVar
  nVar
  rVar rWidth,rHeight
  pVar p00,p01,p02,p03
  sVar
  qVar
  tVar
  cVar
'--------------------------- constants
  lCon
  nCon
  rCon rRi=0,rLe=180,rUp=90,rDo=270
  tCon
'-----------allocations / instructions
  p00= pXY(0,0)
  rWidth = rG(1)
  rHeight= rG(3)
  p01= pPRiLng(p00,rRi,rWidth)
  p02= pPRiLng(p01,rUp,rHeight)
  p03= pPRiLng(p02,rLe,rWidth)
  AusP(p00,p01,p02,p03)
  AusQ(p00+p01,p01+p02)
  AusQ(p02+p03,p03+p00)
' ------------------------ Program end
  End Program
'************************************
```



Picture 18-6

The declaration of the new variables `rWidth` and `rHeight` is carried out, automatically during first compiling. It is not necessary to enter the variables in the declaration lines, yourself.

Directions

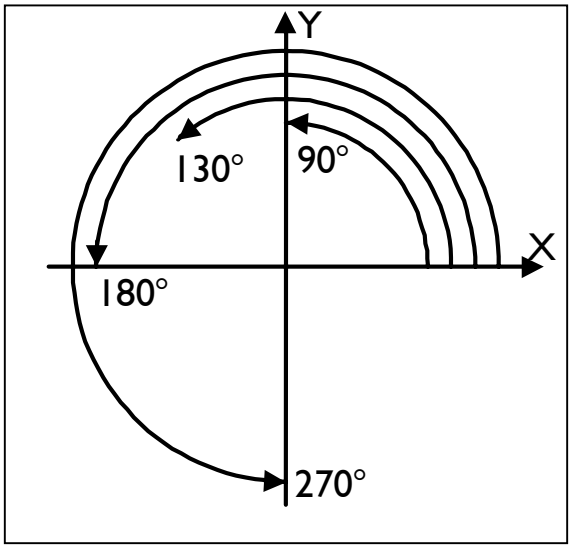
Directions are given in angle degrees. A point is set to the left if the value of the direction is 180. All angle entries relate to the positive x axis and are anti-clockwise (Picture 18-7). If you find it difficult to imagine a direction in angle degrees, you should work with direction constants, e.g. `rRi=0, rLe=180, rUp=90, rDo=270`; see also program example „gradeable rectangle„.

The functions `pXY()`, `rG()`, `pRiLng()`

The lines in the block „allocations / instructions“ have the following significance:

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

The variable `p00` is allocated with the inner function `pXY()`. The parameters after the open bracket state the x and y co-ordinates of the point. In this



Picture 18-7

case both co-ordinates are 0. Thus, `p00` is the zero point.

```
rWidth = rG(1)
rHeight= rG(3)
```

The new variables `rWidth` and `rHeight` are allocated with values which are calculated with the inner function `rG(n)`. ***The function `rG(n)` calculates the nth size value of the measurement chart.*** With `rG(1)` the first value of the measurement chart (in Damen_5 and Basis_D: bust girth) and with `rG(3)` the third value of the measurement chart (in Damen_5 and Basis_D: body height) are transferred.

```
p01= pPRiLng(p00,rRi,rWidth)
```

The new point `p01` is allocated with the result of `pPRiLng(p00,rRi,rWidth)`. `pPRiLng()` calculates a new point which is to be positioned in direction `rRi` and at distance `rWidth` from point `p00`.

Instead of variables, functions of the same type and for real/whole number parameter, numbers can be entered in the parameter list of inner functions. Thus, the following lines have the same meaning:

```
p01= pPRiLng(p00,rRi,rWidth)
p01= pPRiLng(pXY(0,0),rRi,rG(1))
p01= pPRiLng(p00,0,rWidth)
```

In the lines

```
p02= pPRiLng(p01,rUp,rHeight)
p03= pPRiLng(p02,rLe,rWidth)
```

point `p02` is calculated from `p01`, upwards with the distance of the rectangle height. The same applies to `p03`.

Output of objects, test program

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

outputs the corner points of the rectangle to the screen.

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

outputs the connecting lines between the corner points to the screen as individual lines.

After having entered the program lines and having compiled the program  , the program is to be created & tested  . The result appears after  . The programming user interface reopens with the right mouse button. With  instead of  , the program is tested not just in the base size but in all sizes in the size table and displayed after clicking  . For the height of the rectangle to change, short / long sizes or individual sizes must be entered in the size table.

18.5 Program collar band

A collar band according to Picture 18-8 with the application of the following x values is to be programmed:

X	Definition	Step	Value
1	raise CB	p1⇒p2	35mm
2	collar fall width	p2⇒p4	20mm
3	collar width CB	p4⇒p5	40mm
4	collar point (x) to p3	p3⇒p6	40mm
5	collar point (y) to p3	p6⇒p7	45mm
6	angle fold + neck line	in p3	90°
7	angle collar edge	in p7	80°

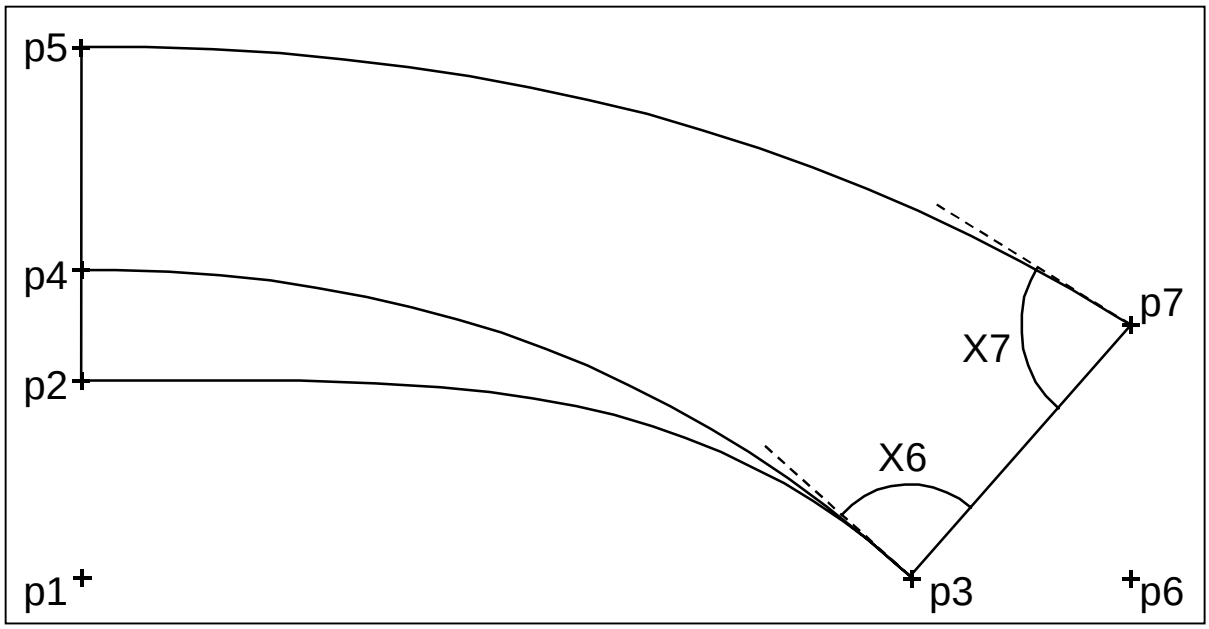
Application of x values

x values must be defined in the program header, immediately after the declaration lines for variables and constants. For definition of x values, the following applies

- With the line
  ```
  XTitel(".............")
  ```
  a program name is transferred which will later appear in the x value list of the basic block. With this, the user can recognise to which basic block these x values belong. The title may have a maximum of 50 characters.
- The x values must be defined at the beginning of the program with the following instruction structure:
  ```
  Defx(1,".....",10.0)
  Defx(2,".....",12.5)
      | |          |
      | |          |_  standard value
      | |____________  comment text
      |______________  consecutive number
  ```
- The standard values must only have one decimal and must be between the values of -3200. <= value <= 3200.
- The consecutive numbers must begin with 1 and ascend in numerical order (no gaps).
- Each x value can be assigned with size-specific values. The definition line of the x values is extended as follows:
  ```
  Defx(3,"addition to nape-waist",0,
&  "_36",3,"_46",4,"_036",1,"_046",2)
     |    |
  ```
  size value

  (_ stands for standard measurement chart!)

  The sign & stands for a continuation line. The x



Picture 18-8

value definition line can have more than one continuation lines.
The code „Defx“ with consecutive number, name and standard value can also be followed with size-specific x value allocations. In definition blocks, a size is allocated with a value which the x value is to apply for this size. The size names for standard measurement charts must start with `"_"`!

Programming of the points

As a rule, a program is developed in steps and tested after each step. Only when the first step has been successful, should the second step be commenced. The first step for programming of the collar band is the programming of the points at the centre back, followed by the points at the collar point.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar
  pVar p1,p2,p4,p5
  sVar
  qVar
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRi=0,rLe=180,rUp=90,rDo=270
  rCon
  tCon
'----------------- x value definitions
  XTitel("collar band")
  Defx(1,"raise CB",35)
  Defx(2,"collar fold width",20)
  Defx(3,"collar width CB",40)
  Defx(4,"collar point(x) to p3",40)
  Defx(5,"collar point(y) to p3",45)
  Defx(6,"ang neck+foldline in p3",90)
  Defx(7,"ang collar edge in p7",80)
'----------------------- points at CB
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rUp,rX(2))
  p5 = pPRiLng(p4,rUp,rX(3))
'---------------------- output points
  AusP(p1,p2,p4,p5)
'------------------------------------
  End Program
'************************************
```

The entries in the line `pVar` are entered by Grafis, automatically after compiling. The line `rCon`, however, contains the assignment for the main directions. A block with definition of the x values follows. For the first comment line, it is sufficient to enter

```
'- x value definitions
```

The remaining characters are entered by Grafis during automatic formatting. The x values are defined, consecutively in the following lines and contain no size-specific x values.
After having defined the x values, the first points are constructed.

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

... defines point `p1` with the co-ordinates (0,0). Thus, `p1` is the zero of the construction.

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

... defines point `p2` with the co-ordinates `(0,rX(1))`, where `rX(1)` applies the value of the first x value. Thus, `p1` is positioned upwards by „raise CB„.

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

… defines point `p4`, positioned upwards from `p2` with the distance `rX(2)` –the second x value.

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

… defines point `p5`, positioned upwards from `p4` with the distance `rX(3)` –the third x value.
Thus, the points at the CB are available within the program. They now have to be output to the screen. The following lines apply

```
'---------------------- output points
  AusP(p1,p2,p4,p5)
```

This first step should be tested, thoroughly with  ,  and  . Only the points of the centre back will appear on the screen. Measure the distances between the points and also their co-ordinates, if necessary. With the right button, you return to the programming user interface. Save the project via *Project* | *Save*.

In the next step, the points of the collar point are constructed. It is recommended to instruct the screen output in a block at the end of the program. Therefore, the next program lines are inserted directly above „`output points`“. All additions are highlighted.

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar
  pVar p1,p2,p3,p4,p5,p6,p7
  sVar
  qVar
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRi=0,rLe=180,rUp=90,rDo=270
  rCon rClLng=150
  tCon
'----------------- x value definitions
  XTitel("collar band")
  Defx(1,"raise CB",35)
  Defx(2,"collar fold width",20)
  Defx(3,"collar width CB",40)
  Defx(4,"collar point(x) to p3",40)
  Defx(5,"collar point(y) to p3",45)
  Defx(6,"ang neck+foldline in p3",90)
  Defx(7,"ang collar edge in p7",80)
'------------------- --- points at CB
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rUp,rX(2))
  p5 = pPRiLng(p4,rUp,rX(3))
'--------------- corner point p3 (CF)
```

```
  p3 = pXY(rClLng,0)
  p6 = pPRiLng(p3,rRi,rX(4))
  p7 = pPRiLng(p6,rUp,rX(5))
'------------ -- output points + lines
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(p2+p5)
  AusQ(p3+p7)
'-------------------------------------
  End Program
'*************************************
```

In this example, the collar width is to be fixed. In section 19.2, the instructions required for automatic length adjustment of a collar to the neck are explained. The collar length `rClLng` is defined as a constant of 150 mm in line

```
  rCon rClLng=150
```

Directly above the output of points and lines the following block was added:

```
'------------- corner point p3 (CB)
  p3 = pXY(rClLng,0)
```

... defines point `p3` with the co-ordinates `(rClLng,0)`. `p3` is positioned to the right of the zero point at a distance of the collar length.

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

… defines point `p6`, positioned to the right of `p3` at a distance of `rX(4)` –the fourth x value.

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

… defines point `p7`, positioned upwards from `p6` at a distance of `rX(5)` –the fifth x value.

In the line for output of the points, the new points `p3`, `p6` and `p7` were added.

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

With the lines

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

the centre back is output as a connection between points `p2` and `p5` as well as the line at the corner points between points `p3` and `p7`. With the output instruction `AusQ()`, lines and curves can be allocated for screen output. Instead of variables, the entry of line or curve functions is also permitted.

Test and check this step with  ,  and  . Save the project.

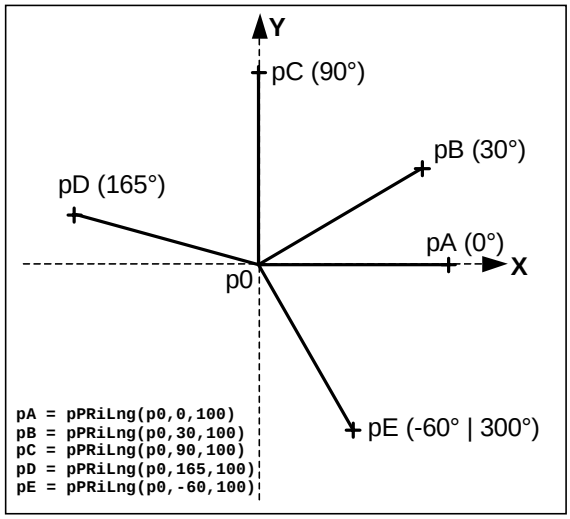
Calculate directions and angles

Direction and angle definitions

Directions are required for positioning of points in a direction and for construction of curves, etc. In the new programming language, directions are always defined as real numbers in degrees.

The points `pA` to `pE` from Picture 18-9 can be programmed as follows, where the distance to p0 is to be 100 mm, respectively:

```
  pA = pPRiLng(p0,0,100)
  pB = pPRiLng(p0,30,100)
  pC = pPRiLng(p0,90,100)
```



Picture 18-9

```
  pD = pPRiLng(p0,165,100)
  pE = pPRiLng(p0,-60,100)
```

Instead, a numbers in degrees, real variables can be entered as parameters.

Calculate directions

A direction can be defined as

- direction from first to second point with `rRiPP(p,p)`,
- direction of a line `rRiS(s)`,
- direction of a curve in starting or final point `rRiQanf(q)` or `rRiQend(q)` or
- direction of a curve in a curve point `rRiQP(q,p)`.

From a mathematical point of view, the direction is identical with a vector. Only when the vector is linked with a point, a straight line is created.

The direction of a point `pB` in relation to `p0` (Picture 18-9) can be calculated as follows:

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

After processing this line, `rB` has the value 30.

Calculate angles

An angle is calculated as

- angle defined by three points with `rWiPPP(p,p,p)` (starting, pivot and end point) or
- angle between two lines `rWiSS(s,s)`.

For the points in Picture 18-9, the functions in the left column result in the values of the right column.

Function	Result
`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

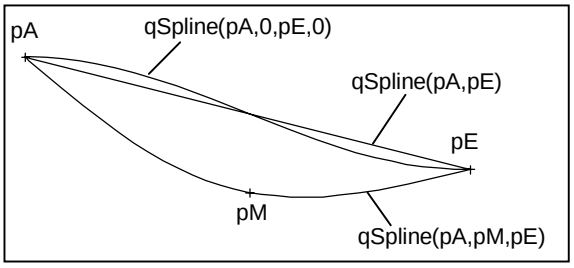
The first parameter in `rWiPPP(p,p,p)` defines the first side of the angle. The rotation direction relates to this side (positive or negative rotation angle).

The same applies to the function `rWiSS(s,s)`, where the sides of the angle must first be defined as lines.

The curve variation spline

A curve of the spline variation can run through any number of fulcrums (base points). In these fulcrums, directions can be defined for the curve. As on a steel ruler, the curve is bent so that all conditions can be fulfilled with as little bending force as possible.
For definition of a spline, a minimum of starting point and final point must be entered. The simplest variation with

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



Picture 18-10

defines a spline from pA to pE. The spline can take on any direction in these points and will therefore, appear as a line (Picture 18-10).
With the instruction line

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

the curve is also forced to run horizontally to the right in points pA and pE with direction 0°. For a reversed curve direction the instruction should be

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

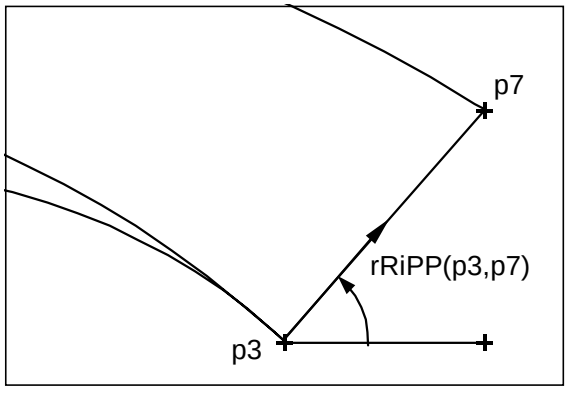
Each curve has a direction !
With the instruction line

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

a curve through three points is created, where the directions in the points are not assigned.

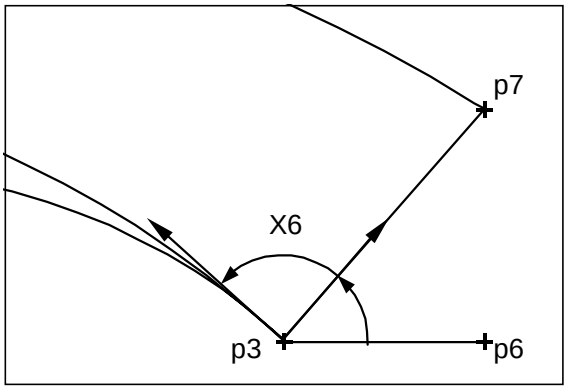
Construct neck line and collar fold line as spline with defined directions

The neck line and the collar fold line should be constructed as splines. The starting point for both curves is p3. Both curves should start at the angle x6, relating to the connection of points p3 to p7.



Picture 18-11

First, calculate the direction from p3 to p7 (Picture 18-11).
The calculated direction is still to be rotated about the given angle (Picture 18-12).



Picture 18-12

The following lines generate the result:

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

For the collar edge, it is to be considered that the angle in p7 is entered as inside the collar. The direction of the curve in p7 can be calculated either with

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

or with

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

After

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

the collar edge is created but not yet, output to the screen. The output instruction for the three curves should read

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

The program for the collar band with (as yet) preset collar length is ready:

```
'************************************
  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 rRi=0,rLe=180,rUp=90,rDo=270
  rCon rClLng=150
  tCon
'---------------- x value definitions
  XTitel("collar band")
  Defx(1,"raise CB",35)
  Defx(2,"collar fold width",20)
  Defx(3,"collar width CB",40)
  Defx(4,"collar point(x) to p3",40)
  Defx(5,"collar point(y) to p3",45)
  Defx(6,"ang neck+foldline in p3",90)
  Defx(7,"ang collar edge in p7",80)
'---------------------- points at CB
```

```
  p1 = pXY(0,0)
  p2 = pXY(0,rX(1))
  p4 = pPRiLng(p2,rUp,rX(2))
  p5 = pPRiLng(p4,rUp,rX(3))
'---------------- corner point p3 (CF)
  p3 = pXY(rClLng,0)
  p6 = pPRiLng(p3,rRi,rX(4))
  p7 = pPRiLng(p6,rUp,rX(5))
'-------------------------- neck line
  rWi3  = rWiPPP(p6,p3,p7)
  rWi3  = rWi3+rX(6)
  q1 = qSpline(p3,rWi3,p2,rLe)
  q2 = qSpline(p3,rWi3,p4,rLe)
'------------------------ collar edge
  rWi7  = rRiPP(p7,p3)-rX(7)
  q3 = qSpline(p7,rWi7,p5,rLe)
'---------- ---- output points + lines
  AusP(p1,p2,p3,p4,p5,p6,p7)
  AusQ(sPP(p2,p5))
  AusQ(sPP(p3,p7))
  AusQ(q1,q2,q3)
'-----------------------------------
  End Program
'***********************************
```

18.6 Program skirt

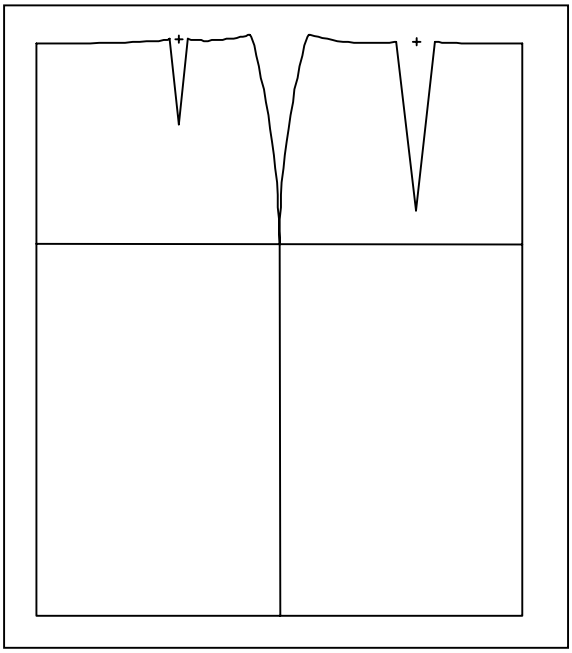
The basic block skirt according to Picture 18-13 is to be programmed with the application of the given x values. The generation of the program ensues in four steps. In each step, there are construction steps (table), a picture, and the program up to the stage displayed. The steps should initially be developed by yourself and then, compared with the prepared program text.

When creating the skirt, you may have the following questions:

What do I do if an error message appears?
How can I find the relevant function ?
What is to be considered for release of a program ?
What is to be considered for alterations / corrections of a program ?

The answers to those questions can be found in the last section 18.7 of this chapter.

X	Definition	Value
1	skirt length from waist	600mm
2	addition to half hip girth	10mm
3	addition to half waist girth	10mm
4	relocate sided seam to front	0mm
5	raise side seam	10mm
6	dart length front	90mm
7	dart point from hip line in bk	35mm



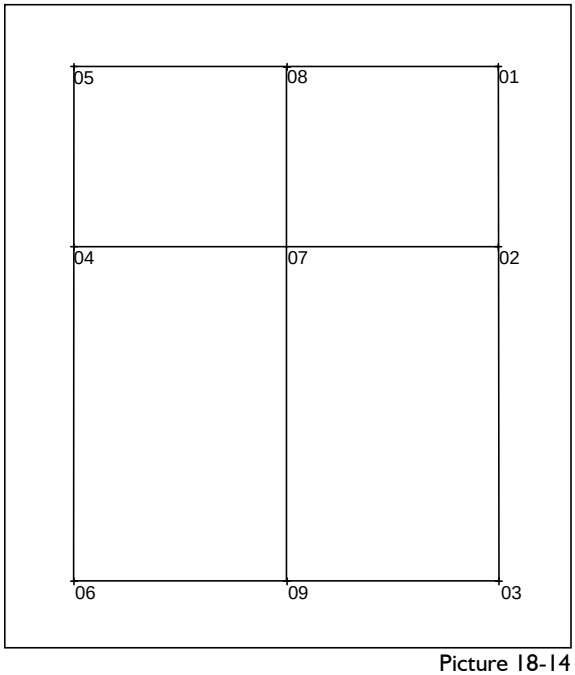
Picture 18-13

1st step: construct points of the centre back, centre front and side seam (Picture 18-14)

from	to	direction	distance
01	02	⇓	G10 (waist to hip)
01	03	⇓	x1 (skirt length from waist)
01	05	⇐	G2/2+X2 (half hip girth + ease)
02	04	⇐	G2/2+X2
03	06	⇐	G2/2+X2
02	07	⇐	½ distance p02⇔p04 + X4
01	08	⇐	½ distance p02⇔p04 + X4
03	09	⇐	½ distance 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 rRi=0,rUp=90,rLe=180,rDo=270
  tCon
'----------------- x value definitions
  XTitel("skirt")
  Defx(1,"skirt length f. waist",600)
  Defx(2,"addition 1/2 hip girth",10)
  Defx(3,"addition ½ waist girth",10)
```

```
  Defx(4,"relocate side seam to ft",0)
  Defx(5,"raise side seam",10)
  Defx(6,"dart length front",90)
  Defx(7,"dart p. f. hip line bk",35)
'------------------------ points at CB
  p01 = pXY(0,0)
  p02 = pXY(0,-rG(10))
  p03 = pXY(0,-rX(1))
'------------------------ points at CF
  rZ  = rG(2)/2+rX(2)
  p05 = pPRiLng(p01,rLe,rZ)
  p04 = pPRiLng(p02,rLe,rZ)
  p06 = pPRiLng(p03,rLe,rZ)
'----------------- points at side seam
  rZ  = rAbstPP(p02,p04)/2+rX(4)
  p07 = pPRiLng(p02,rLe,rZ)
  p08 = pPRiLng(p01,rLe,rZ)
  p09 = pPRiLng(p03,rLe,rZ)
'---------------------- output points
  AusP(p01,p02,p03,p04,p05,p06,p07,
&     p08,p09)
'------------------------ output lines
  AusQ(p01+p03)
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
'-----------------------------------
  End Program
'***********************************
```

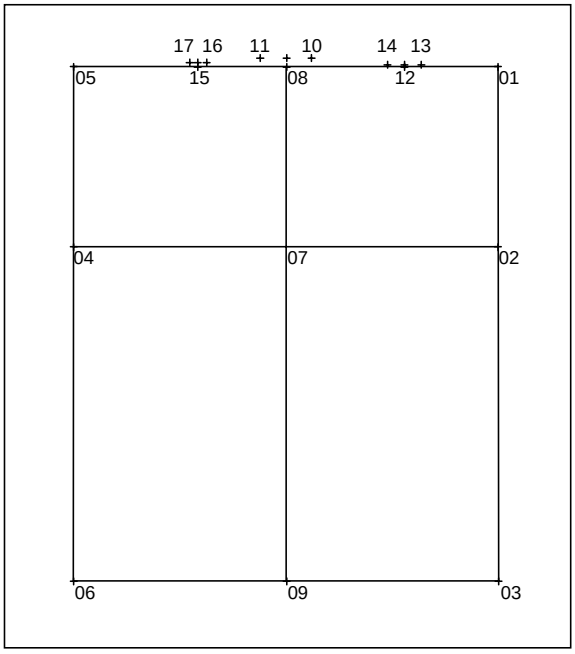


Picture 18-14

2nd step: calculate & distribute ease (Pict. 18-15)

ease *Ea=(G2/2+X2)-(G4/2+X4)*
portion side seam *3/6 ease*
portion skirt back *2/6 ease*
portion skirt front *1/6 ease*

from	to	direction	distance
08	08	⇑	X5 (raise side seam)
08	10	⇒	½ * 3/6 * *ease*
08	11	⇐	½ * 3/6 * *ease*
01	12	⇐	½ distance p01⇔p10
12	12	⇑	¼*X5 (raise waist)
12	13	⇒	½ * 2/6 * *ease*
12	14	⇐	½ * 2/6 * *ease*
05	15	⇒	2/3 distance p05⇔p11
15	15	⇑	½ *X5 (raise waist)
15	16	⇒	½ * 1/6 * *ease*
15	17	⇐	½ * 1/6 * *ease*



Picture 18-15

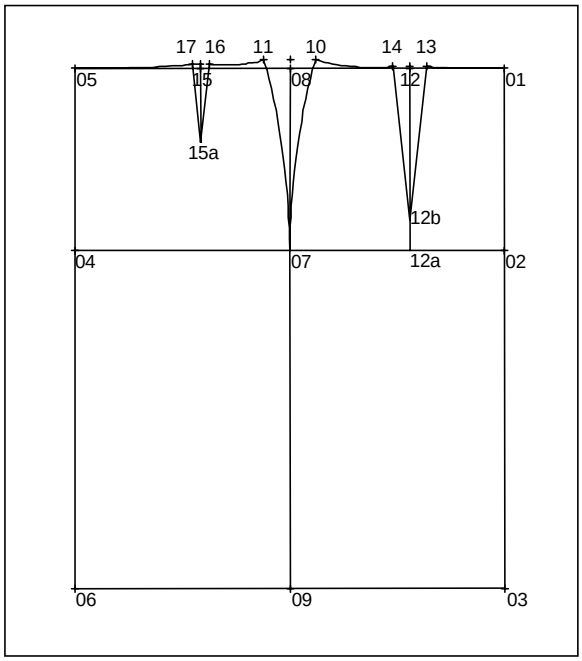
```
'***********************************
  Program Main()
  lVar
  nVar
  rVar rZ,rEa,rSs,rBk,rFt

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 rRi=0,rUp=90,rLe=180,rDo=270
  tCon
'---------------- x value definitions
  XTitel("skirt")
  Defx(1,"skirt length f. waist",600)
  Defx(2,"addition 1/2 hip girth",10)
  Defx(3,"addition ½ waist girth",10)
  Defx(4,"relocate side seam to ft",0)
  Defx(5,"raise side seam",10)
  Defx(6,"dart length front",90)
  Defx(7,"dart p. f. hip line bk",35)
'------------------------ points at CB
  p01 = pXY(0,0)
  p02 = pXY(0,-rG(10))
  p03 = pXY(0,-rX(1))
'----------------------- points at CF
  rZ  = rG(2)/2+rX(2)
  p05 = pPRiLng(p01,rLe,rZ)
  p04 = pPRiLng(p02,rLe,rZ)
  p06 = pPRiLng(p03,rLe,rZ)
'---------------- points at side seam
  rZ  = rAbstPP(p02,p04)/2+rX(4)
  p07 = pPRiLng(p02,rLe,rZ)
  p08 = pPRiLng(p01,rLe,rZ)
  p09 = pPRiLng(p03,rLe,rZ)
'--------------------- distribute ease
  rEa = (rG(2)/2+rX(2))
&       -(rG(4)/2+rX(3))
                        'on half skirt
  rSs = 3/6*rEa  'portion in side seam
  rBk = 2/6*rEa          'portion in Bk
  rFt = 1/6*rEa          'portion in Ft
'-------==-- reduce side seam at waist
  p08 = pPRiLng(p08,rUp,rX(5))
  p10 = pPRiLng(p08,rRi,rSs/2)
  p11 = pPRiLng(p08,rLe,rSs/2)
'------------------------ darts in Bk
  rZ  = rAbstPP(p01,p10)/2
  p12 = pPRiLng(p01,rLe,rZ)
  p12 = pPRiLng(p12,rUp,rX(5)/4)
  p13 = pPRiLng(p12,rRi,rBk/2)
  p14 = pPRiLng(p12,rLe,rBk/2)
'------------------------ darts in Ft
  rZ  = rAbstPP(p11,p05)*2/3
  p15 = pPRiLng(p05,rRi,rZ)
  p15 = pPRiLng(p15,rUp,rX(5)/2)
  p16 = pPRiLng(p15,rRi,rFt/2)
  p17 = pPRiLng(p15,rLe,rFt/2)
'----------------------- output points
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17)
'------------------------ output lines
  AusQ(p01+p03)
  AusQ(p03+p06)
  AusQ(p06+p05)
  AusQ(p04+p02)
  AusQ(p05+p01)
  AusQ(p08+p09)
  End Program
'************************************
```

3rd step: draw dart lines (Picture 18-16)

from	to	direction	distance
12	12a	perp.	perpendicular from p12 onto line p02⇔p07
12a	12b	⇑	X7
15	15a	⇓	X6
			draw dart lines



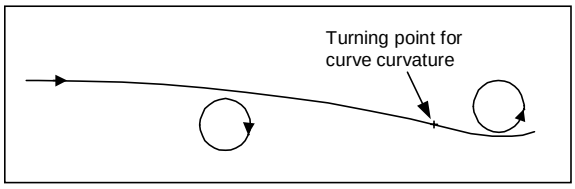
Picture 18-16

```
'************************************
  Program Main()
'------------------------------------
  lVar
  nVar
  rVar rZ,rEa,rSs,rBk,rFt
  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 rRi=0,rUp=90,rLe=180,rDo=270
  tCon
'---------------- x value definitions
  ... as before ...
'------------------------ darts in Ft
  rZ  = rAbstPP(p11,p05)*2/3
  p15 = pPRiLng(p05,rRi,rZ)
  p15 = pPRiLng(p15,rUp,rX(5)/2)
  p16 = pPRiLng(p15,rRi,rFt/2)
  p17 = pPRiLng(p15,rLe,rFt/2)
'---------------------- dart point Bk
  sZ  = sPP(p02,p07)
  p12a= pLotPS(p12,sZ)
  p12b= pPRiLng(p12a,rUp,rX(7))
'---------------------- dart point Ft
  p15a= pPRiLng(p15,rDo,rX(6))
'----------------------- output points
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a)
'-----------------------output lines
  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
'***********************************
```

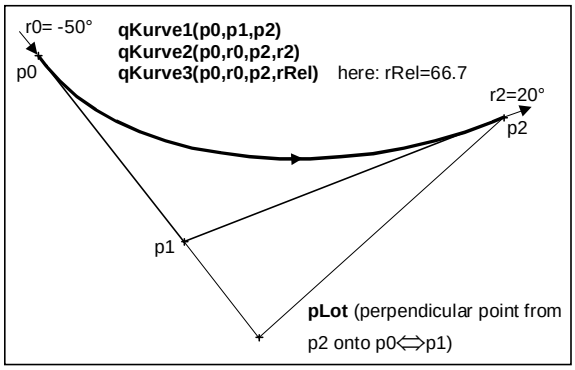
The curve variation circle arc

The curve type circle arc curve is based on distorted, degenerate circle arcs. One of the main differences to the curve type spline is that a circle arc curve has no turning points (Picture 18-17). A curve shape according to Picture 18-17 can only be constructed with the curve type spline.



Picture 18-17

For the curve type circle arc curve, three definition options are available which deliver the same curve shape if comparable parameters are entered. ***Circle arc curves give relatively shallow curves. They are especially suitable for hip and waist curves. If the curve shape with one of the circle arc options is not satisfactory, the only alternative is a curve of the spline type.*** The curve in Picture 18-18 was constructed with one of the three definition options, respectively.



Picture 18-18

qKurve1(pS,pD,pE)

The curve is created from pS to pE. The parameters

- starting point pS,
- direction point pD and
- end point pE are to be entered.

The direction point pD determines the direction of the curve in pS and pE. In pS the curve has the direction pS⇒pD and in pE the direction pD⇒pE. Thus, the curve nestles against the lines pS⇒pD and pD⇒pE.

qKurve2(pS,rS,pE,rE)

The curve is created from pS to pE. The parameters

- starting point pS,
- direction in the starting point rS,
- end point pE and
- the direction in the end point rE are to be entered.

The directions in starting and end point, create the direction point of the first definition option.

qKurve3(pS,rS,pE,rRel[,rE])

The curve is created from pS to pE. The parameters

- starting point pS,
- direction in the starting point rS,
- the end point pE,
- a relative value for the curve shape rRel and
- optional, as result value the direction in end point rE are to be entered.

The direction of the curve in the end point is calculated from the relative value with the following rule:

From the end point, a perpendicular is dropped onto the line starting point with starting direction. The distance pS⇔direction point (analogous curve type1) is calculated from rRel/100*distance pS ⇔perpendicular point.

The value rRel, indirectly adjusts the direction in the end point. The curve shape can thus be altered very delicately. However, it is unsuitable if particular directions must be considered in the starting point and end point.

4th step: draw side seam and waist lines

For the construction of the side seam, the construction option „qKurve3“ is applied as the direction of the side seam at waist is still free. Thus, the hip curve can be adjusted to the optimum shape with the parameter rRel. The hip curve in the front skirt is created with

```
qSs_ft = qKurve3(p07,rUp,p11,60)
```

Alter the numeric value 60 in steps of 5 and after  ,  (also  with a number of sizes) and  adjust a nice hip curve.

With mirror at p07⇔p08, the hip curve in the skirt back is created.

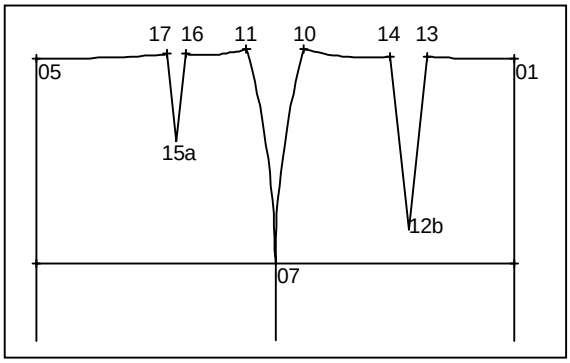
```
qSs_bk = qSs_ft
Spgl(sPP(p07,p08):qSs_bk)
```

Before mirroring, it is first switched to the new curve variable qSs_bk. With

```
Spgl(sPP(p07,p08):qSs_bk=qSs_ft)
```

the switch would ensue directly in the mirror function.

The waist lines are to run at right angle to the side seam, the darts and the centre front and centre back, respectively. Before the individual sections of the waist can be created, the direction of the waist line in starting and final point must be calculated.



Picture 18-19

The waist line from centre back starts at point p01 in horizontal direction to the left (180°) and ends in p13 at right angle to the direction p12b⇒p13 (Picture 18-19). The direction is calculated with rRiPP(p12b,p13) and rotated through 90° in a mathematically positive direction with +90.

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

Then, the waist section from the centre back can be constructed with

```
qWa_bk1= qKurve2(p01,rLe,p13,rRi13)
```

The waist line is to be at right angle to the side seam. With rRiQend(qSs_bk), the direction in the end point of the side seam is calculated. +90 rotates the direction again through 90° in mathematically positive direction. The result is the required direction of the waist rRi10 in the end point.

The waist section from the dart in the skirt back to the side seam can then, be constructed with

```
qWa_bk2=qKurve2(p14,rRi14,p10,rRi10)
```

Analogous, the construction of the waist sections in the front skirt follows.

```
'************************************
  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 qSs_ft,qSs_bk,
&      qWa_bk1,qWa_bk2,
&      qWa_ft1,qWa_ft2
  tVar
  cVar
'------------------------------------
  lCon
  nCon
  rCon rRi=0,rUp=90,rLe=180,rDo=270
  tCon
'----------------- x value definitions
  ... as before ...
'---------------------- dart point Ft
  p15a= pPRiLng(p15,rDo,rX(6))
'----------- draw and mirror side seam
  qSs_ft = qKurve3(p07,rUp,p11,60)
  qSs_bk = qSs_ft
  Spgl(sPP(p07,p08):qSs_bk)
'--------- draw waist line in skirt bk
  rRi13  = rRiPP(p12b,p13)+90
  qWa_bk1= qKurve2(p01,rLe,p13,rRi13)
  rRi14  = rRiPP(p12b,p14)+90
  rRi10  = rRiQend(qSs_bk)+90
  qWa_bk2=qKurve2(p14,rRi14,p10,rRi10)
'--------- draw waist line in skirt ft
  rRi17  = rRiPP(p15a,p17)+90
  qWa_ft1= qKurve2(p05,rRi,p17,rRi17)
  rRi16  = rRiPP(p15a,p16)+90
  rRi11  = rRiQend(qSs_ft)+90
  qWa_ft2=qKurve2(p16,rRi16,p11,rRi11)
'----------------------- output points
  AusP(p01,p02,p03,p04,p05,p06,p07,
&      p08,p09,p10,p11,p12,p13,p14,
&      p15,p16,p17,p12a,p12b,p15a)
'------------------------ output lines
  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(qSs_ft,qSs_bk,qWa_ft1,qWa_ft2,
&      qWa_bk1,qWa_bk2)
'------------------------------------
  End Program
'************************************
```

18.7 General guidelines

With the instructions and functions introduced so far, the majority of basic blocks can be translated into programming language programs. An overview of all available functions can be found in the Grafis Help.

How can I find the relevant function?

First, establish the variable type for the result. If a point is required, only functions beginning with „p“ are relevant, for lines, only the functions with „s“ etc. As a rule, a code for the result type appears in the name of the function

Code	Result type	Example
Wi	angle	rWiSS(s,s)
Ri	direction	rRiPP(p,p)
Lng	total length	rLngQ(q)
TIng	partial length	rTIngSP(s,p)
RIng	relative length	rRIngSP(s,p)
Lot	perpendicular	pLotPS(p,s)
Tang	tangent	pTangPQ(p,q)
…	…	…

followed by the required parameters in capital letter.

What do I do if an error message appears?

There are two types of errors:

- ***syntax error*** = error in the writing convention („spelling error“) and
- ***logical error***, which appears during processing of the program.

Syntax errors are reported during compiling. The respective line is highlighted and a suggestion is made. Syntax errors are for example missing opening / closing brackets, unknown functions or incorrect parameter types in the functions. Syntax errors can usually be rectified, quickly.

A logical error occurs, when the program does not deliver the expected result. ***Logical errors are easier to be found, if the program is developed in small steps and each step is tested, thoroughly (also in small/large sizes).*** In this case, the error is to be found in the last step. For long programs, it is definitely useful to print and annotate the points, lines and curves at a particular interval, analogous to Pictures 18-14, 18-15, 18-16 and 18-19.

A few tips for identifying errors:

- The value of a variable of any type can be checked with the instruction `VList()`. With the instruction line
  ```
  rRi11  = rRiQend(qSs_ft)+90
  VList(rRi11)
  ```
  the value of the variable `rRi11` is displayed in the variable list after  or  .
- The line in which a variable has been allocated with the current value can be found by selecting the variable in the current line and then searching for it with  .
- To identify a searched point `pW`, output a line from the zero point to the searched point: `AusQ(pXY(0,0)+pW)`
- If a point is generated as an intersection between a circle and a line, output the circle and the line, temporarily and observe the result, also in small/large sizes. With a „'“ in front of this temporary output, the line becomes a comment line.
- During compiling, a message appears querying whether undeclared variables are to be declared. Check for each query whether the variable has actually been used or whether is has been „created“ through a typing error.
- After successful compiling, unused variables will be stated in the message window. In tidy programming, unused variables are often an indication of a mix-up.

What is to be considered for release of a program?

Before releasing a program, it should be checked whether

- the program runs in all sizes, also extremely small/ large/ individual sizes without errors,
- all x values are set up, calculated and commented, correctly. An „addition to the waist girth“ must not operate as „addition to the half waist girth“. A positive value for „increase side seam“ must not lead to the side seam being reduced,
- only necessary objects (points, lines, curves) are output. Objects which are not required by the user should not be output,
- all line lengths are correct. In the skirt construction of the previous section, for example the lengths of the side seams in skirt front and back should be compared and the sum of the waist lines should be measured, also in other sizes.

After releasing the program, the program file *.cpr is to be entered into the call list and an info mask is to be generated.

Furthermore, generate a documentation of the following content:

- a print-out of the construction in which all objects are annotated. All used objects should be output, including auxiliary points and lines not appearing in the released construction.
- a print-out of the program,
- the program as file and
- a copy of the construction instructions.

What is to be considered for alterations / corrections of a program ?

Corrections in released programs must be carried out very circumspect manner, as styles developed from this program will always fall back on it.

Before any alteration of released programs, the alteration code must be increased in the project user interface via Extras | Options! This applies, especially to alterations for object output.

For explanation, the record principle of Grafis is to be described at this stage. Each output instruction of a programming language program transfers objects (points, lines) to the Grafis record. The objects obtain Pos-numbers in order of their transfer. The Pos-number is an identification for the objects in the Grafis record. The output instruction

```
AusQ(qHem,qInsideleg,qCrotch)
```

transfers the hem, the inside leg and the crotch seam to the Grafis record which will allocate these lines with the consecutive Pos-Numbers 1, 2 and 3. If now, a parallel is created to the inside leg seam, this record step relates to the object with Pos-Number 2.

If later on, the instruction is altered in the programming language program to

```
AusQ(qInsideleg,qHem,qCrotch)
```

and a test run is carried out in the style, the parallel appears at the hem instead of the inside leg. This alteration will result in errors only in styles which have been developed from the programming language program before the alteration.

The objects must always be output in the same order for identical object types, irrespective of sizes and x values. Therefore, output instructions within IF-ENDIF structures should be avoided.

The alteration code should also be increased before correcting any numbers or formulae. A user of your program may have already corrected the basic shape with construction steps. These construction steps are also carried out in the altered program.