Einführung

Erste Schritte zu Kommandos und Kommandofolgen wurden bereits in der vorhergehenden Lektion behandelt. In diesem Kapitel soll das Wissen dahingehend erweitert werden, dass Sie am Ende selbst in der Lage sind, einfache, funktionsfähige Programme zu erstellen. Zu Beginn sollen ausschließlich Prozeduren beschrieben werden.

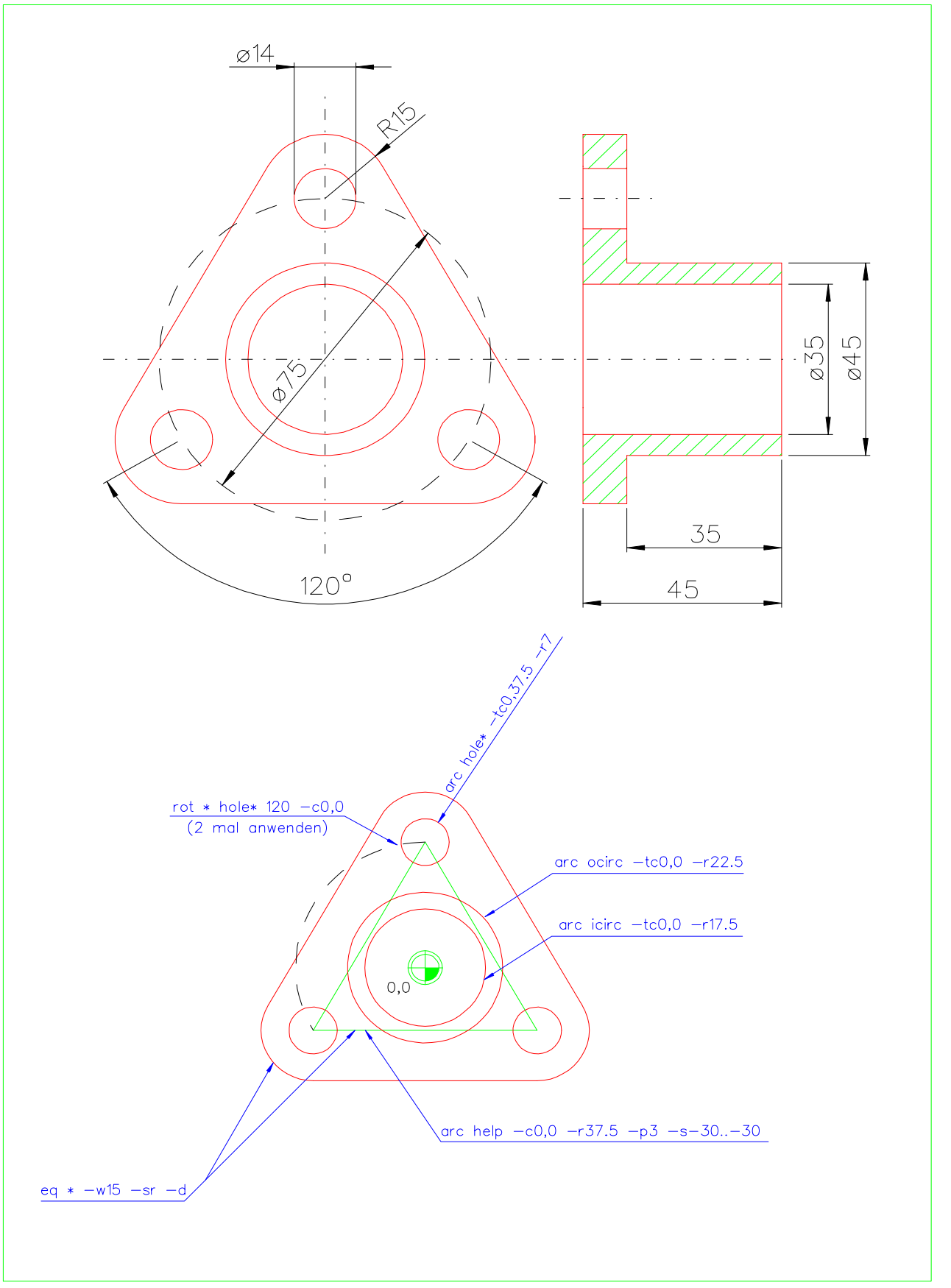
Was bedeutet eigentlich der Begriff "Prozedur"? In unserer Definition ist eine Prozedur ein unter **Pictures by PC** ablauffähiges Programm in der systemeigenen Kommandosprache. In dieser werden ausführbare Kommandos mit Sprachelementen, wie z.B. Abfragen, Meldungen, Fallunterscheidungen, Schleifen, Mathematik etc. gekoppelt und, bei Einhaltung syntaktischer Regeln, schrittweise (prozedural, interpretativ) ausgeführt. Die Funktionalität von **Pictures by PC** lässt sich mit Prozeduren nahezu unbegrenzt erweitern und individualisieren. Eine Prozedur besteht aus einer (den Regeln entsprechenden) Textdatei, die in dem Prozedur-Verzeichnis "Procs" oder deren Unterverzeichnisse (z.B. samples) gespeichert sein sollte.

Pictures by PC verfügt über eine enorme Anzahl von Prozeduren, deren Wirkungen z.B. von der Zahnrad-Genierung (cogwheel.prc) bis zur Erzeugung von Blindenschrift (Braille-Schrift, braille.prc) und vielem mehr reichen. Im Laufe der 35-jährigen Entwicklung hat sich da Einiges angesammelt. Der Fundus wächst ständig und stellt schon eine Art "Wundertüte" dar.

Sobald eine neue Prozedur unter "procs" abgelegt wird, verfügt **Pictures** über diese Funktionalität, die wie ein neues Kommando wirkt (daher der Name "Kommando-Prozedur").

Lassen Sie uns das jetzt im Folgenden "erforschen". Dabei soll das Hauptaugenmerk auf den praktischen Nutzen einer neuen Funktion gerichtet werden und nicht auf den Anspruch einer perfekten Programmierung.

Als Aufgabenstellung soll die Erzeugung eines einfachen 3-Loch-Dreiecksflanschs (s. Abb.) gewählt werden. Obwohl in der Literatur zum CAD-Basis-Seminar ("Triflange") als auch in der Kommando-Sequenz-Datei "NuetzlicheKommandos.stk" schon vergleichbare Lösungsansätze vorliegen, soll an dieser Stelle die Wirkungsweise Schritt für Schritt erklärt werden. Um das Maß an Mathematik momentan möglichst gering zu halten, wird eine spezielle Konstruktionsmethode gewählt, die Sie vermutlich so nicht genutzt hätten, die aber recht "gewitzt" ist.


⌀14
R15
⌀75
120°
⌀35
⌀45
35
45
arc hole* –tc0,37.5 –r7
rot * hole* 120 –c0,0
(2 mal anwenden)
arc ocirc –tc0,0 –r22.5
arc icirc –tc0,0 –r17.5
0,0
arc help –c0,0 –r37.5 –p3 –s–30..–30
eq * –w15 –sr –d

In der Abbildung erkennen Sie eine spezielle, blaue Bemaßungsskizze, bestehend aus Kommandos, die die zugeordneten Geometrien erzeugt haben.

Begonnen werden soll mit einem "Hilfsdreieck", dessen "Nutzen" sich erst im zweiten Arbeitsschritt erschließt. Geben Sie einfach folgendes Kommando in die Kommandozeile von Pictures ein:

```
arc help -c0,0 -r37.5 -p3 -s-30..-30
```

Dieses Dreieck wird überraschenderweise nicht mit einem "poly-" sondern mit einem "entarteten" "arc-"Kommando erzeugt. Prinzipiell lässt sich mit der Option "-p#" ein Kreis aus Geraden-Segmenten erzeugen. Im "Extremfall" können Sie also mit dem "arc-"Kommando und der Option "-p1" einen Punkt, mit "-p2" eine Gerade und, wie in unserem Fall, mit "-p3" ein Dreieck zeichnen. Lesen Sie selbst in der Hilfe-Funktion ("arc" eintippen und F1-Taste) die Kommando-Syntax nach. In unserem Kommando wird der Objektname explizit "help" genannt, da dieses Objekt nur eine Hilfsgeometrie darstellt und später gezielt gelöscht wird. Die Option "-r37.5" ist der "Radius" des Dreiecks (Lochführungskreis, Abstand zum obersten Loch). "-p3" wurde schon erklärt. "-s-30,,--30" besagt, dass das Dreieck als Segment mit den Winkelangaben von Winkel "-30" bis "-30" erzeugt wird. Gewöhnlich würde die Option "-t" für einen "Voll-Dreiecks-Kreis" genutzt. Der würde aber mit dem Winkel 0 (also auf der Horizontalen) beginnen Unsere Dreieck wäre also gegenüber den Konstruktionsvorgabe um 30 Grad verdreht. Dieses vermeidet man mit der Angabe der Option "-s-30..-30".

Im nächste Schritt wird mit dem Kommando

```
eq * -w15 -sr -d --nod
```

für "ältere" Pictures-Versionen

```
eq * -w15 -sr -d
```

zu dem zuvor generierten Dreieck als Duplikat (-d) eine Äquidistante mit einem Abstand von 15 "-w15" rechts außen (-sr) erzeugt.

Das ist, wie Sie sehen, ein schöner Trick. Wir erhalten die Außenkontur "automatisch", d.h. ohne Generierung von Vollkreisen und Trimmen. Das erklärt jetzt auch die Funktion unseres "Hilfs-Dreiecks".

Wenn Sie mit der F1-Hilfe die Informationen zu "eq" nachlesen, erkennen Sie, dass es sich um ein häufig genutztes, sehr vielseitiges und komplexes Kommando (genau genommen ein BIX-Routine, die wie ein Kommando wirkt) handelt. Auf weitere Details soll an dieser Stelle nicht näher eingegangen werden. Das würde Stunden dauern. Eines muss allerding noch erwähnt werden. Mit **Pictures Rev. 3.8** wurde zur einfacheren, interaktiven Bedienung von "eq" der Benutzer-Dialog erweitert, so dass der Benutzer-Dialog noch eine (weitere) manuelle Eingabe erwartet, die (hier mit "0") quittiert werden muss. Daher muss für diesen Fall das Kommando

```
eq * -w15 -sr -d --nod
```

lauten. Mit der Option "--nod" (no dialog) wird bei der "automatischen" Kommandonutzung die "störende", manuelle Eingabe unterdrückt.

Die soeben mit "eq" erzeugte Außenkontur bildet zusammen mit der 3-Lochanordnung und dem Zentralloch die Grundgeometrie für den Flansch. Diese Objekte sollen für die spätere 3D-Extrusion zu einem Makro-Objekt zusammengefasst werden. Also umschließen wir das aktive Objekt (die mit "eq" erzeugte Außenkontur) mit einem (unsichtbaren) Hüllobjekt namens "tri" mit folgendem Kommando

```
cover * tri
```

In der Zeichnung bewirkt das Kommando optisch nichts. Allerdings hat sich die interne Objektstruktur (unsichtbar) geändert. "tri" ist jetzt das Makro-Objekt (Hüllobjekt) zur Außenkontur. Momentan enthält das Makro "tri" nur das eine Unterobjekt (die "eq"-Kontur). Im Folgenden werden aber noch zusätzlich benötigte Objekte hinzugefügt.

Nach einem "cover"-Kommando sollte immer ein

```
box *
```

oder alternativ ein

```
box all
```

folgen. Damit übernimmt das Makro-Objekt intern auch die Objektabmessungen der eingeschlossenen Kontur (vgl. box-Kommando, Vektor-Befehl "bx").

Als nächstes wird der oberste Kreis der Lochanordnung mit dem Kommando

```
arc hole* -tc0,37.5 -r7
```

erzeugt und erhält dabei den Objektnamen "hole1". Dieser Kreis wird jetzt mit

```
cat * tri -i
```

zu dem Makro-Objekt "tri" hinzugefügt (cat für concatenate, verknüpfen). "tri" hat also jetzt zwei Unterobjekte.

Achtung! Im Kommando oben, dürfen Sie auf keinen Fall die Option "-i" vergessen. Anderenfalls wird das Ausgangsobjekt fälschlich fest mit dem Zielobjekt verknüpft und verliert seinen Objektnamen. Das ist eine ungewünschte und störende Objektstruktur.

Mit der letzten Verknüpfung ist jetzt das Makro "tri" das aktive Objekt.

Sie können das überprüfen, indem sie Folgendes eingeben

```
set actobj
```

oder alternativ

```
echo $actobj
```

Übrigens bei der Meldung "1:tri" bedeutet "1:", dass sich das Objekt im Zeichnungspuffer 1 befindet (s. Variablen DEFBUF , MODBUF)

Da jetzt die Lochanordnung vervollständigt werden soll, muss wieder auf das Objekt "hole1" zurückgegriffen werden. Grundsätzlich erreicht man das durch

```
find hole1
```

Damit ist das Objekt wieder "aktiv". Das könnte man auch optisch anzeigen mit der "ACT-Anzeige" aus dem Buttonmenü  oder durch das Toggle-Kommando

```
bix ToggleMSTATE verbose:=true
```

Man hätte aber noch einfacher den Namen dezidiert im Kommando angeben können

```
rot hole1 hole* 120 -c0,0
```

Das bewirkt, dass das Objekt "hole1" bei der Drehung mit 120 Grad um den Nullpunkt ("-c0,0" (für center)) dupliziert wird und dieses den Namen "hole2" erhält.

(Die Angabe von zwei Objektnamen bewirkt die Duplizierung (aktives Quellobjekt "*" wird zum Zielobjekt "hole2"))

Letzteres wird wieder mit dem Makro verkettet

```
cat * tri -i
```

Und das 3. Loch wird dann natürlich durch die Kommandofolge

```
rot hole2 hole* 120 -c0,0
cat * tri -i
```

erzeugt.

Zur Vervollständigung der Grundplatte fehlt jetzt nur noch das zentrale Loch. Das bewirken die Kommandos

```
arc icirc -tc0,0 -r17.5
cat * tri -i
box tri
```

Den größeren Außenkreis erzeugen wir, wie gewohnt mit

```
arc ocirc -tc0,0 -r22.5
```

verknüpfen ihn aber nicht.

Alle notwendigen, zweidimensionalen Geometrien sind jetzt entsprechend der Zeichnungsvorlage erstellt. Also gilt es nur noch daraus einen 3D-Körper zu erstellen. Die 2D-Geometrie-Objekte des Makroobjekts "tri" müssen zu einem 10 mm dicken 3D-Volumenkörper generiert werden. Das leistet das Kommando

```
extrude3d tri -h10
```

oder besser noch, um den Dialog (mit Standard-Werten) zu unterdrücken

```
extrude3d tri -h10 --nod
```

Wir sprechen hier von einer Extrusion der (geschlossenen) Konturen des Makro-Objekts "tri" bis zur Höhe von 10 mm ("-h10"). Den entstandenen 3D-Volumenkörper benennen wir, z.B. durch

```
name * tribase
```

Um das Ergebnis besser erkennen zu können, empfiehlt sich eine isometrische Sicht, entweder per Button 

oder mit dem Kommando

```
setpp -i
```

Ggf. kann auch der Schattiermodus eingeschaltet werden

```
shade -d
```

Der Rohransatz des Flansches wird ebenfalls durch Extrusion aus den zwei Zentrumkreisen erzeugt. Allerdings braucht der Rohrkörper, da die Dreiecksplatte schon existiert, erst ab einer Höhe von 10mm zu beginnen, muss dann 35 mm lang sein und endet also auf einer Höhe von 45 mm. Das erzeugende Kommando lautet somit

```
extrude3d icirc ocirc -h10..45 --nod
```

(Alternativ hätte die Höhe aber auch bei h=0 beginnen können. Die dadurch entstehende Köperüberschneidung würde dann anschließend durch die Köpervereinigung "beseitigt".)

Es ist nun ein zweiter Volumenkörper erzeugt worden, der noch mit dem Dreieckskörper mittels logischer Verknüpfung (Addition nach Regel des englischen Mathematikers Bool, "Boolsche Operation") verbunden werden muss. Das geschieht mit

```
bop3d * tribase -u
```

Damit ist ein einziger Volumenkörper (die Option "-u" bedeutet "unify", Vereinigung, Addition) entstanden, der auch noch mit

```
name * flange*
```

sinnvoll zu "flange1" umbenannt wurde.

Sobald der 3D-Flansch fehlerfrei erzeugt wurde, können die zweidimensionalen "Hilfsobjekte" gelöscht werden. Das geschieht mit

```
del help
del tri
del ocirc
```

Mit den oben beschriebenen Arbeitsschritten wurde zunächst einmal die Aufgabenstellung der Erzeugung des 3-Loch-Dreiecksflanschs mittels Kommandos erfüllt. Worin liegt nun der Vorteil einer Konstruktion mittels Kommandos gegenüber einer per Menü-Dialog interaktiven Methode? Die Antwort ist natürlich einfach. Die Kommandobasierte ist zwar zunächst aufwendiger, lässt sich aber letzten Endes voll automatisieren, die interaktive benötigt immer den Eingriff eines Benutzers.

Nun gilt es also unsere Kommando-Einzelschritte zu einer "Automatisierung" zusammenzufassen. In einer ersten Stufe hilft dabei der Kommando-Stapelspeicher (Kommandostack). Alle oben zur Erzeugung des Flanschs eingegebenen Kommandos (einschließlich eventueller, fehlerhafter Eingaben) wurden (unsichtbar durch den Benutzer) automatisch in den Kommandostack geschrieben. Diese Kommandofolge kann jetzt in eine Datei gespeichert und natürlich auch nachträglich editiert werden. Die Speicherung erfolgt mit dem "csave"-Kommando und der Angabe eines Dateinamens, also z.B.

```
csave test
```

Dadurch ist jetzt eine Datei "test.stk" im **Pictures**-Hauptverzeichnis erzeugt worden, auf die wir anschließend noch zugreifen werden.

Nun besagt der Titel dieser Übungslektion ja schon, dass unsere Kommandofolge in einem eigenständigen "Programm" genutzt werden soll. In unserem Fall wollen wir zunächst eine Prozedur, also ein Programm in der Pictures-eigenen Kommandosprache erzeugen. Dabei darf der zu vergebende Dateiname nicht mit existierenden Kommando- bzw. Programmnamen kollidieren.

Beispielsweise wollen wir für unser Programm momentan einmal den Namen "fltest" verwenden (genaugenommen "fltest.prc"). Um zu prüfen, ob ein solches Kommando oder ein so benanntes Programm existieren, geben Sie in die Kommandozeile

```
fltest
```

ein und quittieren mit der <ENTER>-Taste. Wir erwarten, dass die Meldung

"fltest: unbekanntes Kommando" in der Infozeile erscheint. D.h. also, dass weder ein namensgleiches Kommando oder Programm schon existiert. Genau das erwarten wir, denn wir wollen ja "fltest" als neue Kommandoprozedur erstellen.

Randbemerkung: Sollte eine Datei "fltest.prc" bereits existieren, sollte sie natürlich zuvor umbenannt oder gelöscht werden. Löschen z.B. mit

```
rm "C:\Program Files\SchottSysteme\PicturesByPC\Procs\OK_in_30\fltest.prc"
```

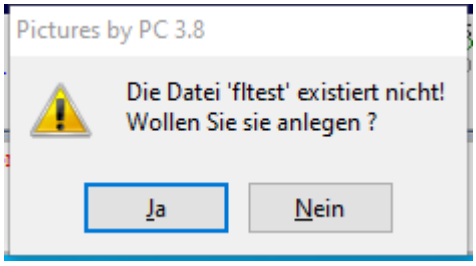
oder

```
rm "C:\Program Files\SchottSysteme\PicturesByPC\Procs\samples\fltest.prc"
```

Sofern also ein Kommando "fltest" (Kommandoprozedur) noch nicht existiert, wird durch das Kommando

```
proc fltest
```

folgende Meldung angezeigt



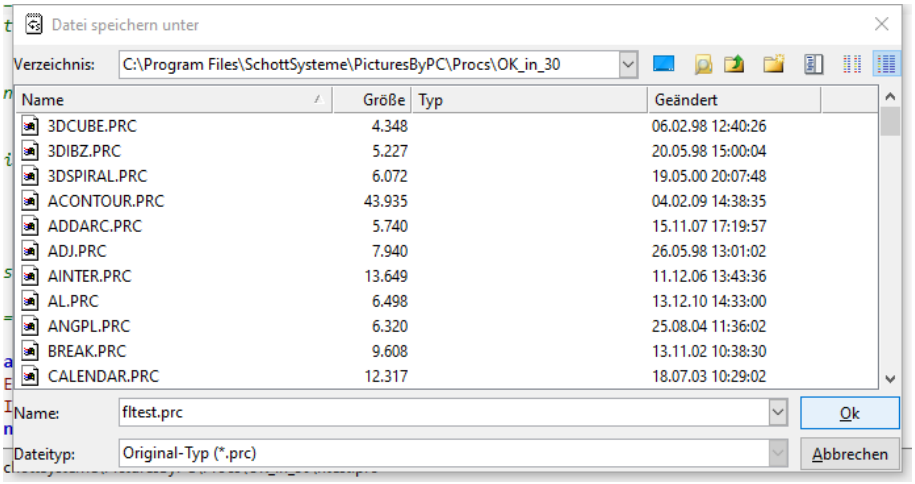
Diese quittieren Sie mit <ENTER> bzw. <Alt+J> oder klicken das Feld "Ja" mit der Maus anklicken. Daraufhin wird der **Pictures**-Texteditor aufgerufen und öffnet ein (leeres) Grundschema für unsere "fltest"-Prozedur. In dieses "Prozedur-Formular" fügen Sie vor der Zeile "goto End -f" jetzt einige Zeilenumbrüche ("leere Zeilen") ein, denn an dieser Stelle müssen unsere Programmzeilen eingefügt werden.

Um einen ersten Entwurf zu generieren, öffnen Sie jetzt mit

```
proc "test.stk"
```

die oben erzeugte Kommandostack-Datei "test.stk". Die zu nutzende erste Kommandozeile muss ja, wie Sie wissen " arc help -c0,0 -r37.5 -p3 -s-30..-30" lauten. Suchen Sie sie also durch "Blättern" oder mit der Suchfunktion (<Strg>+<F>). Markieren Sie jetzt den Text einschließlich dieser Zeile bis zum Dateiende. Am einfachsten geschieht dies mit der Tastenkombination <Strg>+<Umschalt>+<Ende> (oder englischsprachig <Ctrl>+Shift>+<End>). Den Text kopieren Sie jetzt mit <Strg>+<C> in die Zwischenablage, wechseln in die Prozedur-Datei "fltest.prc" und fügen den Text aus der Zwischenablage mit <Strg>+<V> vor der Zeile mit "goto End -f" wieder ein.

Natürlich muss der Text, um die Funktionsfähigkeit der Prozedur zu garantieren, noch editiert werden. Vorerst sichern Sie aber schon einmal den aktuellen Stand in das "procs"-Unterverzeichnis "Samples". Dazu klicken Sie im textuellen Menu im Editor auf "Datei" und "Speichern unter" (oder alternativ <Strg>+<Alt>+<S>) und sehen folgenden Dialog.



Das vorgegebene Verzeichnis "C:\Program Files\SchottSysteme\PicturesByPC\Procs" ist für "professionelle Prozeduren" im Grunde schon der richtige Vorschlag,. Übungsprozeduren, wie unsere, sollten allerdings vorerst einmal zu Versuchszwecken in das Procs-Unterverzeichnis "Samples" kopiert werden. Deshalb müssen wir in der Verzeichnisstruktur zunächst mit  eine Stufe zurück gehen und danach ins Unterverzeichnis "samples" umschalten, um unsere Datei "fltest.prc" dort zu speichern. Da der Text der Kommandofolge vermutlich fehlerbehaftet, auf jeden Fall aber unvollständig ist, muss er editiert werden. Beachten Sie z.B. auch, dass im Kommando-Stack jeweils ein Kommando nur einmal vorkommt. Eine Zeile wie etwa "cat * tri -i" muss also mehrfach kopiert und eingefügt werden. Am einfachsten gelingt das im Editor mit <F7> (Zeileninhalt "merken") und an der gewünschten Stelle mit <F8> (Einfügen der Zeile). Komplett fehlerhafte Zeilen löschen Sie am besten mit <F9> (genaugenommen "merken" und "löschen", also mit <F8> wieder einfügbar) und andere Fehler korrigieren Sie mit den üblich Editier-Möglichkeiten. Am Ende sollte der Text wie folgt aussehen:

```
arc help -c0,0 -r37.5 -p3 -s-30..-30
eq * -w15 -sr -d --nod
cover * tri
box *
arc hole* -tc0,37.5 -r7
cat * tri -i
rot hole1 hole* 120 -c0,0
cat * tri -i
rot hole2 hole* 120 -c0,0
cat * tri -i
arc icirc -tc0,0 -r17.5
cat * tri -i
box tri
arc ocirc -tc0,0 -r22.5
extrude3d tri -h10 --nod
name * tribase
setpp -i
extrude3d icirc ocirc -h10..45 --nod
bop3d * tribase -u
name * flange*
del help
del tri
del ocirc
```

Das ist letzten Endes nichts anderes als die vollständige Liste aller unserer oben eingegebenen Kommandos in der richtigen Reihenfolge und mit den Wiederholungen. Natürlich müssen Sie die geänderte und geprüfte Datei noch sichern. Das können Sie mit Anklicken von  (Diskettensymbol) oder mit <Str>+<S> machen.

Bevor Sie die "Prozedur" auf ihre Funktionsfähigkeit prüfen, sollten Sie den Puffer löschen, z.B. mit

```
new -p
```

Wenn Sie dann in die Pictures-Kommandozeile "fltest" eingeben und mit der <ENTER> Taste abschließen

```
fltest
```

wird der Flansch "automatisch" erzeugt. Das ist zwar schon recht nett, aber von einer "richtigen" Prozedur erwarten wir doch wohl erheblich mehr.

Die erste "Unzulänglichkeit" unserer "Prozedur" ist nämlich, dass jedes einzelne Kommando einen eigenen "undo"-Schritt erzeugt. Allerdings lässt sich das schnell beheben, indem vor unsere Zeilen ein

```
undo -b
```

vorangestellt und ans Ende vor dem "goto End -f" ein

```
undo -e -i Flansch
```

eingefügt wird. Auf diese Weise können alle Kommandoschritte zwischen "undo -b" (undo begin) und "undo -e" (undo end) durch ein "undo"-Kommando zurückgenommen werden. D.h. der Flansch würde nach einem "undo" wieder komplett gelöscht. Der Text "Flansch" wird mit der Option -i in die Undo-Liste eingetragen. Dies dient der besseren Identifikation bei mehrstufigem Undo.

Selbstverständlich soll unsere Prozedur auch variierende Abmessungen unseres 3-Loch-Dreiecksflansches generieren können. Dazu müssen natürlich Variable eingeführt und per Benutzerdialog abgefragt werden, auf die man an entsprechender Stelle im Programm wieder zugreifen kann

Exemplarisch soll das zunächst anhand der Abfrage für den Radius des 3-Loch-Führungskreises gezeigt werden. Wir definieren also z.B. eine Variable "pcr" (für "pitch circle radius") für den Lochkreis-Radius. Diese Variable soll nur innerhalb der Prozedur, also "lokal" (local), gültig sein. Dazu wird oben in der Prozedur nach der Zeile "trap Inter -i" eine neue Zeile

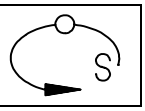
```
local pcr
```

eingefügt. Die Variable "pcr" ist damit als lokal definiert und kann im Weiteren genutzt werden. Lokale Variablen sind salopp gesprochen nichts Anderes als sheet-Zellen mit "handlicheren" Namen. Natürlich sollten Namensgleichheiten mit Kommandos sowie Standard- und Global-Variablen (selbstverständlich auch Sheet-Zellen-Bezeichnungen) vermieden werden.

Wenn man nun eine solche Variable eingeführt hat, muss man ihr natürlich auch Werte zuweisen. Exemplarisch machen wir das mit der früher schon benutzten BIX-Routine "dinput".

Unterhalb der "local" Zeile (ggf. mit einigen Leerzeilen) geben wir

```
dinput pcr, "Lochkreis-Radius eingeben:  " -d37.5 -e37.5
```



ein. Einen solchen Ausdruck haben wir in der vorherigen Lektion schon kennengelernt. Am Ende hat die Variable "pcr" den vom Benutzer eingegebenen oder bestätigten Wert ("-d" ist die Option für den Vorgabewert, "-e" ist die Option für den editierbaren Wert).

Nun muss der Wert der Variablen zusätzlich natürlich noch in die zugeordneten Kommandos eingefügt werden. Die Zeile "arc help -c0,0 -r37.5 -p3 -s-30..-30" muss also auf

```
arc help -c0,0 -r$pcr -p3 -s-30..-30
```

und die Zeile "arc hole* -tc0,37.5 -r7" auf

```
arc hole* -tc0,$pcr -r7
```

geändert werden. Selbstverständlich müssten auf vergleichbare Weise auch alle anderen zeichnungsrelevanten Werte abgefragt werden. Momentan stellen wir das aber noch zurück und testen erst einmal unsere Syntax.

Sichern Sie also die Prozedur mit den genannten Änderungen und rufen Sie sie erneut auf, dabei sollten Sie den Kommandostack und die "Pfeil"-Taste nutzen.

Es wird jetzt von Ihnen ein "vernünftiger" Wert für den Lochkreis-Radius angefordert und nach der Eingabe-Bestätigung sollte der Flansch mit dem entsprechenden Wert erzeugt werden. Sollte ein Fehler auftreten, korrigieren Sie die Zeile in der er auftritt (s. Variablen ERRPROCEDURE, ERRLINE, ERRNUMBER, ERRMESSAGE in der Prozedur-Sprungmarke "Error: ")

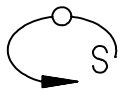
Nach den obigen Ergänzungen kann der Flansch zwar jetzt geometrisch variiert aber leider nicht mehr automatisch ausgeführt werden. Es wird immer eine Benutzereingabe verlangt. Für "Automatismen" ist das aber nicht gewünscht. Um das zu vermeiden, ergänzen wir die Prozedur mit einer Option "-r...", in der der Lochkreis-Radius schon beim Kommandoaufruf mit übergeben werden kann. In einem solchen Fall soll dann natürlich die Dialogabfrage entfallen

Um das richtig umzusetzen, führen wir beispielsweise eine Option "-r.." mit Parameter ein, in dem wir die Zeile "options new" z.B. zu

```
options new -r...
```

ergänzen (Minuszeichen, Optionsbuchstabe und 3 Punkte). Wenn wir jetzt beim Prozeduraufruf mit der Option "-r" einem Wert übergeben, soll dieser natürlich übernommen werden und andererseits der Abfrage-Dialog somit entfallen. Das erreichen wir im einfachsten Fall z.B. durch

```
if not option r
  dinput pcr, "Lochkreis-Radius eingeben: " -d37.5 -e37.5
else
  cal pcr=var('optval_r')
endif
```



Die Dialog-Abfrage ist hier also in eine "if-else-endif"-Bedingung eingeschlossen. Im Klartext bewirken obige Zeilen Folgendes. Sofern die Option "-r" nicht genutzt wird, erfolgt die Abfrage des Lochkreis-Radius per "dinput" mit dem Vorgabewert "37.5", der mit der Option "-e" auch zur Editierung angeboten wird. Anderenfalls wird der Radiuswert der Option "-r" entnommen (s. Standard-Variable OPTVAL_..).

Hinweise zu Syntax: Eine Fallentscheidung wird immer mit "if" eingeleitet und muss mit einem "endif" abgeschlossen werden. Eine Alternative wird mit "else", und mehreren "elif" eingeleitet. Details hierzu finden Sie in der F1-Hilfe unter dem "if"-Kommando, u.a. auch die Beschreibung zur Bedingungs-Syntax ("condition"). Innerhalb einer "if-endif"-Bedingung werden Kommandozeilen für die jeweiligen Bedingungen gewöhnlich etwas eingerückt, um die Programm-Lesbarkeit zu verbessern.

Zurück zu unserer Prozedur. Sichern Sie nun diese mit den obigen Änderungen und rufen sie unter Angabe der Radius-Option mit Parameter auf, z.B.

```
fltest -r50
```

Jetzt sollte die Benutzer-Abfrage unterdrückt und der Flansch mit einem Lochkreis-Radius von 50 erzeugt worden sein. Wenn Sie andererseits die Option "-r..." beim Kommandoaufruf weglassen, werden Sie erwartungsgemäß wieder nach dem Radius gefragt.

Sollte Ihnen in der Prozedur versehentlich ein Fehler unterlaufen, meldet **Pictures** alle notwendigen Informationen zur Fehleranalyse (s. Variablen ERRPROCEDURE, ERRLINE, ERRNUMBER, ERRMESSAGE).

Hinweis: Grundsätzlich "springt" der Editor im Fehlerfall durch Eingabe des Kommandos

```
proc -e
```

direkt in die fehlerhafte Zeile. Das ist insbesondere bei umfangreichen Prozeduren sehr hilfreich.

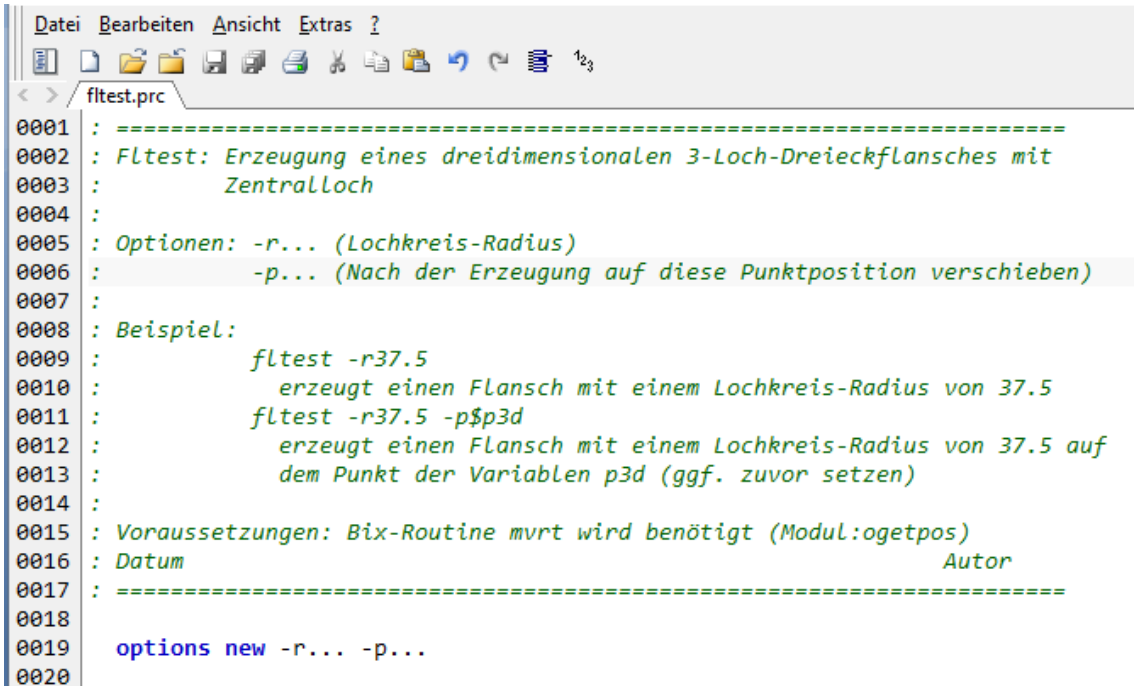
Für eine eventuelle Fehleraustestung, kann es außerdem sehr nützlich sein, mit dem Kommando "**goto End -f**" einen Programmabbruch in einer bestimmten Programmzeile zu erzwingen, um ggf. davor mit "**echo**" einen Wert zu überprüfen, z.B.

```
echo Wert: $pcr
```

Vorausgesetzt Ihre Prozedur "fltest" läuft nun fehlerfrei, ist es spätestens jetzt angeraten, die Wirkung und Bedienung der Prozedur im Kopf der Prozedur zu beschreiben. Prinzipiell wird jede Programmzeile, die mit einem Doppelpunkt und einem Leerzeichen ": " beginnt als Kommentartext interpretiert. Davon sollten Sie innerhalb Ihres Programms reichlich Gebrauch machen, um einzelne Sequenzen verständlich zu erklären.

Für die grundsätzliche Prozedur-Beschreibung empfehlen wir zu Beginn des Programms einige Kopfzeilen, die zumindest die Wirkung und die Bedienung (inkl. Optionen) für

den Benutzer erläutern, anzulegen. Die Kopfzeilen für unseren Fall bis zum jetzigen Programmstand könnten etwa wie in der Abbildung unten aussehen.



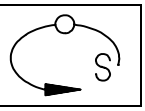
Jede Ergänzung oder Parameter-Änderung in der Prozedur sollten Sie in den Kopfzeilen möglichst dokumentieren, damit jeder Bediener die "neuen, tollen" Funktionen nutzen kann.

Bislang ist unsere Prozedurbeschreibung und Bedienung ausschließlich deutschsprachig verfasst. Prinzipiell war **PicturesByPC** selbst unter DOS-Betriebssystem schon zweisprachig, nämlich deutsch und englisch. Sofern also jemand seine Prozedur auch englischsprachigen Benutzern erschließen will, kann er dies mit einigen, einfachen Ergänzungen erreichen.

Für jede Zeile, die deutsche Texte enthält, muss ein entsprechendes Duplikat mit englischsprachigem Text erstellt werden. Dazu ist lediglich an den Anfang der Zeile entweder ein "?D" oder ein "?E" voranzustellen. Dies gilt selbstverständlich auch für alle Kommentare. Dies wird hier z.B. an unserem "dinput"-Kommando gezeigt.

```
?D  dinput pcr, "Lochkreis-Radius eingeben: " -d37.5 -e37.5
?E  dinput pcr, "Enter pitch circle radius : " -d37.5 -e37.5
```

Wir wollen die Sprachanpassung an dieser Stelle nicht weiter verfolgen, da daraus keine neuen Erkenntnisse zu gewinnen sind.



Unsere Prozedur schien bislang ja ganz brauchbar zu funktionieren. Das lag natürlich an halbwegs "sinnvollen" Eingabewerten durch Sie als kundigen Bediener. Aber es leuchtet ja wohl ein, dass bestimmte Eingabewerte wie z.B. Null ("0") für unsere geometrische Anordnung keinen Sinn macht. Unsere Prozedur würde in diesem Fall mit einer Fehlermeldung "fltest: fehlerhafte Option -r" sofort beendet werden.

Daher ergänzen wir unsere Eingabe-Sequenz noch mit einer "minimalen" Fehlerbehandlung. Eine komplett schlüssige Fehleranalyse, die alle möglichen Geometrie-Fehler abfängt ist bei einer einzeiligen Dialog-Eingabe natürlich nicht zu realisieren. Schließlich kann die nächste "dinput"-Abfragen erst folgen, nachdem die aktuelle Eingabe abgeschlossen ist. Salopp gesprochen kann unser Lochkreis-Radius noch nicht "wissen", welche Werte in der Folge noch eingegeben werden und welche dabei "Konflikte" schaffen können.

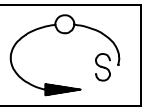
An dieser Stelle der Prozedur kann also nur eine nach unten begrenzte Limitierung oder Text- Eingabe fehlermäßig behandelt werden. Unsere bisherige "if"-Bedingung ersetzen wir jetzt durch

```
local lim pcr
...
cal lim=0.1
...
  if option r then cal pcr=var('optval_r')
Inp1:
  if cal val(pcr)<lim
?D  dinput pcr, "Lochkreis-Radius eingeben: " -d37.5 -e37.5 -s2 -lEnd
?E  dinput pcr, "Enter pitch circle radius : " -d37.5 -e37.5 -s2 -lEnd
  endif

  if cal val(pcr)<lim
     beep
?D     echo Ungültiger Lochkreis-Radius
?E     echo Wrong pitch circle radius
     goto Inp1 -b
  endif
```

Aus Gründen der einfachen Lesbarkeit wurde in diesem Fall eine "sehr einfache" Variante mit drei "if"-Anweisungen und einer neu eingeführten Sprungmarke "Inp1:" gewählt.

Betrachten wir in den Programmzeilen oben die erste "if-then"-Bedingung, so bewirkt sie, dass sofern ein Optionswert im Kommandoaufruf angegeben wurde, dieser der



Variablen "pcr" zugewiesen wird. Prinzipiell könnte dieser fälschlich aber auch noch unsinnige Werte (Negativzahlen, kleiner als Grenzwert "lim" oder Texte) zuweisen.

In der zweite Bedingung wird abgeprüft, ob die Variable "pcr" einen "unvernünftigen" Wert kleiner als das Limit "lim" hat. Ist das der Fall, wird der Wert mit "dinput" abgefragt. Beachten Sie, dass in diesem Kommando die Option "-s2" ergänzt wurde. Diese besagt, dass keine Texte sondern nur numerische Werte eingegeben werden können. Letztere lassen sich übrigens im Dialog mit der Maus rauf und runter schalten (Spin num edit). Zusätzlich wird mit der Option "-lEnd" bei Betätigen des "Abbrechen"-Buttons zur Sprungmarke "End" gesprungen und somit die Prozedur gezielt beendet.

Die dritte Bedingung prüft noch einmal die "Sinnhaftigkeit" beider Wertzuweisungen (<lim) an die Variable "pcr". Ist diese nicht gegeben, wird ein Piepton erzeugt (beep), eine echo-Meldung ausgegeben und rückwärts (-b back) zur Sprungmarke "Inp1:" zurückgesprungen, um die Eingabe mit "brauchbaren" Werten zu wiederholen.

Wie oben schon erwähnt, lassen sich an dieser Stelle in der Prozedur mit der Eingabe einer einzelnen geometrischen Größe keine eventuell noch folgenden "Geometrie-Kollisionen" verhindern. Wir müssen also auf einen "kundigen" Benutzer hoffen, der realistische Werte wählt.

Nebenbemerkung zum Editor: Im Pictures-Text-Editor gibt es unter "Extra" für die Spalten-Bearbeitung hilfreiche Funktionen. Wollen Sie also z.B. mehrere Zeilen einrücken, markieren Sie mit der <Alt> und den Pfeiltasten die erste Spalte aller gewünschten Zeilen, klicken im Menü "Extra", danach "Spalten-Text einfügen" an, geben z.B. zwei Leerzeichen an und rücken so Ihre Zeilen entsprechend ein (alternativ: <Alt>+<E>+<S>) Andere Spalten-Operationen unter "Extra" (wie z.B. löschen, angleichen, ersetzen etc.) sollten Sie selbst ausprobieren.

Bevor wir die weitere Vorgehensweise besprechen, könnte noch eine nette Ergänzung unserer Prozedur diskutiert werden. Unser Flansch wird zunächst parametrisch auf dem Nullpunkt erzeugt. Möchte man ihn jedoch an eine andere Position verschieben, bieten sich dafür zwei Möglichkeiten an. Dazu fügen wir in unser Prozedur vor der "undo -e" - Zeile folgende Sequenz ein

```
(nicht vergessen zuvor:    options new -r... -p...)
...
  if option p
    scale3 * -md0,0,0..$optval_p
  else
    mvrt * -3n0,0,0 -p
  endif
```

Wenn man jetzt beim Kommandoaufruf mit der Option -r einen 3D-Punkt angibt, z.B.

```
fltest -p 100,80,50
```

wird der Flansch-Nullpunkt nach der Erzeugung auf diese Punktposition 100,80,50 verschoben.

Lässt man jedoch die Option weg, wird der Flansch mit der Bix-Routine "mvrt" (aus dem Modul "ogetpos") am Fadenkreuz hängend interaktiv platziert. Dabei kann er mit den Tasten "Bild nach oben" bzw. "Bild nach unten" auch noch um die Senkrechte gedreht und mit der Angabe eines Faktors auch noch skaliert werden

Eine weitere Möglichkeit bestünde natürlich auch darin, dass man den Flansch auf einen zuvor mit abgegriffenen oder in der Variablen "p3d" gespeicherten Punkt verschieben will, dann würde man beim Kommandoaufruf von fltest mit der Option -p auch den Inhalt der Variablen p3d mit übergeben, also "-p$p3d".

Probieren Sie ggf. alle Varianten aus.

Im Prinzip müssten Sie jetzt in der Lage sein, die Abfrage-Dialoge für die übrigen Zeichnungsparameter selbst zu ergänzen. Das sollten Sie nun auch versuchen, um eigene Erfahrungen zu machen. Im Anhang dieser Lektion wird noch eine "vollständige" Prozedur "Fltest2.prc" gelistet, deren Lektüre Ihnen ggf. weiterhelfen kann.

Vergessen Sie niemals oben im Programm den Options-Buchstaben, den lokalen Variablen-Namen zu ergänzen und im ausführenden Kommando die explizite Zahl durch die Variable mit vorangestelltem $-Zeichen zu ersetzen. Zeigen wir das hier exemplarisch noch ein letztes Mal

```
: Radius des Durchganglochs
  if option i
    cal ird=var('optval_i')
  endif

Inp2:
  if cal val(ird)<lim
?D  dinput ird, "Radius des Durchgangsloches eingeben: " -d17.5 -e17.5 -s2 -lEnd
?E  dinput ird, "Enter inner circle radius : " -d17.5 -e17.5 -s2 -lEnd
  endif

  if cal val(ird)<lim
     beep
?D     echo Ungültiger Radius des Durchgangslochs
?E     echo Wrong inner circe radius
     goto Inp2 -b
  endif
```

Vergessen Sie also auch nicht die Zeile "options" und "local" zu ergänzen

```
options -r... -i...
```

bzw.

```
local lim pcr ird
```

sowie auch weiter unten im Programm

```
arc icirc -tc0,0 -r$ird
```

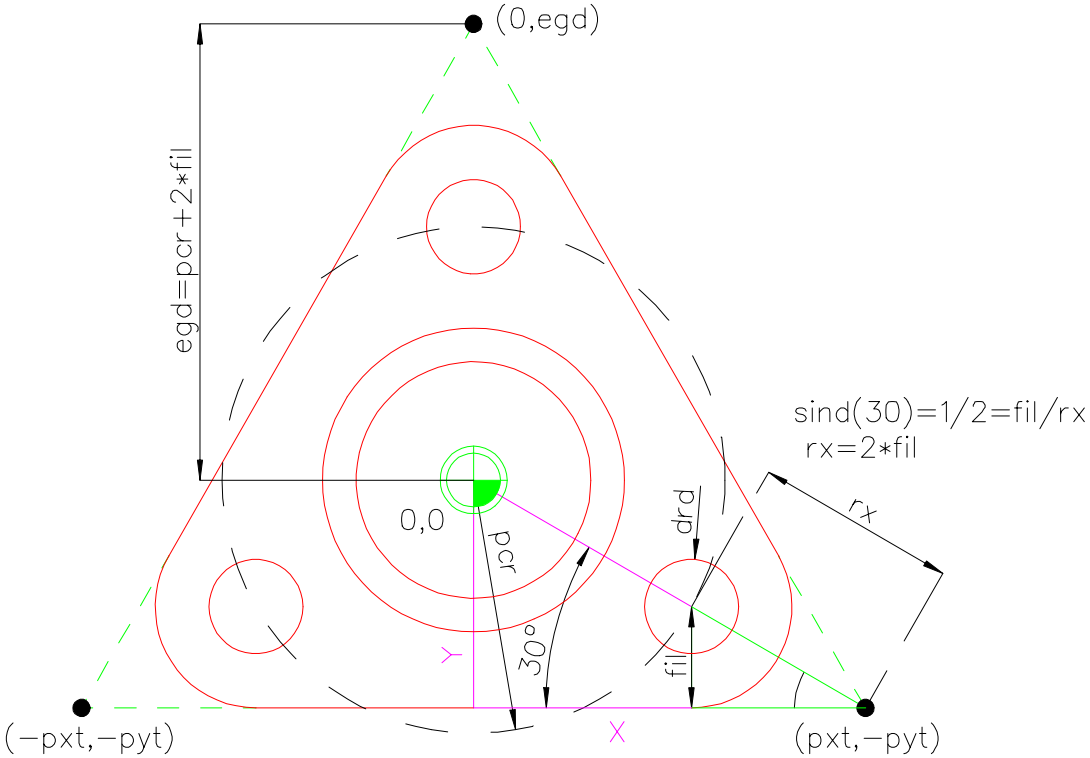
Da die Ergänzung der noch fehlenden Dialoge im Prinzip keine neuen Erkenntnisse ergibt, überspringen wir die Restlichen und überlassen Ihnen diese Fleißarbeit.

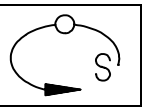
Sofern Sie die Parameter-Abfragen in die obengenannte Prozedur "fltest.prc" einbauen, ersparen Sie sich durch den "Konstruktions-Trick" mit dem "Dreiecks-Kreis" und der Nutzung der variantenreichen BIX-Routine "EQ.BIX" natürlich komplexere Geometrieberechnungen.

So einfach ist das im Allgemeinen jedoch nicht. Grafische Darstellungen lassen sich gewöhnlich nicht ohne mathematische Berechnungen mit den wichtigsten Rechenarten und Funktionen erzeugen. Besonders wichtig sind u.a. die Winkel-Funktionen.

Darum wird in der Prozedur "fltest2.prc" der Flansch bewusst mit einem "poly"-Kommando aus "selbst" berechneten Punkten erzeugt. Das ist natürlich anspruchsvoller als zuvor mit dem "arc ..-p3". In unserem Fall wollen wir den Flansch aus einem gleichseitigen Dreieck mit abrundendem Trimmen zu erzeugen.

Für die folgende Erklärung müssen Sie daher in der Abbildung 2 den "bemassten" Flansch rechts unten betrachten.





Von Interesse ist u.a. die Strecke rx (des kleinen, grünen Dreiecks), die die Gerade vom Mittelpunkt (0,0) zum rechten, unteren Eckpunkt über den Lochkreis-Radius pcr hinaus verlängert. Um diese zu berechnen, betrachten wir das kleine, grüne, rechtwinklige Dreieck mit dem Öffnungswinkel von 30° (rechts unten in der Zeichnung). Dessen Gegenkathete fil entspricht dem Radius der Eckenrundung, der uns als Parameter ja bekannt ist. Ermitteln wir den Sinus an diesem kleinen Dreieck, so lautet der

```
sin(30°)=Gegenkathete/Hypotenuse=fil/rx=1/2
```

da der Sinus von 30° bekanntlich 1/2 ist. In Pictures-Syntax mit der Sinus-Funktion im Grad-Maß lautet das

```
sind(30)=1/2=fil/rx
```

d.h.

```
rx=2*fil
```

Fazit ist also in einem solchen Dreieck ist die Hypotenuse (hier: rx) zweimal so groß wie die Gegenkathete (hier: fil).
Betrachtet man jetzt das gleichgeartete, magenta-farbene, große Dreieck vom Null- bis zum Eckpunkt und dem Fußpunkt des Flansches, so kann man mit der obigen Erkenntnis den Y-Wert für einen unteren Eckpunkt entweder berechnen mit

```
Y=(pcr+2*fil)/2=pcr/2+fil
```

oder auch gleichwertig durch Berechnung mit dem Sinus

```
sin(30°)=Y/(pcr+2*fil)
```

also

```
Y=(pcr+2*fil)*sin(30°)
```

Um den X-Wert, der die Ankathete in dem Dreieck darstellt, zu berechnen, benutzen wir den Kosinus. Der Ausdruck lautet dann

```
cos(30°)=Ankathete/Hypotenuse=X/(pcr+2*fil)
```

also

```
X=(pcr+2*fil)*cos(30°)
```

Genau diese Formeln werden im Programm "fltest2" in folgender Sequenz genutzt

```
: Hilfsvariable zur Vereinfachung
  cal pxt=($pcr+2*$fil)*cosd(30)
  cal pyt=($pcr+2*$fil)*sind(30)
  cal egd=($pcr+2*$fil)
```

Dabei ist, bezogen auf den Nullpunkt, pxt der x-Absolutwert und pyt der y-Absolutwertes eines Punkte auf der Basis des Flansch-Dreiecks. Der dritte Wert egd ist der y-Wert des obersten Dreieckspunktes. Diese drei Hilfsvariablen wurden nur eingeführt, um das folgende Polygon-Kommando etwas übersichtlicher zu gestalten. Achten Sie im Kommando besonders auf die unterschiedlichen Vorzeichen.

```
poly help -cp -$pxt,-$pyt,$pxt,-$pyt,0,$egd
```

Damit haben wir jetzt das Ausgangsdreieck für unseren Flansch mit numerischer Koordinaten-Berechnung, anders als in der Prozedur "fltest", selbst erzeugt.

Jetzt gilt es noch die spitzen Ecken mit dem trim-Kommando abzurunden. Das soll mit dem Trimmen mit Ausrundungsradius (Option -f..) vollzogen werden und geschieht mit der Sequenz

```
: Punktabstand fürs Trimmen
  cal trd=lim/10

: Eckenrundung durch Trimmen, wenn Wert für Eckenradius größer 0 ist
  if not cal val($fil)==0
:   Trimmen Punkt links
    trim help -f$fil -p-$[$pxt-$trd],-$[?$pyt-$trd],-$[?$pxt-$trd],-$pyt
:   Trimmen Punkt rechts
    trim help -f$fil -p$[?$pxt-$trd],-$[?$pyt-$trd],$[?$pxt-$trd],-$pyt
:   Trimmen Punkt oben
    trim help -f$fil -p1,$[?$egd-$trd],-$trd,$[?$egd-$trd]
  endif
```

Zunächst wird darin ein Punktabstand trd "relativ willkürlich" auf ein Zehntel des kleinsten Grenzwerts der Abmessungen gesetzt. Sie wissen ja vom interaktiven Trimmen, das ein Trim-Punkt nur "ein wenig" vom Eckpunkt versetzt auf der jeweiligen Geraden "angeklickt" werden muss. Dieser "Abstand" trd wird in den Programmzeilen oben jeweils zu der Punktkoordinate addiert oder von ihr abgezogen.

Zum Trimmen einer Ausrundung müssen dann jeweils die angrenzenden Schenkel eines Eck-Scheitelpunktes "angeklickt" werden. Im Trim-Kommando erfolgt das mit der Option -p ..., während mit der Option -f fil der Ausrundungsradius angegeben wird.

Damit ist die wichtigste Passage unseres "fltest2"-Programms, nämlich die der expliziten numerischen Berechnung der Punktkoordinaten abgeschlossen.

Die übrigen Programmzeilen sind uns ja weitgehend bekannt und sollten kein Verständnisproblem bereiten.

Wie Sie leicht erkennen können, haben wir in diesem Programm für die Variablen "drd" und "fil" durch abweichende "if"-Anweisung allerdings auch zugelassen, dass die Flanschecken ggf. nicht ausgerundet werden sollen und die Bohrlöcher unterdrückt werden können. Probieren Sie das selbst einmal aus.

Fazit:

Sobald ein Pictures-Benutzer Kommandos, elementare Mathematik und die einfache Prozedursprachensyntax beherrscht, erschließen sich ihm unglaubliche, neue Möglichkeiten für seine tägliche Arbeit.

Ausblick:

Diese Lektion befasste sich lediglich mit der elementarsten Einführung in die Programmierung. Bedenken Sie, dass **Pictures by PC** zusätzlich mit seinem eigenständigen Basic (BIX) eine komplette, objektorientierte, grafische Entwicklungs-umgebung für Windows bereitstellt, die nahezu jeglicher Aufgabe gewachsen ist.

Unsere Empfehlung seitens **SCHOTT SYSTEME** ist daher, nutzen Sie unseren Support und unsere Seminarreihen (u.a. Programmierung) um sich weiterzubilden. Wir helfen Ihnen dabei, Ihre Aufgaben selbständig zu lösen und Ihre tägliche Arbeit zu erleichtern.

Denken Sie daran

WER NICHT AUTOMATISIERT, DER VERLIERT.

Anhang: Programm fltest2.prc

```
: ======================================================================
: Fltest2: Erzeugung eines dreidimensionalen 3-Loch-Dreicksflansches mit
:         Zentralloch
:
: Optionen: -r... (Lochkreis-Radius)
:           -i... (Radius des Durchgangslochs)
:           -t... (Rohrwandstärke)
:           -b... (Materialdicke der Basisplatte
:           -h... (Gesamthöhe des Flansches)
:           -f... (Eckenradius)
:           -d... (Bohrloch-Radius)
:
:           -p... (Nach der Erzeugung auf diese Punktposition verschieben)
:
: Beispiel:
:           fltest2 -r37.5
:             erzeugt einen Flansch mit einem Lochkreis-Radius von 37.5
:           fltest2 -r37.5 -i17.5
:           fltest2 -r37.5 -i17.5 -t5
:           fltest2 -r37.5 -i17.5 -t5 -b10
:           fltest2 -r37.5 -i17.5 -t5 -b10 -h45
:           fltest2 -r37.5 -i17.5 -t5 -b10 -h45 -d7
:           fltest2 -r37.5 -i17.5 -t5 -b10 -h45 -d7 -f15
:
:           fltest2 -r37.5 -i17.5 -t5 -b10 -h45 -d7 -f15 -p$p3d
:
: klein     fltest2 -r3.75 -i1.75 -t0.5 -b1 -h4.5 -d0.7 -f1.5
:
: Voraussetzungen: ......
: Datum                                                          Autor
: ======================================================================

  options new -r... -i... -t... -b... -h... -d... -f... -p...

  codepage pictures
  trap Error
  trap Inter -i

  local lim pcr ird thi bth fhi drd fil pxt pyt egd trd

: Kleinster Grenzwert für die Geometrie-Abmessungen
  cal lim=0.1

: Lochkreis-Radius
  if option r
    cal pcr=var('optval_r')
  endif

Inp1:
  if cal val(pcr)<lim
?D  dinput pcr, "Lochkreis-Radius eingeben: " -d37.5 -e37.5 -s2 -lEnd
?E  dinput pcr, "Enter pitch circle radius : " -d37.5 -e37.5 -s2 -lEnd
  endif

  if cal val(pcr)<lim
     beep
?D     echo Ungültiger Lochkreis-Radius
?E     echo Wrong pitch circle radius
     goto Inp1 -b
```

```
  endif

: Radius des Durchganglochs
  if option i
    cal ird=var('optval_i')
  endif

Inp2:
  if cal val(ird)<lim
?D  dinput ird, "Radius des Durchgangsloches eingeben: " -d17.5 -e17.5 -s2 -lEnd
?E  dinput ird, "Enter inner circle radius : " -d17.5 -e17.5 -s2 -lEnd
  endif

  if cal val(ird)<lim
     beep
?D     echo Ungültiger Radius des Durchgangslochs
?E     echo Wrong inner circe radius
     goto Inp2 -b
  endif

: Rohrwandstärke
  if option t
    cal thi=var('optval_t')
  endif

Inp3:
  if cal val(thi)<lim
?D  dinput thi, "Rohr-Wandstärke eingeben: " -d5 -e5 -s2 -lEnd
?E  dinput thi, "Enter pipe thickness: " -d5 -e5 -s2 -lEnd
  endif

  if cal val(thi)<lim
     beep
?D     echo Ungültige Rohr-Wandstärke
?E     echo Wrong pipe thickness
     goto Inp3 -b
  endif

: Basisplatten-Wandstärke
  if option b
    cal bth=var('optval_b')
  endif

Inp4:
  if cal val(bth)<lim
?D  dinput bth, "Dicke der Basisplatte eingeben: <10> " -d10 -e10 -s2 -lEnd
?E  dinput bth, "Enter base plate thickness : " -d5 -e5 -s2 -lEnd
  endif

  if cal val(bth)<lim
     beep
?D     echo Ungültige Wandstärke der Basisplatte
?E     echo Wrong base plate thickness
     goto Inp4 -b
  endif

: Flansch-Gesamthöhe
  if option h
    cal fhi=var('optval_h')
  endif

Inp5:
  if cal val(fhi)<lim
```

```
?D  dinput fhi, "Gesamthöhe des Flanschs eingeben: <45> " -d45 -e45 -s2 -lEnd
?E  dinput fhi, "Enter maximum flange height: " -d5 -e5 -s2 -lEnd
  endif

  if cal val(fhi)<lim
     beep
?D     echo Ungültige Flanschhöhe
?E     echo Wrong flange height
     goto Inp5 -b
  endif

: Bohrloch-Radius
  if option d
    cal drd=var('optval_d')
  endif

Inp6:
   if $drd==
?D  dinput drd, "Bohrloch-Radius eingeben: " -d7 -e7 -s2 -lEnd
?E  dinput drd, "Enter drill radius: " -d7 -e7 -s2 -lEnd
    endif

: Eckenradius
  if option f
    cal fil=var('optval_f')
  endif

Inp7:
   if $fil==
?D  dinput fil, "Ecken-Radius eingeben: " -d15 -e15 -s2 -lEnd
?E  dinput fil, "Enter fillet radius: " -d15 -e15 -s2 -lEnd
  endif

  if cal val(fil)<$drd
     beep
?D   echo Der Eckenradius $fil muss >= dem Bohrlochradius $drd sein
?E   echo The fillet radius $fil must be >= then the drill radius $drd
     cal fil=''
     goto Inp7 -b
  endif

: Undo-Beginn
  undo -b

: Hilfsvariable zur Vereinfachung
  cal pxt=($pcr+2*$fil)*cosd(30)
  cal pyt=($pcr+2*$fil)*sind(30)
  cal egd=($pcr+2*$fil)

: Dreieck aus Polygon mit Winkel-Funktionen
  poly help -cp -$pxt,-$pyt,$pxt,-$pyt,0,$egd

: Punktabstand fürs Trimmen
  cal trd=lim/10

: Eckenrundung durch Trimmen, wenn Wert für Eckenradius größer 0 ist
  if not cal val($fil)==0
:   Trimmen Punkt links
    trim help -f$fil -p-$[$pxt-$trd],-$[?$pyt-$trd],-$[?$pxt-$trd],-$pyt
:   Trimmen Punkt rechts
    trim help -f$fil -p$[?$pxt-$trd],-$[?$pyt-$trd],$[?$pxt-$trd],-$pyt
:   Trimmen Punkt oben
    trim help -f$fil -p1,$[?$egd-$trd],-$trd,$[?$egd-$trd]
```

```
  endif

: Makro-Objekt tri
  cover * tri
  box *

: Bohrloch-Kreise, sofern drd nicht Null ist
  if not cal val($drd)==0
    arc circ* -tc0,$pcr -r$drd
    cat * tri -i
    rot circ1 circ* 120 -c0,0
    cat * tri -i
    rot circ2 circ* 120 -c0,0
    cat * tri -i
  endif

: Zentralloch-Kreis
  arc icirc -tc0,0 -r$ird
  cat * tri -i
  box tri

: Zentraloch-Kreis außen
  arc ocirc -tc0,0 -r$[?$ird+$thi]

: Extrusion der Dreiecksplatte
  extrude3d tri -h$bth --nod
  name * tribase
  setpp -i

: Extrusion des Rohransatzes
  extrude3d icirc ocirc -h$bth..$fhi --nod

: Vereinigung von Rohr und Platte
  bop3d * tribase -u

: Benennung zu flange...
  name * flange*

: Löschen der Hilfsobjekte
  del help
  del tri
  del ocirc

: Undo-Ende
  undo -e

  if option p
:   mit der Option -p eine 3D-Punktkoordinate übergeben, um den Flansch-
Nullpunkt dahin zu verschieben
    scale3 * -md0,0,0..$optval_p
  else
:   Platzieren mit Fadenkreuz (mvrt: BixAlias-Kommando (Modul 'ogetpos'))
    mvrt * -3n0,0,0 -p
  endif


  goto End -f

Inter:
  trap Ignore -i
  trap Error -f
  error 999 ": unterbrochen"
```

```
Error:
  trap Ignore -i
  trap off
  call -f Reset
  error -p $errprocedure -l$errline $errnumber ": $errmessage[2..]"

End:
  trap Ignore -i
  trap off
  call -f Reset
  chain

Reset:
  return
```