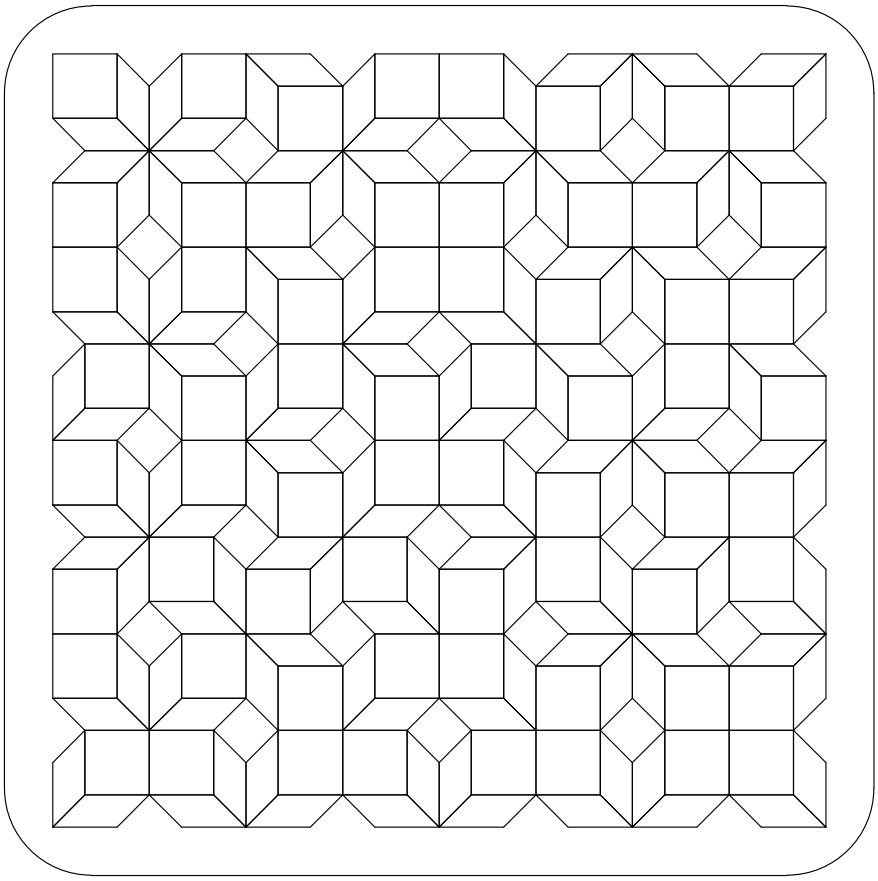
PIC



Eine Sprache
zur
grafischen Aufbereitung
von
Texten

PIC

Eine Sprache zur
grafischen Aufbereitung
von Texten

28. September 2005

Andreas Pidde

Entstanden im Rahmen einer Studienarbeit
unter der Leitung von
Prof. Dr. H. Giesen
an der
Universität Koblenz/Landau, Abt. Koblenz
Rheinau $3-4$
56075 Koblenz

Institut für Informatik

Vorwort

Grafiken in Texte einbinden ist, hat man nicht gerade ein Desktop Publishing System und entsprechende Mal– und Scannsoftware zur Verfügung, immer eine kritische Angelegenheit. Zwar kann (insbesondere auf dem Amiga) fast jedes Textverarbeitungsprogramm Pixel–Grafiken einbinden, aber die Qualität läßt oft noch Wünsche offen, sprich: Die Pixel sind zu grob. Abhilfe schaffen Vektorgrafiken, bei denen dafür oft der Text in den Bildern Schwierigkeiten macht. Mit PIC wird versucht Vektorgrafiken mit Text zu vereinen. Anders als bei Zeichenprogrammen wird das Bild nicht interaktiv eingegeben, sondern durch ein Programm spezifiziert. Bei der Sprachdefinition von PIC wurde besonders darauf geachtet, daß Bilder möglichst unabhängig von Koordinatenangaben aufgebaut werden können. Die Sprache stellt einige grundlegende Grafikobjekte (Primitive) zur Verfügung, die untereinander kombiniert werden können.

Sicherlich, es klingt heutzutage archaisch, Bilder als Programm einzugeben. In der Tat, PIC stammt noch aus einer Zeit, als Computer größtenteils über Textterminals bedient wurden. Aber insbesondere die automatische Anordnung von Text, die Möglichkeit der Erstellung von Makros und die genaue Plazierung der Primitive, ohne daß Koordinaten explizit angegeben werden müssen, macht PIC zu einem im großen und ganzen noch recht brauchbaren Werkzeug. Allerdings wird ein Textsystem benötigt, das die Ausgaben von PIC einbinden kann. Mit TEX, LATEX und dem EEPIC–Style steht ein solches System zur Verfügung. Die Formatanweisungen der Textformatierer können in den Texten innerhalb der Bildbeschreibungen verwendet werden. Da PIC eine PostScript Ausgabe besitzt, können im Prinzip alle Programme, die PostScript Dateien einbinden können, zur Weiterverarbeitung der Bilder (sprich deren Einsetzen in Texten) verwendet werden.

Die Orginalversion von PIC wurde um 1981 von Brian W. Kernighan entwickelt und wird auf Unix–Maschinen in einer Pipe als Präprozessor von NROFF verwendet. Soweit ich weiß, wurde sie inzwischen des öfteren überarbeitet. Etwas später wurde eine Version des Programms an der damaligen EWH Koblenz von Prof. Dr. H. Giesen implementiert, um möglichst einfach Grafiken für die Verwendung in seinem SCRIPT–Textsystem erstellen zu können. 1989 wurde PIC dann von mir um einige Funktionen erweitert und komplett neu in der Sprache C unter Verwendung der Tools Lex und Yacc im Rahmen einer Studienarbeit programmiert. 1992 schließlich wurde es durch das Erscheinen der neuen PasTEX Version von Georg Heßmann, deren DVI–Treiber die TPIC–Makros unterstützen, möglich, das Programm auf dem Amiga einzusetzen. Glücklicherweise unterstützt auch der DVIPS–Treiber von Tomas Rokicki die TPIC–Makros, wodurch die PostScript Ausgabe der Dokumente unterstützt wird.

Im Gegensatz zu der Version von Kernighan müssen für diese Version von PIC die Bilder in einzelnen Dateien bereitgestellt gestellt werden. Sie werden einzeln übersetzt (übrigens, PIC

enthält auch eine simple Preview–Funktion) und später in den jeweiligen Text eingebunden. In der Originalversion mußten die Bildbeschreibungen mit dem Text gemischt werden. PIC wurde dann dem eigentlichen Textformatierer vorgeschaltet und ersetzte die Bildbeschreibungen durch die jeweiligen Formatierer–Anweisungen. Die hier beschriebene Version von PIC wird *nicht* als Präprozessor in einer Pipe verwendet. Die Bildbeschreibungen können auf diese Weise getrennt vom eigentlichen Text gehalten werden. Die Syntax dieser PIC–Version weicht an einigen Stellen von der des Originals ab, so daß Programme die für die eine PIC–Version geschrieben wurden, nicht ohne weiteres von der anderen akzeptiert werden. Das Umsetzen dürfte aber keine größeren Schwierigkeiten bereiten. Anzumerken bleibt noch, daß PIC sehr viel Speicher benötigt (für Bilder mit vielen Primaries ohne weiteres mehr als ein Megabyte).

Das vorliegende Handbuch soll eine Einführung in die Verwendung von PIC geben. Der Text ist für die Verwendung von PIC in Verbindung mit SCRIPT auf PCS–Cadmus Rechnern geschrieben, gilt aber auch für die anderen Rechner auf denen PIC läuft (SUN–3, SUN–4, NeXT, Mips, Amiga). Die Beispiele sind für die Verwendung mit LaTeX aufbereitet, da auch dieses Handbuch mit diesem Satzsystem erstellt wurde. Spezielle Implementationsdetails sind am Ende des Buchs vermerkt.

Die vorliegende Version des Programms PIC richtet sich stark nach dem Original gleichen Namens von Brian W. Kernighan. Sie wurde nach dem, in dem amerikanischen Computerfachblatt 'Software – Practice and Experience' Vol. 12 (1982), Seite $1-21$ erschienenen Artikel von Brian W. Kernighan: 'PIC – A Language for Typsetting Graphics', in der Programmiersprache C, unter Verwendung der Tools: Lex, Yacc und Make, auf einem Unix–Rechner (PCS–Cadmus: pcs7), an der damals noch[1] EWH – Rheinland–Pfalz, Abt. Koblenz, Rheinau $3-4$, 5400 Koblenz, im Rahmen einer Studienarbeit, unter der Leitung von Prof. Dr. H. Giesen, von Andreas Pidde erstellt. An dieser Stelle möchte ich mich auch bei Volker Riediger für das Korrekturlesen der ersten Version des Benutzerhandbuches bedanken.

Als Hilfe bei der Programmerstellung diente neben dem Artikel ([KER]) von B. W. Kernighan ein bereits vorhandenes Pascal–Programm Namens PIC, erstellt von Prof. Dr. H. Giesen, das ähnlich wie diese Version arbeitet. Als weitere Hilfe wurde der Artikel ([BEN]) 'Little Languages' von Jon Bentley, erschienen in den 'Communications of ACM', August 1986, Volume 29, Number 8, Seite $711-721$, sowie Literatur über das Unix–System ([GUL]), die C–Sprache [K&R] und die Tools Lex und Yacc ([SIM]), verwendet.

Der vorliegende Text wurde im Orginal mit Hilfe der hier beschriebenen Version sowie mit SCRIPT erstellt. Der Textformatierer SCRIPT steht, incl. eines Benutzerhandbuchs, an der Universität Koblenz–Landau zur Verfügung. Die vorliegende Version des Handbuches wurde für das LaTeX Satzsystem umgeschrieben und ist mit diesem formatiert worden.

Diese Implementierung von PIC ist ein sogenanntes „Freely Distributable Program“, mit der Weitergabe des Programms darf also kein Gewinn erzielt werden, das Programm und die Dokumentation dürfen nicht in veränderter Form weitergegeben werden und die Rechte an Programm und Handbuch bleiben bei mir, dem Autor. Ich habe das Programm jetzt zwar schon einige Zeit in Gebrauch, ohne daß es Fehler gezeigt hat, kann aber weder Haftung noch irgendeine juristische Verantwortung für Fehlerfreiheit des Programms, falsche Angaben im Handbuch und den daraus resultierenden Folgen übernehmen.

[1] Heute endlich: **Universität** Koblenz–Landau

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Mit dem Programm PIC ist es möglich, Grafiken zu erstellen, sie sich auf einem Bitmap–Terminal anzusehen und sie mit Hilfe des Textformatierers SCRIPT[1] in Texte einzubinden. Grafiken werden mit PIC nicht interaktiv über ein Grafik–Terminal aufgebaut, sondern durch ein Programm in einer speziellen Programmiersprache spezifiziert. Dieses muß vorher mittels eines Editors erstellt oder direkt in die Standardeingabe geschrieben werden. Die Grundsprachelemente erzeugen sogenannte Grafik–Primaries, das sind:

Rechtecke, Kreise, Ellipsen,
Linien, Pfeile, Kreisbögen, Kurven und beliebige Texte

Es besteht natürlich auch die Möglichkeit in einem Koordinatensystem umherzuwandern ohne dabei etwas zeichnen zu lassen. Die Primaries werden durch entsprechende englische Schlüsselwörter eingeleitet und können beliebig in der Maßeinheit 'Inch' dimensioniert werden. Ein oder mehrere Primaries können zu einem Grafik–Block zusammengefaßt werden, der danach wie ein eigenständiges Grafikelement behandelt wird.

In der PIC–Sprache wurde besonderer Wert darauf gelegt, daß der/die Benutzer/in mit möglichst wenigen absoluten Koordinaten arbeiten muß. Die Plazierung der Primaries erfolgt implizit durch den aktuellen Standpunkt innerhalb des Koordinatensystems. Dieser Standort wird je nach angegebener Laufrichtung und gesetztem Primary automatisch umgesetzt. Sie können sich das Setzen der Primaries so vorstellen, als ob Sie auf einem Blatt Papier mit einem Zeiger umherwandern und an bestimmten Stellen Primaries fallen lassen. Die gesamte Grafikhülle (ein Rechteck, das alle Ihre Grafik–Primaries umschließt) wird am Ende der Eingabe horizontal zentriert.

Sprachelemente von PIC sind weiterhin: For–Schleifen, bedingte Anweisungen, Variablen, Labels zur Kennzeichnung von bestimmten Primaries, Rechenfunktionen und ein Makromechanismus. Variablen gelten im aktuellen Grafik–Block lokal, können aber exportiert werden. Auf Labels kann erst zugegriffen werden, nachdem sie definiert wurden.

Der Vorteil der Benutzung einer Programmiersprache zur Erstellung von Grafiken ist (oder besser: War damals), daß Sie nicht auf ein Bitmap–Terminal angewiesen sind. Sie können ihre Grafiken auf einem ganz normalen Textterminal erstellen und durch einen Textformatierer, in diesem Fall SCRIPT, zum Ausdruck bringen. Auch ist der Plattenspeicherverbrauch von PIC–Programmen i.A. nicht so hoch wie der von den entsprechenden Bit–Planes. Weiterhin von Vorteil ist, daß Änderungen der Primary–Größen in PIC, falls beim Programmieren darauf geachtet

[1] Die Möglichkeit der PostScript oder LaTeX Ausgabe besteht ebenfalls und kann durch daß Setzen von Optionen benutzt werden.

wurde, schnell durch Änderung signifikanter Variablen erreicht werden können. In interaktiven Systemen ist dies nicht immer so einfach. Sie müßten wahrscheinlich in einem solchen Fall Ihre Grafik noch einmal zeichnen. Nachteile von PIC sind vor allem die mitunter schwierige Programmierung und die Beschränkung auf eine kleine Menge von Primaries. Durch Letzteres bleibt die Sprache jedoch relativ übersichtlich. Da es so etwas wie Pixelpunkte, die an beliebigen Stellen gesetzt werden können, in PIC nicht gibt, können bei weitem nicht alle denkbaren Grafiken mit PIC mit vernünftigem Aufwand erstellt werden. PIC eignet sich vor allem für Schemazeichnungen und Graphen.

2. Beispielsitzung mit PIC

Die folgende Beschreibung einer Beispielsitzung gilt in dieser Form wohl nur für die PCS-Cadmus–Rechner (Unix), ist aber im Prinzip auch auf andere Rechner und Betriebsysteme übertragbar. Um mit PIC arbeiten zu können, müssen Sie zuerst ein Programm in der PIC–Sprache erstellen. Machen Sie sich deshalb bitte ein wenig mit einem Editor (z.B. ey) vertraut. Außerdem kann es nicht schaden wenn Sie sich ein paar wichtige Unix–Befehle, wie den ls–Befehl zum Directory–Listen, den rm–Befehl zum Löschen von Dateien und den lp–Befehl zum Drucken von Dateien, ansehen. Eine Einleitung in das Unix–System ist in [GUL] zu finden.

Falls Sie die Grafiken in SCRIPT benutzen möchten, sollten Sie natürlich auch wissen, wie dieses Programm zu bedienen ist und wie Sie die Result–Dateien ausdrucken können.

Ein Programm, das Sie in der PIC–Sprache erstellt haben, können Sie durch das Programm PIC interpretieren lassen. Durch das Setzen von bestimmten Optionen wird das Bild entweder als SCRIPT–Datei oder auf einem Bitmap–Terminal grafisch dargestellt. Zu letzterem müssen Sie sich auch an einem solchen eingeloggt haben.

Aber nun das Bild, das entstehen soll, wenn Sie die darunter folgende Beispielsitzung durchführen:

Abbildung 2.1: hallo.pic

```
# hallo.pic, eine einfache Box mit Text
# Bild Nr. 1

box "Hallo" "user !"
```

Loggen Sie sich dazu bitte zuerst an einem PCS–Rechner, auf dem PIC läuft, möglichst an einem Bitmap–Terminal, korrekt ein. Nachdem das Unix–Betriebssystem durch die Ausgabe eines $–Zeichens meldet, daß Sie eine Eingabe tätigen können, rufen Sie bitte mit folgender Zeile den Editor auf:

```
$ ey hallo.pic
```

Das $–Zeichen ist der Eingabeprompt und soll nicht mit eingegeben werden. ’hallo.pic’ ist der Name der Datei, die Sie erstellen werden. Nachdem Sie die RETURN–Taste gedrückt haben befinden Sie sich bereits (oder nicht so bereits, je nach Systembelastung) im Editor. Schreiben Sie dort bitte folgenden Text:

```
box "Hallo" "user !"
```

Speichern Sie Ihr Erstlingswerk danach durch das nacheinander zu erfolgende Drücken der ESC–Taste und der w–Taste, gefolgt vom Drücken der RETURN–Taste, als Bestätigung auf eine Anfrage des Editors (Write:), ob Sie auch wirklich abspeichern wollen, ab[1]. Nachdem Sie dies ausgeführt haben, haben Sie automatisch auch den Editor verlassen und befinden sich wieder in der Unix–Shell. In Ihren Dateikatalog sollte nun die Datei: ’hallo.pic’ zu finden sein. Diese können Sie nun auf dem Bitmap–Terminal als interpretierte Grafik ausgeben lassen, indem Sie folgende Zeile eingeben:

```
$ pic -b <hallo.pic
```

Nachdem Sie wiederum am Ende der Zeile die RETURN–Taste gedrückt haben, sollte die oben gezeigte Grafik auf dem Bitmap–Terminal erscheinen. Wie Sie sehen, wird der Text standardmäßig in dem angegebenen Primary, hier das Rechteck (**box**), zentriert.

Falls Sie nicht direkt an einem Bitmap–Terminal arbeiten, sondern sich nur an einem solchen eingeloggt haben, müßen Sie direkt hinter dem ’−b’ den Namen des Bitmap–Terminals schreiben (an PCS–Anlagen meistens ’−bbip0’). Sie können dann Ihre Grafik–Programme an einem ASCII–Terminal edieren und sie sich an einem Bitmapterminal ansehen.

Wie Sie sicherlich an dem ’<’–Zeichen in ihrer letzten Shell–Eingabe bereits erkannt haben, liest PIC aus der Standardeingabe. Sie können, vor allem sehr kleine Programme, also auch direkt nach dem Aufruf von PIC eingeben und durch das Drücken der z–Taste bei niedergehaltener Ctrl–Taste beenden[2]. Gehen Sie dabei wie folgt vor: Schreiben Sie

```
$ pic -b
```

drücken Sie die RETURN–Taste, geben Sie den Text:

```
box "Hallo" "user !"
```

ein, drücken wieder die RETURN–Taste und danach die Tastenkombination Ctrl–z. Das Ergebnis sollte das gleiche sein, wie in dem Beispiel weiter oben.

Falls Sie eine SCRIPT–Datei erzeugen wollen, können Sie dies durch die Ausgabeumleitung durchführen. Geben Sie dazu bitte folgendes ein (auf PCS-Rechner ist die SCRIPT–Ausgabe voreingestellt):

$ pic <hallo.pic >hallo.scr

oder (`-s` ist die Option für die SCRIPT–Ausgabe):

[1] Das gilt natürlich in dieser Form nur für den ’ey’–Editor.

[2] Auf anderen Rechnern muß u.U. eine andere Tastenkombination verwendet werden.

```
$ pic -s <hallo.pic >hallo.scr
```

oder nur:

```
$ pic hallo
```

Nachdem dieser Aufruf bearbeitet wurde, steht die SCRIPT–Datei 'hallo.scr' in Ihrem Directory. Sie können diese mit SCRIPT weiterbearbeiten. Die Dateinamen können natürlich beliebig gewählt werden. Sie sollten der Übersichtlichkeit halber aber darauf achten, daß PIC–Dateien '.pic' und SCRIPT–Dateien '.scr' als Anhängsel (Suffix) bekommen. Sie können Ihre Dateien dann auch leichter durch Shell–Prozeduren behandeln, außerdem unterstützt PIC diese Suffixe und setzt sie, falls sie fehlen, beim Aufruf automatisch ein. Ausgabedateien aus SCRIPT werden von diesem Programm mit '.res' (result) gekennzeichnet.

PIC verwendet folgende Suffixe:

Suffix	Dateityp
`.pic`	PIC Eingabedatei
`.scr`	SCRIPT-Datei
`.tex`	TEX– oder LATEX–Datei
`.ps`	PostScript–Datei

Übrigens wenn Sie

```
$ pic -u
```

oder (nicht in der C–Shell):

```
$ pic ?
```

aufrufen, bekommen Sie die aktuellen Aufrufkonventionen auf dem Bildschirm ausgegeben.

Das Unix–System können Sie durch die Eingabe von

```
$ exit
```

verlassen.

3. Die PIC–Sprache

In diesem Kapitel finden Sie einen allgemein gehaltenen Überblick über die PIC–Sprache, gefolgt von einer genaueren Beschreibung der einzelnen Befehlstypen mit einigen Beispielen. In einem folgenden Kapitel ist noch die gesamte Syntax in BNF dargestellt. Dort befindet sich auch eine Beschreibung dieser Metasprache.

3.1 Allgemeiner Überblick

Ein PIC–Programm besteht aus einer Aneinanderreihung von Befehlszeilen. Leerzeichen und Tabulatorzeichen haben eine trennende Bedeutung für die einzelnen Sprachelemente. Das Programmende entspricht dem Dateiendezeichen (hat den Wert 0 im ASCII–Code). Wenn Sie ein Programm direkt in die Standardeingabe tippen wollen, können Sie dieses Zeichen an den Terminals der PCS–Rechner durch ein CTRL–z (an vielen anderen Rechnern durch CTRL–d, auf dem Amiga durch CTRL–\) simulieren. An manchen Terminals ist es möglich, das 0–Zeichen (Dateiende) durch Ctrl–Klammeraffe anzugeben. Auch eine leere Datei wird als Programm erkannt.

Befehlszeilen werden entweder durch ein Zeilenende, oder durch ein Semikolon abgeschlossen. Sollte ein Befehl länger als eine Zeile werden, so muß vor die Stelle, an der getrennt wurde, ein sogenannter Backslash ('\') gesetzt werden. Er sorgt dafür, daß das Zeilenende von PIC überlesen wird. Leerzeilen (leere Befehle) zwischen den einzelnen Befehlen können verwendet werden.

PIC unterscheidet zwischen großen und kleinen Buchstaben. Das Sonderzeichen '_', der sogenannte Unterstrich oder Underscore, wird im folgenden zu der Menge der Buchstaben (letter) hinzugezählt. Ziffern (digits) seien die Zeichen von $0 - 9$.

Identifier sind Variablen– und Labelnamen, beginnen mit einem Buchstaben, gefolgt von keinem, einem oder mehreren Buchstaben oder Ziffern. Solche Identifier dürfen keine Schlüsselwörter (wie z.B. 'box') sein.

Hier ein Syntaxdiagramm zu der Schreibweise von Identifiern:

```
# identifier.pic, Syntaxdiagramm der Schreibweise von Identifiern
# Bild Nr. 2

"\\large \\bf Identifier :" ljust wid 1.5

# Syntaxdiagramm
[ box "letter"              # Die Box aussen links
```

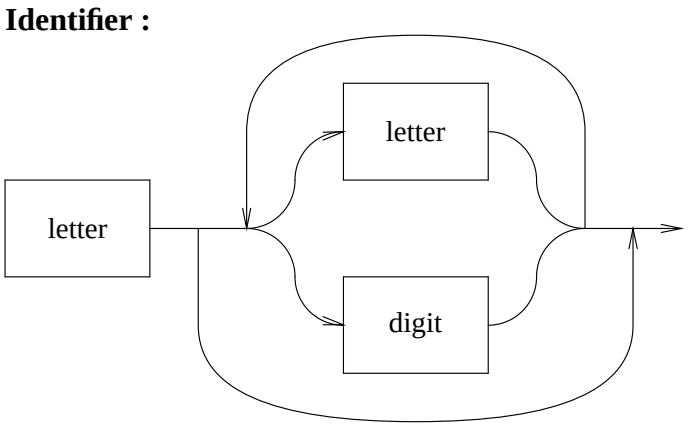


Abbildung 3.1: identifier.pic

```
  line
  arc
  arc cw ->
  box "letter"               # Die Box oben
  arc cw
  arc
  arc cw from last line.end  # Neuer Anfang an der linken Linie
  arc ->
  box "digit"                # Box unten
  arc
  arc cw
  arrow
  # Grosse obere Kurve
  spline up 1 from last arrow.start then left 1.75 then down 1 ->
  # Grosse untere Kurve
  spline down 1 from 1st line.center then right 2.25 then up 1 ->
] with nw at last text.sw  # Verschiebung des gesamten Blocks
```

PIC macht keinen Unterschied zwischen den Zahlentypen der Integer– und Real–Zahlen, wie das die höheren Programmiersprachen tun. Intern benutzt PIC nur Real–Zahlen. Sie haben die Möglichkeit, jede Zahl optional mit Nachkommastellen oder mit einem Exponenten anzugeben. Als Dezimalpunkt dient der normale Punkt '.'. Führende Nullen vor dem Punkt können weggelassen werden. Sie können also anstatt 0.5 auch .5 schreiben. Die Schreibweise der Zahlen ist im Syntax–Kapitel in BNF dargestellt.

Zwei Zahlen, durch ein Komma getrennt und mit einer runden Klammer umgeben, bilden eine Koordinate (Bsp: (7,.35)). Es gilt (x–Wert, y–Wert).

Strings sind in PIC Zeichenketten, die in zwei Häkchen (“) eingeschlossen werden. Diese Zeichenketten werden von PIC unverändert übernommen. Eine Ausnahme davon bildet lediglich der Backslash. Der Backslash bewirkt, daß das auf ihn folgende Zeichen in jedem Fall übernommen wird. Steht hinter einem Backslash ein Häkchen, wird dieses in den String *über-*

nommen und beendet *nicht* den String. Der Backslash selbst wird nicht in die Ausgabe geschrieben. Möchten Sie dennoch einen solchen innerhalb eines Textes ausgeben, müssen Sie zwei Backslashes hintereinander schreiben. Ein Backslash gefolgt von einem Backslash wird also zu einem einzelnen Backslash. Befindet sich hinter dem Backslash ein Zeilenende, wird der String in der nächsten Zeile weitergelesen. Bitte beachten Sie dabei, daß das Einrücken dieser Zeile den String verändern kann (die Blanks und Tabulatoren werden in den String übernommen). Beispiele für Ersetzungen:

\\ wird zu \
\“ wird zu “
\x wird zu x

Die Primaries bilden den wichtigsten Teil innerhalb der PIC–Sprache. Sie werden durch folgende Schlüsselwörter eingeleitet:

box
circle
ellipse
line
arrow
arc
spline
move

’spline’ erzeugt eine Bézier–Kurve und ’move’ führt eine Bewegung durch. Wenn eines dieser Schlüsselwörter am Anfang einer Befehlszeile steht, wird später ein entsprechendes Primary ausgegeben. Zu den oben angegebenen Primaries bleibt noch hinzuzufügen, daß auch Text, das heißt eine Befehlszeile, die durch einen String eingeleitet wird, als Primary zählt.

Eine Primary–Befehlszeile kann außer dem Schlüsselwort noch Dimensionierungen in der Maßeinheit ’Inch’, wie Rechteckhöhe oder Kreisradius, sowie Angaben zur Linienart, Plazierungshinweise und Texte (evtl. mit Textattributen zu deren Ausrichtung), als Attribute enthalten. Außerdem ist es möglich, Angaben zu machen, ob eine Linie so verkürzt werden soll, daß sie nur den Rand des Primaries (’box’, ’circle’ oder ’ellipse’), in dem ihr Anfangs–, bzw. Endpunkt liegt, berührt. Falls eines der obigen Attribute einmal auf ein bestimmtes Primary nicht angewendet werden kann, wird es ignoriert. Falls keine Attribute angegeben werden, werden die standardmäßig vorgegebenen Werte, die sogenannten Defaultgrößen, übernommen. Das spart einiges an Tipparbeit.

Ein oder mehrere Primaries können durch eckige Klammern (’[’ und ’]’) zu einem Grafik–Block geklammert werden. Er darf direkt nach der schließenden Klammer auch Attribute erhalten. Diese Grafik–Blöcke dürfen ineinander verschachtelt werden. Jeder Block hat eigene, lokale Koordinaten, Variablen und Labels und er kann als Ganzes verschoben werden.

Außer den Primary–Befehlen gibt es noch solche, die die standardmäßige Bewegungsrichtung und deren Weite ändern. Es gibt vier Bewegungsrichtungen, die in der Form von Himmelsrichtungen angegeben werden können. Dabei gilt, daß Norden immer auf die Oberkante des Bildes weist. Die Voreinstellung ist die Bewegung von links nach rechts, also nach Osten, in Schritten

von 0.5 Inch Länge. Ein Laufrichtungs–Befehl besteht aus der Angabe einer Bewegungsrichtung, z.B. 'north', 'east', oder auch 'up', oder 'right', gefolgt von einer Weite in Form einer Zahl wie 0.5 oder .25. Die Maßeinheit ist immer Inch. Die Weite dient als Voreinstellung für die Linienlänge oder für den 'move'–Befehl, sie kann auch negativ sein. Die Bewegungsrichtung kann durch bestimmte Primaries (z.B. 'line' und 'arc') beeinflußt werden. Wie dies geschieht, wird in den zu diesen Primaries gehörenden Kapiteln erklärt.

Da in PIC mit Variablen gerechnet werden kann, sind natürlich auch Zuweisungen in der Befehlssyntax enthalten. Außerdem steht eine Vielzahl von mathematischen Funktion zur Verfügung. Variablen gelten immer lokal in einem Grafik–Block. Sie werden nicht deklariert. Wird eine Variable benutzt ohne daß ihr ein Wert zugewiesen wurde, erhält sie den Wert aus dem übergeordneten Grafik–Block. Existiert kein solcher Block, bekommt diese Variable als Anfangswert 0.0 zugewiesen. Wie die mathematischen Ausdrücke (Expressions) aussehen, wird in einem eigenen Kapitel erklärt.

Weitere nützliche Befehle in PIC sind der Schleifenbefehl und die bedingte Anweisung. Die Schleifenanweisung ist eine 'for'–Schleife, deren Stepwert angegeben werden kann. Nach der bedingten (if) Anweisung darf ein 'else'–Teil folgen. Die Anweisungs–Blöcke, auf die sich die strukturierten Anweisungen beziehen, *müssen* durch geschweifte Klammern ('{' und '}') vollständig geklammert werden. Eine genauere Beschreibung dieser beiden Anweisungen folgt.

Desweiteren ist PIC noch mit einem leistungsfähigen Makromechanismus ausgestattet, der Parameter und Rekursionen zuläßt. Makros gelten im Gegensatz zu Variablen und Labels nicht nur lokal in dem Block, in dem sie definiert wurden, sondern global im ganzen PIC–Text. Sie können auch vor ihrer Definition aufgerufen werden.

Kommentarzeilen werden durch ein Doppelkreuz ('#') eingeleitet; sie dürfen bis zum Zeilenende beliebige Zeichen enthalten. Kommentare können auch hinter dem Backslash, der das Zeilenende aufhebt, angebracht werden. Auch dort können beliebige Zeichen bis zum Zeilenende folgen. Der Unterschied zwischen den beiden Möglichkeiten besteht darin, daß der Backslash das Zeilenende aufhebt, das Doppelkreuz jedoch nicht.

Vor jedem Befehl, der ein Primary enthält, kann ein Label definiert werden. Das geschieht, indem vor diesem Befehl ein Identifier, gefolgt von einem Doppelpunkt, geschreiben wird. Bsp:

```
Label: circle
```

bewirkt eine Kennzeichnung des Kreises durch den Namen 'Label'. Labels gelten ebenfalls lokal in ihrem Grafik–Block. Es gibt jedoch eine Möglichkeit, sie aus den darüberliegenen Blöcken anzusprechen. Diese Benennung von Primaries ermöglicht eine einfache Plazierung von anderen Primaries in Bezug auf die so gekennzeichneten. Es können auch Grafik–Blöcke mit einem Label versehen werden. Labels gelten erst, nachdem sie definiert wurden. Es besteht die Möglichkeit, sie durch ein Label gleichen Namens überzudefinieren.

Eine weitere Anwendung von Labels ist, sie vor eine Koordinatenangabe zu setzen. Bestimmte Punkte können so gekennzeichnet werden. Geschieht eine solche Angabe eines Punktes innerhalb eines Grafik–Blockes, kann dieser Block mit der so markierten Stelle auf einen bestimmten, anzugebenden Punkt plaziert werden (`with Punkt at Punkt`).

3.2 Die Expressions

Um mit PIC arbeiten zu können, ist es wichtig zu wissen, wie mathematische Ausdrücke dargestellt werden. Außerdem sollten Sie wissen, welche Möglichkeiten Sie haben, Bildkoordinaten aufzuführen. Deshalb werde ich versuchen, Ihnen diese beiden Sprachbestandteile in den nächsten beiden Kapiteln zu erläutern.

Expressions sind auf unterster Stufe einfache Zahlen, Variablen, oder die vordefinierten Konstanten 'true', 'false' oder 'pi'. Expressions können untereinander durch binäre Operatoren verknüpft werden und bilden so wieder Expressions. Desweiteren können eine oder mehrere, durch ein Komma getrennte, Expressions die Argumente eines Funktionsaufrufes bilden. Ich habe die bekanntesten mathematischen Funktionen aus der C–Library in PIC übernommen (siehe [GUL], Seite 461). Expressions können außerdem geklammert und mit den unären Operatoren '−' und 'not' versehen werden. Auch lassen sich logische Operatoren, wie 'and' und 'or' in Expressions anwenden. Zwei Expressions lassen sich weiterhin durch Vergleichsoperatoren verknüpfen. Die Operator–Präzedenzen sind äquivalent zu denen in der Programmiersprache C. Es gilt insbesondere die Punkt–vor–Strich–Rechnung.

Der Wert von Expressions kann durch Gleichheitszeichen einer oder mehreren Variablen zugewiesen werden.

Hier nun einige Beispiele für Expressions:

```
a * 10 / sin(phi)
```

'a' und 'phi' sind hierbei Variablenbezeichner, 'sin()' ein Funktionsaufruf, der '*' ist das Zeichen für eine Multiplikation und der Schrägstrich das für eine Division. In dieser Expression wird der Wert der Variable 'a' mit 10 multipliziert und anschließend durch den Sinus von dem Wert der Variable 'phi' dividiert.

```
-(a + 3) * 4.3
```

Hier wird der Wert der Variablen 'a' mit 3 addiert, das Ergebnis negiert und schließlich mit 4.3 multipliziert.

```
a and b
```

ist eine Ausdruck mit dem logischen 'und'. 'true' und 'false' sind die logischen Konstanten. Es gelten alle Werte, die ungleich 0 sind, als logisch wahr ('true') und der Wert 0 als logisch falsch ('false'). Es können durchaus auch Zahlen mit Nachkommastellen als logische Konstanten verwendet werden, aber nur 0.0 gilt als logisch falsch.

Hier noch drei mögliche Zuweisungen:

```
a = 3
a = b = c = d / (7 * c)
v = b >= c
```

Im ersten Beispiel bekommt die Variable ’a’ den Wert 3 zugewiesen. Im zweiten Beispiel werden alle drei Variablen, ’a’, ’b’ und ’c’, mit dem Ergebnis der Operation `d/(7*c)` belegt. Im letzten Beispiel bekommt ’v’ den Wert ’true’ zugewiesen, wenn ’b’ größer oder gleich ’c’ ist, ansonsten wird ’v’ mit dem Wert ’false’ belegt.

Die Syntax der Expressions in BNF finden Sie in dem entsprechenden Kapitel.

Hier noch eine Aufzählung der Operatoren, mit ihren Präzedenzen, in ähnlicher Aufstellung, wie in [K&R], Seite 54. Operatoren mit gleicher Präzedenz stehen in der gleichen Zeile. Die Operatoren mit der höchsten Präzedenz stehen in der ersten Zeile, die mit der niedrigsten in der letzten. In der zweiten Spalte steht, von welcher Seite aus Operanden mit ihren Operatoren zusammengefasst werden.

Operator	**Zusammenfassung**
()	von links her
not −	von rechts her
∗/ div mod	von links her
+−	von links her
<<=>>=	von links her
==<>	von links her
and	von links her
or	von links her
=	von rechts her

Abbildung 3.2: prec.pic

```
# prec.pic, Operatorpraezedenzen von PIC
# Bild Nr. 3

textht = textht + 1/72
textwid = 1/2 * textht

define R     '"von rechts her"'
define L     '"von links her"'

# Ueberschriften der zwei Tabellenspalten
box ht 1.5*textht wid 15*textwid "\\bf Operator" len -3
box ht 1.5*textht wid 19*textwid "\\bf Zusammenfassung" len -3

# Grosse Boxen, in denen der gesamte untere Text Platz findet

# Linke Spalte
box ht 11*textht wid 15*textwid with .nw at 1st box.sw \
"()" \
"not $-$" \
```

```
"$* /$ div mod" \
"$+ -$" \
"$< <= > >=$" \
"$== <>$" \
"and" \
"or" \
"$=$"

# Rechte Spalte
box ht 11*textht wid 19*textwid L R L L L L L L R
```

Abschließend noch eine Tabelle der mathematischen Funktionen (siehe auch [GUL] Seite 461), die in PIC übernommen wurden, mit ihrer Bedeutung. 'x', 'y' stehen hierbei für Expressions, coord für eine Koordinate. Wobei 'coord.x' der x–Teil und 'coord.y' der y–Teil einer Koordinate ist.

Funktion	Bedeutung
xcoord(coord)	x–Teil einer Koordinate liefern.
ycoord(coord)	y–Teil einer Koordinate liefern.
cabs(coord)	Resultat: sqrt(pow(coord.x,2)+pow(coord.y,2))
fmod(coord)	coord.x = i*coord.y+f, f ist das Resultat.
hypot(coord)	Resultat: sqrt(pow(coord.x,2)*pow(coord.y,2))
atan2(x, y)	atan(x, y)
pow(x, y)	Potenzierung von x mit y
acos(x)	Arcus Kosinus von x mit 0 <= x <= pi
asin(x)	Arcus Sinus von x mit −pi/2 <= x <= pi/2
atan(x)	Arcus Tangens x mit −pi/2 <= x <= pi/2
ceil(x)	Kleinste ganze Zahl >= x
cmtoi(x)	Umrechnung cm in inch
cos(x)	Kosinus von x
cosh(x)	Kosinus Hyperbolikus von x
exp(x)	Exponentialfunktion
fabs(x)	Betragsfunktion
floor(x)	Größte ganze Zahl <= x
int(x)	x ohne Nachkommastellen
log(x)	Natürlicher Logarithmus von x
log10(x)	Logarithmus zur Basis 10 von x
sin(x)	Sinus von x
sinh(x)	Sinus Hyperbolikus von x
sqrt(x)	Quadratische Wurzel von x
tan(x)	Tangens von x
tanh(x)	Tangens Hyperbolikus von x
rand()	Generiert Zufallszahlen aus dem Intervall $[0.0, 1.0[$. Kann durch srand() initialisiert werden.

Es sei noch erwähnt, daß Zahlen in Strings umgewandelt werden können. Für diese Aufgabe steht das 'str'–Kommando zur Verfügung. Es wird wie folgt verwendet:

str(Zahl, Vorkommastellen, Nachkommastellen)

Die Vorkommastellen sind als Richtwert für die Ausrichtung gedacht und brauchen nicht mit der tatsächlichen Anzahl übereinstimmen. Wird für Nachkommastellen der Wert 0 eingegeben,

werden keine Nachkommastellen gedruckt, ansonsten wird auf die Anzahl der Nachkommastellen gerundet.

3.3 Die Koordinaten

Eine Koordinate beschreibt in PIC einen bestimmten Punkt innerhalb eines kartesischen Koordinatensystems. Dabei gilt, daß jeder Grafik–Block ein eigenes Koordinatensystem besitzt. Der Ursprung ist der Aufsetzpunkt, mit dem der Grafik–Block beginnt. Die Werte in y–Richtung werden zum oberen Bildrand hin größer und wachsen in x–Richtung zum rechten Bildrand hin. In PIC können Koordinaten untereinander addiert und subtrahiert werden.

Punkte können in PIC nicht nur als Folge von zwei Expressions (’expr’) in der Form:

```
(expr, expr)
```

geschrieben werden, wobei die linke Expression den x–Wert des Punktes und die rechte den y–Wert angibt. Sie können auch als Ecken von Primaries angegeben werden. Die Ecken des Primaries sind nach den Himmelsrichtungen ’n’, ’s’, ’w’, ’e’, ’nw’, ’ne’, ’sw’, ’se’ benannt. Außerdem kann noch die Primary–Mitte (’center’) angegeben werden. Bei linienartigen Gebilden existieren der Start, das Ende und die Mitte. Auch die Mitte eines Kreisbogens (’middle’) kann angegeben werden. Anstelle der Himmelsrichtungen können auch Schlüsselworte wie ’top’ und ’right’ verwendet werden. Es folgt eine Auflistung dieser Ecken–Bezeichner. Bezeichner gleicher Bedeutung stehen in der gleichen Zeile.

’north’, ’n’, ’top’, ’t’, ’up’, ’u’
’south’, ’s’, ’bottom’, ’bot’, ’b’
’west’, ’w’, ’left’, ’l’
’east’, ’e’, ’right’, ’r’
’nw’
’ne’
’sw’
’se’
’center’, ’cen’, ’c’
’middle’, ’mid’, ’m’
’start’, ’st’
’end’, ’en’

Die Primaries, auf die sich diese Eckbezeichner beziehen sollen, können durch die Schlüsselwörter:

’first’, ’1st’
’second’, ’2nd’
’third’, ’3rd’
’4th’, usw.
’last’

oder durch Kombinationen mit ’last’, wie ’2nd last’, gefolgt von einem Primarybezeichner, sowie durch Labelnamen selektiert werden. ’first’ bezieht sich auf das erste Primary eines Grafik–Blocks, ’second’ auf das zweite, usw. Mit

```
third last box
```

wird das dritt–letzte Rechteck ausgewählt. Die Ecken eines Primaries können nun durch einen Punkt von einem Primary–Bezeichner abgetrennt als Punktangabe angegeben werden, z.B.:

```
last line.start
last text.nw
last circle
```

Wird wie im letzten Beispiel keine Eckangabe gemacht, wird ’center’ als die gesuchte Stelle angenommen. Hier noch ein Beispiel mit einem Label:

```
AnEllipse: ellipse
AnEllipse.se
```

Durch den Labelnamen ’AnEllipse’ mit dem Anhängsel ’.se’ wurde die rechte untere Ecke (liegt auf dem Ellipsenrand) der Ellipse, die durch das Label gekennzeichnet wurde, als Punkt ausgewählt.

Eine andere Möglichkeit eine Ecke auszuwählen bietet die Möglichkeit der Benutzung des Tokens ’of’ (im folgenden benutze ich den Ausdruck Token anstatt des Wortes Schlüsselwort). Sie können anstelle der durch einen Punkt abgetrennten Eckangabe auch folgendes schreiben:

```
nw of last box
.nw of last box
```

welches das Gleiche bedeutet wie:

```
last box.nw
```

Welche Form Sie verwenden, bleibt Ihrer persönlichen Vorliebe überlassen.

Eine andere Möglichkeit einer Koordinatenangabe ist die der Benennung eines Zwischenpunktes von zwei Punkten durch einen Faktor. Beispiel:

```
1/2 between the way of last circle and end of 2nd spline
```

Die beiden Token ’the’ und ’way’ haben keine Bedeutung. Sie können weggelassen werden, oder an beliebigen anderen Stellen im Programm geschrieben werden, damit der Text flüssiger aussieht. ’between’ und ’of’ können auch allein verwendet werden. Der obige Ausdruck wählt den Punkt, der genau in der Mitte zwischen den beiden angegebenen Punkten (dem Mittelpunkt des letzten Kreises und dem Endpunkt des letzten Splines) liegt.

Hier ein Bild und das entsprechende PIC–Programm dazu:

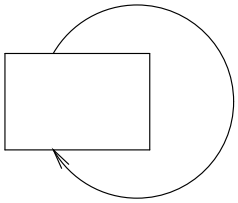


Abbildung 3.3: between.pic

```
# between.pic, zeigt eine Verwendung von between coord and coord
# Bild Nr. 4

box
# Grossen Kreisbogen in Uhrzeigersinn mit Pfeilspitze um die Box ziehen
arc -> from 1/3 between last box.nw and last box.ne \
       to   1/3 between last box.sw and last box.se \
       cw
```

'cw' bedeutet hierbei, daß der Kreisbogen ('arc') im Uhrzeigersinn gedreht wird. '$->$' bewirkt, daß an dessen Ende noch eine Pfeilspitze gesetzt wird.

Abschließend noch eine letzte Möglichkeit der Darstellung von Koordinaten, wie sie vor allem in Linienpfaden Verwendung findet: Die relative Darstellung von Koordinaten. Bei ihr geben Sie eine Richtung, gefolgt von einer Schrittweite in Inch, an. Danach können weitere solcher Richtungsangaben folgen. Wird keine Schrittweite angegeben, wird die Defaultweite (die von der Laufrichtungsangabe) angenommen. Bezugspunkt zu der relativen Koordinate ist immer der aktuelle Standpunkt.

Beispiele:

```
right .25 up .5
```

bezeichnet die Koordinate, die .25inch rechts und .5inch über dem aktuellen Standpunkt liegt.

```
west west
```

bezeichnet die Stelle, die um zwei Default–Schrittlängen links vom aktuellen Standpunkt liegt.

```
north south
```

ist der aktuelle Standpunkt selbst.

Die Syntax der Koordinaten in BNF finden Sie wieder in dem entsprechenden Kapitel über die Syntax.

3.4 Die Defaultvariablen

Die Defaultvariablen dienen dazu, die Standardgrößen von Primaries festzulegen. Mit Hilfe dieser Variablen ist es möglich, z.B. den Radius aller Kreise zu verändern. Ein Default–Wert wird dann angenommen, wenn in den Attributen der darauffolgenden Befehle keine Angaben bezüglich dieses Wertes mehr gemacht werden. Mit Defaultvariablen können Sie wie mit gewöhnlichen Variablen rechnen. Sie sind mit den Standardgrößen der Primaries vorbelegt. Hier eine Auflistung der Variablen mit den dazugehörigen Standardwerten:

Größe	Variable	=Standardwert
Rechteckbreite	boxwid	=.75
Rechteckhöhe	boxht	=.75
Ellipsenbreite	ellipsewid	=.75
Ellipsenhöhe	ellipseht	=.75
Kreisradius	circlerad	=.25
Arcradius	arcrad	=.25
Buchstabenbreite	textwid	=.1
Buchstabenhöhe	textht	=1/6
Schrittweite (Linienlänge)	linewid	=.5

Die Buchstabenmaße sind normalerweise auf den PICA–Font voreingestellt. Wird ein anderer Font verwendet, sollten Sie die beiden Defaultvariablen entsprechend ändern, was z.T. automatisch geschieht. Z.B. ist der Font auf den Bitmap–Terminals etwas größer als Pica. Falls Sie Ihre Grafik auf einem Bitmap–Terminal ausgeben lassen, werden die beiden Defaultvariablen automatisch auf folgende Werte gesetzt:

```
textwid = .119
textwid = .168
```

Der Bitmap–Font wird als Default–Font gewählt, wenn Sie Ihre Grafik auf einem Bitmap–Terminal ausgeben, ansonsten ist Pica der Defaultfont. Angaben sind wieder in Inch zu machen.

Die Defaultvariable linewid hat immer den Wert der aktuellen Schrittweite, ändern Sie also die Richtung, ändert evtl. auch die Variable ’linewid’ ihren Wert. ’right 2*linewid’ bewirkt, daß erstens, durch die Angabe von ’right’, die Variable linewid durch PIC automatisch auf die Schrittweite der Richtung ’rechts’ gesetzt wird. Danach wird dieser Wert mit 2 multipliziert und dient als neue Schrittweite. ’right 2*linewid’ bezeichnet also den Punkt, der zwei Schrittweiten rechts vom aktuellen Standpunkt entfernt liegt.

3.5 Primary–Befehle

Es folgt nun eine ausführlichere Beschreibung der einzelnen Primary–Befehle. Den PIC–Quelltext zu den Bildern finden Sie jeweils in der Nähe des Bildes.

Die Zusammenfassung der Schreibweise ist in BNF geschrieben. ’expr’ steht für eine Expression, deren Dimension falls nötig in Inch angenommen wird. ’text textattr’ steht für ein Textattribut. ’coord’ kann durch einen absoluten Koordinatenwert ersezt werden, ’relcoord’ durch eine

relative Koordinate (z.B. 'right .45'). 'corner' steht für eine Eckangabe. Wie diese Nonterminals ersetzt werden können, entnehmen Sie bitte aus dem Kapitel der Syntaxbeschreibung. Ich habe auch hier die Terminal–Symbole (Tokens) in Doppel–Häkchen (") gesetzt. Es können für sie auch die jeweiligen Synonyme, die in der Token–Zusammenfassung aufgeführt sind, geschrieben werden.

Ein Primary–Befehl beginnt immer mit einem einleitenden Schlüsselwort, fakultativ gefolgt von Attributen. Bitte beachten Sie, daß die Reihenfolge der Attribute prinzipiell zwar keine Rolle spielt, daß Sie aber die Textattribute direkt hinter einen String schreiben müssen. Die Textattribute gelten für alle Strings, die in diesem Primary aufgeführt werden.

Schreiben Sie einmal Attribute, die sich gegenseitig ausschließen (z.B. 'wid .5 wid 1'), gilt die zuletzt gemachte Angabe (im Beispiel 'wid 1').

Sie können alle Primaries mit allen Attributen belegen. PIC sucht sich die Attribute aus, die für das jeweilige Primary auch eine Bedeutung haben. In der Befehlszusammenfassung habe ich deshalb nur die Attribute aufgeführt, die auch ausgewertet werden.

Ein Befehl endet am Zeilenende, es sei denn, Sie haben die Zeile durch den Backslash ('\') aufgebrochen. Sie können einen Befehl auch durch ein Semikolon (';') beenden, um mehrere Befehle in eine Zeile zu schreiben.

3.5.1 box – Rechteck

Das Rechteck ist eines der am einfachsten zu verwendenden Primaries. Der Befehl, der ein Rechteck ausgibt, beginnt immer mit dem Token 'box'.

Der Befehl 'box' erzeugt, wenn er allein steht, an der aktuellen Standposition ein Rechteck in den Defaultgrößen. Das sieht so aus:

Abbildung 3.4: box.pic

```
# box.pic, die Default-Box wird ausgeben
# Bild Nr. 5

box
```

Nachdem die Box gezeichnet wurde, befindet sich der Standpunkt in Bewegungsrichtung am gegenüberliegenden Ende der Box. Ist die Bewegungsrichtung also Osten ('east'), wird der Standpunkt von der Mitte der westlichen Seite der Box (vor dem Zeichnen der Box) an die Mitte der östlichen Seite verlegt.

Damit der Rechteck–Befehl etwas flexibler wird, besteht die Möglichkeit, die Rechteckgröße zu verändern. Sie schreiben dazu hinter das einleitende Token, irgendwo in die Attributliste, je nach Wunsch:

```
'wd' expr | 'wid' expr | 'width' expr
```

um die Breite zu verändern, oder:

```
'ht' expr | 'height' expr
```

um der Höhe einen anderen Wert zu geben. 'expr' steht für eine beliebige Expression.

Falls Sie mit der Rechteckumrandung, eine durchgezogene Linie ('solid'), nicht einverstanden sind, können Sie ihr durch die Angabe eines der folgenden Token ein neues Aussehen geben:

```
dashed, dotted, invis
```

Diese heißen im Klartext: Gestrichelt, gepunktet und unsichtbar. Zusätzlich können Sie die Box noch mit einem ihrer Eckpunkte an einen bestimmten, anzugebenden Punkt setzen. Die Eckpunkte sind wie folgt benannt:

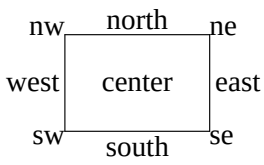


Abbildung 3.5: corner.pic

```
# corner.pic, eine Box mit den Eckbezeichnungen wird ausgegeben
# Bild Nr. 6
define B ' box invis at 1st box '

textwid = 2/3 * textht

box "center"
B "north" above
"ne" with sw at last box.ne
B "east" right
"se" with nw at last box.se
B "south" under
"sw" with ne at last box.sw
B "west" left
"nw" with se at last box.nw
```

Die Neuplazierung eines Rechteckes geht wie folgt von statten: Sie schreiben hinter das Token 'with' die Ecke (z.B. 'top', 'south', 'center'), mit der Sie die Box setzen wollen. Danach (oder davor) schreiben Sie irgendwo das Token 'at' gefolgt von der gewünschten Position, an die Sie Ihr Primary setzen wollen.

Schreiben Sie keine 'at' Angabe, sondern nur die 'with' Angabe, wird die Box mit der angegebenen Ecke am aktuellen Standpunkt abgelegt. Schreiben Sie dagegen nur das 'at' Attribut in den Box–Befehl, wird als 'Ecke' die Boxmitte, also 'center', angenommen.

Als weiteres Box–Attribut können Sie auch Text, incl. Angaben zu dessen Ausrichtung und Plazierung, aufnehmen. Die Ausrichtungsangaben sagen aus, ob der Text zentriert, links– oder rechtsbündig ausgegeben werden soll. Dazu wird der längste Textstring in der Box zentriert und die anderen Strings relativ zu diesem im gewünschten Format ausgegeben. Die Ausrichtungsangaben lauten:

Ausrichtung	Befehl
Linksbündig	ladjust, ljust, la
Rechtsbündig	radjust, rjust, ra
Zentriert	centered

'centered' ist der Defaultwert, braucht also nicht geschrieben werden. Standardmäßig wird der Text in die Box geschrieben. Wollen Sie ihn jedoch über, unter, rechts oder links von der Box stehen haben, können Sie dies durch die Token:

```
above, under, right, left
```

erreichen.

Falls Sie in einem Box–Befehl die Größen ändern und diese dann in die darauffolgende Box übernehmen möchten, ohne die Defaultvariablen zu verändern, können Sie eine solche Übernahme durch das Token 'same' in der Attributliste erreichen.

Hier noch ein hoffentlich anschauliches Beispiel zu den verschiedenen 'box'–Attributen, mit dem dazugehörenden PIC–Programm, in dem die Möglichkeiten des 'box'–Befehls ausgenutzt werden:

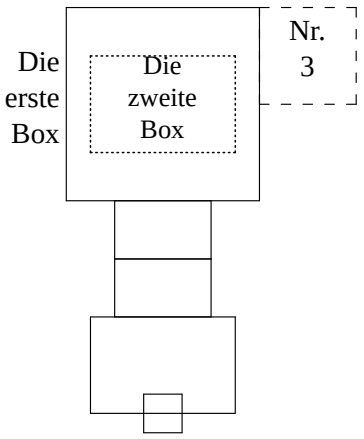


Abbildung 3.6: boxes.pic

```
# boxes.pic, verschiedene Boxen mit Text
# Bild Nr. 7
```

```
BOX:    box    \ Label BOX und einleitendes Schluesselwort
        wid 1 ht 1 \ Groessenangaben
        "Die " "erste " "Box " \ Text
        rjust left # Textausrichtung

box "\\small Die" "zweite" "Box" \
    dotted      \ Gepunktete Umrandung
    at last box \ Positionierung in die Mitte der letzten Box
    ;

boxht = boxwid = .5     # Neue Defaultgroessen

box "Nr." str(3.0,0,0) with .nw at 1st box.ne dashed

down    # Bewegungsrichtung nach unten

move to the bottom of BOX  # Standpunkt Umsetzung

# Boxen in Laufrichtung
box ht boxht-.2            # Kleine Box
box same                   # Die gleiche Box
box wid .75
box wid .2 ht .2 with center
     # Box mit Mitte an aktuellen Standpunkt,
     # falls die with-Angabe gefehlt haette, waere
     # dies Box mit ihrem oberen Ende an den aktuellen
     # Standort gesetzt worden.
```

Abschließend noch die Schreibweise des ’box’–Befehls:

```
"box"
{ "wid" expr | "ht" expr | "same" | text textattr |
  "with" corner | "at" coord |
  "solid" | "dashed" | "dotted" | "invis"
}
```

3.5.2 line – Linie

Linien sind die wohl wichtigsten Primaries. Mit dem Linien–Befehl ’line’ können einzelne Linien, oder ganze Linienzüge ausgegeben werden. Die Liniendarstellung kann wie beim ’box’–Befehl variieren. Start– und Endpunkt können beliebig gesetzt werden. Auch eine Beschriftung der Linien ist möglich. Diese Beschriftung wird immer in Bezug auf die den Linienzug umgebende Rechteck–Hülle ausgerichtet, mit ’above’, ’right’, etc. Zusätzlich können Sie die Linie noch mit Pfeilspitzen versehen. Falls Sie die Linien von einem geschlossenen Primary (Rechteck, Kreis oder Ellipse) zu einem anderen ziehen, kann die Linienlänge so gekürzt werden, daß der End– oder Anfangspunkt auf dem Rand dieses Primaries zu liegen kommt.

Defaultmäßig wird die Linie in Laufrichtung in der angegebenen Schrittweite (standardmäßig nach rechts um .5inch) gezogen. Eine Linie ohne Attribute sieht also so aus:

```
# line.pic, die Default-Linie in Laufrichtung WEST wir erzeugt
# Bild Nr. 8
```

Abbildung 3.7: line.pic

```
line
```

Mit ’up 1; line;’ folgende Linie erzeugt:

Abbildung 3.8: line-up.pic

```
# line-up.pic, Eine Linie von 1inch Laenge wird in
#              Laufrichtung NORTH gezogen
# Bild Nr. 9

up 1 ; line
```

Der Linienstart kann auch durch ’from coord’ angegeben werden. Das Linienende durch eine alleinstehende relative Koordinate, oder durch eine absolute Koordinate mit vorangestelltem ’to’. Werden mehrere Linienendpunkte angegeben, wird ein Linienzug durch alle Endpunkte gezeichnet. Relative Koordinaten müssen in einem solchen Linienzug durch ein vorangestelltes ’then’ gekennzeichnet werden. Wird eine relative Koordinate angegeben, ändert sich die Laufrichtung entsprechend. Wird diese relative Koordinate aus mehreren Richtungen zusammengesetzt (z.B ’right .5 up .5’), gilt die jeweils letzte Richtungsangabe für die Laufrichtungsänderung (im Beispiel also ’up’). Eine bloße Zahlenangabe n in der Attributliste erzeugt eine Linie in der Defaultrichtung um n Inch.

Hier ein Beispiel zum Linienpfad:

```
# path.pic, ein Linienzug in Form eines Hauses wird gezogen
# Bild Nr.10

line up then up right then down right \ Umrandung
     then down then left left         \ Umrandung
     to (1,.5) to (0,.5) to (1,0)     # Inhalt
```

Steht in einem Linienpfad als erste Koordinate eine ’then’–Koordinate, wird zuerst eine Linie in Defaultrichtung gezogen. Der Befehl:

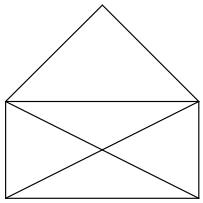


Abbildung 3.9: path.pic

`line then down`

erzeugt, da die Defaultrichtung ’east .5’ ist, folgende zwei Linien:

Abbildung 3.10: lines.pic

```
# lines.pic, 2 Liniensegmente werden gezeichnet
# Bild Nr. 11

line then down # entspr. line right then down, wobei right die
               # Default-Richtung ist.
```

Wird eine Linie von einem geschlossenen Primary zu einem andern gezogen, können Sie durch die Angabe des Tokens ’chop’ das Linienende und den Linienanfang so kürzen, daß beide auf dem jeweiligen Primary–Rand enden. Dazu folgendes Beispiel, welches auch gleich die Beschriftung von Linien zeigt:

```
# chop.pic, eine Verwendung des Linienchoppens
# Bild Nr. 12

box with c "Box 1"        # Boxen, an denen gechoppt werden soll
box at (-1,-1) "Box 2"
box at (1,-1)  "Box 3"

line from 1st box to 2nd box chop \   beide Linienteile werden am Box-Rand
                                  #   gechoppt
     "Linie 1\\hspace{2pt}" left with e at last line
line from 1st box to 3rd box chop #   wie oben
     "\\hspace*{2pt}Linie 2" right with w at last line
```

Durch die Angabe von ’start’ oder ’end’ hinter ’chop’ haben Sie die Möglichkeit, entweder Linienstart oder Linienende zu kappen.

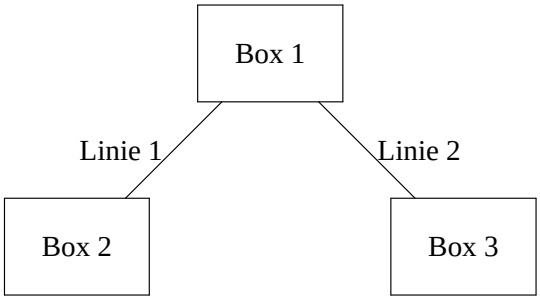


Abbildung 3.11: chop.pic

Schließlich können Sie noch Pfeilspitzen an die Linien zeichnen lassen. Durch ’$->$’ lassen Sie eine Pfeilspitze am Linienende, durch ’$<-$’, eine am Linienstart und durch ’$<->$’ beide setzen. Hierzu wieder ein Beispiel:

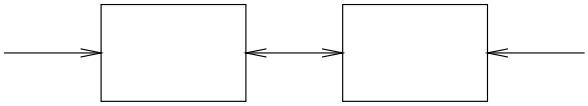


Abbildung 3.12: line-arrow.pic

```
# line-arrow.pic, Linien mit Pfeilspitzen
# Bild Nr. 13


line ->  # Pfeilspitze in Laufrichtung, am Linienende
box
line <-> # Zwei Pfeilspitzen
box
line <-  # Pfeilspitze gegen die Laufrichtung am Linienanfang
```

Hier noch die Schreibweise des Linien–Befehls:

```
"line"
{  "from" coord | "to" coord | relcoord | "then" relcoord |
   expr | text textattr | "chop" [ "start" | "end" | "both" ] |
  "solid" | "dashed" | "dotted" | "invis" |
  "->" | "<-" | "<->"
}
```

3.5.3 arrow – Pfeil

Der ’arrow’–Befehl funktioniert wie der ’line’–Befehl, mit dem Unterschied, daß eine Pfeilspitze in Laufrichtung (’$->$’) voreingestellt ist. Der Befehl ’arrow’ ergibt folgenden Pfeil:

Abbildung 3.13: arrow.pic

```
# arrow.pic, der Default-Pfeil
# Bild Nr. 14

arrow
```

Die Pfeilspitze kann durch ’< −’ und ’< − >’ verändert werden. Die Schreibweise lautet also:

```
"arrow"
{  "from" coord | "to" coord | relcoord | "then" relcoord | expr |
   text textattr | "chop" [ "start" | "end" | "both" ] |
   "solid" | "dashed" | "dotted" | "invis" |
   "->" | "<-" | "<->"
}
```

3.5.4 move - Bewegung

Mit Hilfe des ’move’–Befehls kann der Standort auf einen bestimmten Punkt gesetzt werden. Er kann auch durch die Angabe einer relativen Koordinate verändert werden. Mit ’from coord’ kann ein Startpunkt für die Bewegung angegeben werden. Der ’move’–Befehl verhält sich wie der ’line’–Befehl, mit dem Unterschied, daß das Attribut ’invis’ voreingestellt ist. Ein Beispiel hierzu:

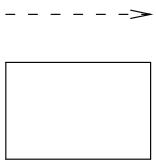


Abbildung 3.14: move.pic

```
# move.pic, Bewegung ausfuehren
# Bild Nr. 15

box                      # Box zeichnen
move up from last box.w  # .5 inch ueber der Mitte der
                         # linken Seite der Box
right boxwid             # Defaultrichtung nach rechts,
                         # Laenge = .75 inch
line dashed ->           # gestrichelte Linie mit Pfeilspitze
                         # in Laufrichtung
```

Die Schreibweise lautet wie folgt:

```
"move"
{  "from" coord | "to" coord | relcoord | "then" relcoord | expr |
   text textattr | "chop" [ "start" | "end" | "both" ] |
   "solid" | "dashed" | "dotted" | "invis" |
   "->" | "<-" | "<->"
}
```

3.5.5 ellipse - Ellipse

Der Befehl zum Zeichnen einer Ellipse ist mit dem zum Zeichnen einer Box identisch. Anstelle eines Rechteckes wird eine Ellipse ausgegeben. Deshalb zur Erklärung nur ein kleines Beispielbild:

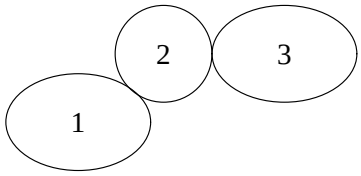


Abbildung 3.15: ellipse.pic

```
# ellipse.pic, drei Ellipsen
# Bild Nr. 16

ellipse "1" # wid .75 ht .5 ist default
ellipse wid ellipseht with .sw at last ellipse.ne "2" # Kreis
ellipse "3"
```

Die Schreibweise des ’ellipse’–Befehls:

```
"ellipse"
{ "wid" expr | "ht" expr | "same" | text textattr | "with" corner |
  "at" coord | "solid" | "dashed" | "dotted" | "invis"
}
```

3.5.6 circle – Kreis

Auch der Kreis–Befehl verhält sich wie der Rechteck–Befehl. Statt ’wid’ und ’ht’, kann allerdings nur der Radius des Kreises durch ’rad expr’ oder durch eine alleinstehende ’expr’ verändert werden. Hier ein Beispiel:

```
# circle.pic, einige Kreise mit wachsenden und wieder fallenden
#             Radius erzeugen
# Bild Nr. 17
```

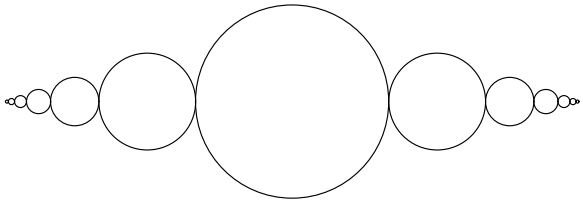


Abbildung 3.16: circle.pic

```
# circlerad = .5 ist voreingestellt

circle 1/32*circlerad
circle 1/16*circlerad
circle 1/8*circlerad
circle 1/4*circlerad
circle 1/2*circlerad
circle
circle 2*circlerad
circle
circle 1/2*circlerad
circle 1/4*circlerad
circle 1/8*circlerad
circle 1/16*circlerad
circle 1/32*circlerad
```

Abschließend die Schreibweise:

```
"circle"
{ "rad" expr | expr | "same" | text textattr |
  "with" corner | "at" coord |
  "solid" | "dashed" | "dotted" | "invis"
}
```

3.5.7 arc – Kreisbogen

Der ’arc’–Befehl ist recht vielseitig. Es können Angaben zum Start– und Endpunkt des Kreisbogens, zu dessen Mittelpunkt, Radius, Drehwinkel und –richtung gemacht werden. Es genügt aber in den meisten Fällen, nur wenige bis gar keine Punkte festzulegen. PIC füllt die fehlenden Werte mit Standardangaben auf oder berechnet sie nach einem Standardverfahren.

Wenn Sie z.B. einen Kreisbogen zwischen zwei Punkten ziehen wollen, brauchen Sie nicht den Mittelpunkt angeben, PIC berechnet ihn für Sie. Der Radius wird so gewählt, daß er der Entfernung zwischen Start– und Endpunkt entspricht. Dadurch bekommt der Kreisbogen eigentlich immer eine annehmbare Wölbung. Durch das Vertauschen von Start– und Endpunkt können Sie bestimmen, in welche Richtung sich der Bogen wölben soll. Der Kreisbogen wird standardmäßig immer gegen den Uhrzeigersinn (’ccw’ – counterclockwise) gedreht. Das entspricht dem kleineren Bogen. Möchten Sie lieber den größeren Bogen gezeichnet haben, können Sie dies durch die Angabe von ’cw’ (clockwise, im Uhrzeigersinn), in der Attributliste des Primary–Befehls,

erreichen. Der Bogen wird dann im Uhrzeigersinn gedreht. Um dies zu verdeutlichen, wieder eine kleine Zeichnung:

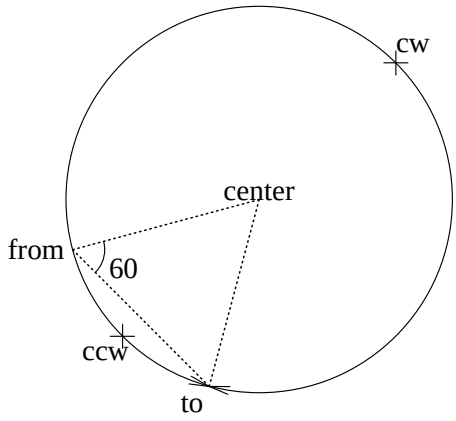


Abbildung 3.17: arc.pic

```
# arc.pic, zeigt die Ermittlung des arc.center
# Bild Nr. 18

# Makro: Kreuz an $1 malen
define CROSS ’
  move to $1 then up 1/16
  line down 2/16
  move up 1/16 left 1/16
  line right 2/16
’

circle rad .5 invis # Nur zur Positionierung

# Die beiden Kreisboegen
arc -> from last circle.nw to last circle.se
arc -> dashed cw from last circle.nw to last circle.se


# Linien, die zeigen, wie der Mittelpunkt ermittelt wurde
line dotted from 2nd last arc.start to 2nd last arc.end
line to last arc dotted
line to 2nd last arc.start dotted

# Beschriftungen, len n gibt tatsaechliche Textlaenge an
box wid boxwid/2 "from"  with .e at last arc.start invis
circle rad circlerad/2 "to" with ne at last arc.end invis
circle rad circlerad/2 "ccw" with .ne at 2nd last arc.m invis
circle rad circlerad/2 "cw" with .sw at last arc.m invis

# Winkel zeichnen
```

```
arc from 1/6 between 1st  line.start and 1st line.end \
    to 1/6 between last line.end and last line.start
"60" right with nw at last arc.m

# Mitte des Vollkreises einzeichnen
"center" with .s at 1st arc

# Kreuze an den beiden arc-Mitten
CROSS (1st arc.m)
CROSS (2nd arc.m)
```

Der gestrichelte Bogen ist der im Uhrzeigersinn, der durchgezogene der gegen den Uhrzeigersinn. Um die Drehrichtung des Kreisbogens zu verdeutlichen, sind Pfeilspitzen in die Drehrichtung eingetragen. Der Mittelpunkt des Kreisbogens (der, der auf dem Bogen liegt, nicht der Mittelpunkt des umgebenden Kreises) heißt 'middle', 'mid' oder einfach nur 'm', an dieser Stelle befindet sich auf der Zeichnung jeweils ein Kreuz. Die gepunkteten Linien verdeutlichen, wie der Mittelpunkt des umgebenden Kreises ermittelt wurde.

Es bleibt Ihnen freigestellt, auch noch den Mittelpunkt ('center') des Kreises anzugeben, der den Kreisbogen umgibt. In diesem Fall, wird der Kreisbogen soweit gedreht, bis er die Linie zwischen Mittelpunkt und Endpunkt schneidet. Der Mittelpunkt kann durch 'at coord' angegeben werden. Verschoben werden kann der Kreisbogen nicht.

Fehlt der Startpunkt, wird der aktuelle Standpunkt als Startpunkt angenommen. Fehlt auch noch der Endpunkt, wird der Kreisbogen standardmäßig um 90 Grad gedreht. Diesen Drehwinkel können Sie durch eine Expression hinter den Token 'cw' oder 'ccw' ändern. Der Wert der Expression gibt den Winkel an und sollte zwischen -360 und 360 liegen, wobei auch die beiden Grenzwerte benutzt werden dürfen. Werden die Grenzen überschritten, wird ein Vollkreis gemalt. Ist der Drehwinkel negativ, ändert sich die Drehrichtung. 'ccw -90' entspricht also 'cw 90'.

Fehlt auch noch der Mittelpunkt, wird der Radius des Arcs zur Festlegung des Mittelpunkts zu Hilfe genommen. Er kann durch das Ändern der Defaultvariable 'arcrad' oder durch 'rad expr' in der Attributliste des Befehls verändert werden. 'same' übernimmt den Radius aus dem vorigen Kreisbogen. Wo der Mittelpunkt schließlich hingelegt wird, hängt auch noch von der Laufrichtung ab, die vor dem Zeichnen des Arcs gilt. Stellen Sie sich das Bogenziehen am besten wie das Verbiegen eines Stücks Draht vor. Die Bewegungsrichtung ändert sich nach dem Zeichnen des Bogens in die Richtung, in die das Ende des fiktiven Drahtes zeigen würde. Um das alles zu verdeutlichen, wieder eine kleine Demografik:

```
# worm.pic, zeigt die Richtungsumsetzung bei der Verwendung von ARC
# Bild Nr. 19

down
arc
arc
arc cw
arc cw
arc ccw
line 1/2*linewid
arc
```

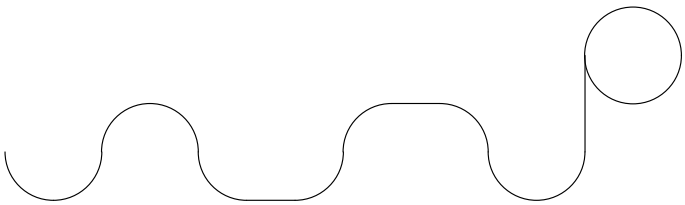


Abbildung 3.18: worm.pic

```
arc cw
line 1/2*linewid
arc cw
arc ccw 180
line
arc cw 360
```

Wie bei den anderen linienartigen Primaries können Sie auch beim ’arc’ Pfeilspitzen einzeichnen lassen, die Strichart wählen und Texte anbringen. Texte werden übrigens immer auf der Mitte des Bogens ausgegeben. Durch Leerstrings und Blanks können sie die Texte noch etwas verschieben. Das Choppen von Kreisbögen ist in der vorliegenden Version nicht möglich. Hier ein Beispiel zur Verwendung von Attributen:

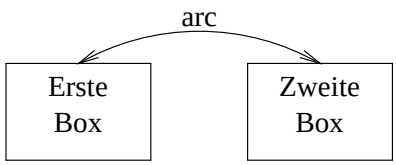


Abbildung 3.19: arc2.pic

```
# arc2.pic, arc mit zwei Stuetzpunkten und Text
# Bild Nr. 20

box "Erste" "Box"
move
box "Zweite" "Box"
arc <-> from top of last box to top of first box "arc" above
```

Der ’arc’–Befehl hat folgende Syntax:

```
"arc"
{ "rad" n | "same" | text textattr | "at" coord | "from" coord | "to" coord |
  expr | ("cw" | "ccw" ) [ expr ] | relcoord |
  "solid" | "dashed"| "dotted" | "invis" |
  "->" | "<-" | "<->"
}
```

3.5.8 spline – Bézier–Kurve

Der Spline–Befehl funktioniert im Prinzip wie der ’line’–Befehl, nur daß im Unterschied zu einem Linienzug eine Kurve ausgegeben wird. Eine Kurve besteht aus einer oder mehreren Bézier–Kurven. An drei Linienpunkte wird jeweils eine Bézier–Kurve gelegt, wobei an den Mittelpunkten der beiden Teillinien, diese jeweils tangential zu der Kurve verlaufen. Sie sind die einzigen Punkte die die Kurve berührt (Ausnahme hiervon: Anfang und Ende der Gesamtkurve); sie sind Verlaufspunkte der Kurve. Die Endpunkte dagegen sind nur Stützpunkte und nicht Bestandteil der Kurve. Eine Ausnahme bilden lediglich der Start– und der Endpunkt des gesamten Linienzuges. Durch diese beiden Punkte geht die Kurve auf jeden Fall.

Hier ein Beispiel zu einer solchen Kurve. Der Linienzug, der die Kurve umgibt ist gestrichelt eingezeichnet:

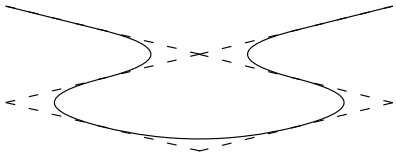


Abbildung 3.20: spline.pic

```
# spline.pic, Kurve, mit umgebender Linie, zeichnen
# Bild Nr. 21

line right 1 down 0.25 \  umgebende Linie
  then left 1 down 0.25 \
  then right 1 down 0.25 \
  then right 1 up 0.25 \
  then left 1 up 0.25 \
  then up 0.25 right 1 \
  dashed

spline from start of last line \ Bezier-Kurve
  right 1 down 0.25 \
  then left 1 down 0.25 \
  then right 1 down 0.25 \
  then right 1 up 0.25 \
  then left 1 up 0.25 \
  then up 0.25 right 1
```

Die Attribute eines Splines sind denen von ’line’, ’arrow’ und ’move’ gleich. Hier ist die Befehlszusammenfassung:

```
"spline"
{  "from" coord | "to" coord | relcoord | "then" relcoord | expr |
   text textattr | "chop" [ "start" | "end" | "both" ] |
   "solid" | "dashed" | "dotted" | "invis" |
   "->" | "<-" | "<->"
}
```

3.5.9 Texte

Auch Texte können ein eigenständiges Primary bilden, nämlich dann, wenn ein String, evtl. auch der leere String “”, am Anfang eines Befehls steht. Das Textprimary wird dann so behandelt, als ob es von einer unsichtbaren Box umgeben wäre. Tatsächlich ist die interne Darstellung eines Textes mit der einer Box bis auf den Namen identisch. Deshalb können auch sämtliche Box–Attribute verwendet werden.

Die einzelnen Strings werden untereinander wie gewöhnlich ausgerichtet. Die einzelnen Primaries werden in Laufrichtung aufgereiht. Zur Verdeutlichung ein Beispiel:

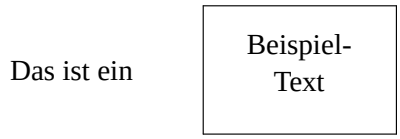


Abbildung 3.21: text.pic

```
# text.pic, Texte ausgeben, font ist Pica
# Bild Nr. 22

"Das ist ein" len 0 wid 1

"" "Beispiel-" "Text" "" len 0 wid 1 solid # mit Umrandung
```

Die Schreibweise Text–Befehls ist der des Box–Befehls ähnlich, ’wid expr’ und ’ht expr’ ändern die umgebende Hülle:

```
"text"
{ "wid expr" | "ht" expr | text textattr | "with" corner |
  "at" coord | "solid" | "dashed" | "dotted"| "invis"
}
```

3.6 Die Benutzung von Laufrichtungen

Noch eine abschließende Bemerkung zur Benutzung von Laufrichtungen und Schrittweiten. Zu Beginn ist die Laufrichtung von links nach rechts voreingestellt. Es gibt nur 4 Laufrichtungen. Die Schrittweite beträgt standardmäßig 0.5inch und kann verändert werden. Diese Schrittweite nimmt Einfluß auf ’line’, ’arrow’, ’move’ und ’spline’, es werden jeweils die Default–Längen dieser Primaries verändert. Die Laufrichtung und die Schrittlänge werden beim ffnen eines neuen Grafik–Blockes beibehalten. Wird in einem Laufrichtungs–Befehl nur die Richtung, nicht aber die Schrittweite angegeben, wird diese auf ihren Defaultwert (0.5inch) gesetzt. Die Laufrichtung kann sich bei der Verwendung der Befehle ’line’, ’move’, ’spline’ und ’arc’ verändern, was in den meisten Fällen sinnvoll genutzt werden kann. Es ist so nicht nötig, nach der Benutzung eines der oben angegebenen Primaries die Laufrichtung von Hand umzuändern.

Das Setzen von Primaries geschieht, falls durch ’with’ und ’at’ keine anderen Angaben gemacht werden nach folgenden Schema:

Das Primary wird mit dem gegenüberliegenden Ende der Laufrichtung gesetzt (falls die Laufrichtung z.B. nach rechts zeigt, wird das Primary mit der linken Seite, bzw. mit dem Startpunkt, auf die aktuelle Standposition gelegt). Danach wird der Standort in die Mitte der gegenüberliegenden Seite des Primaries, bzw. an das Primary–Ende, gelegt. Wird die Laufrichtung danach durch einen Laufrichtungsbefehl geändert, wandert der Standort, bei geschlossenen Primaries, auf der Primaryhülle an die Stelle, an die sie liegen würde, wenn die Richtungsänderung vor dem Aufruf des Primary–Befehls vorgenommen worden wäre. Dazu ein Beispiel:

```
box; up; arrow
```

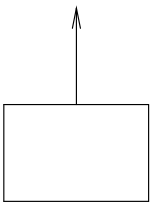


Abbildung 3.22: dir1.pic

```
# dir1.pic, Punktumsetzung bei Richtungsaenderung
# Bild Nr. 23

box
up     # Aendern der Laufrichtung und Umsetzen des Standortes auf
       # last box.top
arrow
```

Führen Sie die Laufrichtungsänderung innerhalb eines linienartigen Primaries durch, wird der Standort nicht verändert. Auch hierzu wieder ein Beispiel:

```
box; arrow up
```

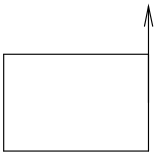


Abbildung 3.23: dir2.pic

```
# dir2.pic, keine Punktumsetzung, wenn Richtungsaenderung innerhalb eines
#           Primaries geschieht
# Bild Nr. 24

box
arrow up # Nur Laufrichtung umsetzen
```

Durch diese beiden Möglichkeiten der Standortumsetzung können Sie sich, durch geschickte Anwendung, einige ’move’–Statements sparen.

3.7 Der Primaryblock und Labels

Durch einen Primaryblock können Primary–Befehle zusammengefasst werden. Innerhalb eines Grafik–Blocks wird ein neues Koordinatensystem angelegt. Das hat den Vorteil, daß Sie innerhalb eines Blocks unabhängig von anderen Teilen Ihrer Grafik arbeiten können.

Ein Block wird durch eine geöffnete eckige Klammer ’[’ eingeleitet, darauf folgt eine beliebige Anzahl von Primary–Befehlen, worunter sich auch weitere Blöcke (Unterblöcke) befinden dürfen. Abschließend folgt eine geschlossene eckige Klammer ’]’, optional gefolgt von Attributen, die den Block abschließen. Bitte achten Sie darauf, daß Sie den letzten Befehl innerhalb eines Blockes durch ein Semikolon oder Zeilenende abgeschlossen haben. Die Attribute ’with’ und ’at’ dienen, wie im ’box’–Befehl, dazu, den gesamten Block zu verschieben. Außer dieser Verschiebung werden noch die Attribute ’solid’, ’dashed’, ’dotted’ und ’invis’ beachtet. ’invis’ ist voreingestellt. Durch sie kann eine Blockumrandung ausgegeben werden.

Variablenwerte werden zwar selbstständig in den neuen Block übernommen, wenn Sie auf der rechten Seite einer Zuweisung vorkommen, geänderte Werte werden jedoch nicht wieder zurückgeschrieben. Falls Sie dies jedoch möchten, können Sie dies durch den Befehl:

```
"export" "(" identifier { "," identifier } ")" eostm
```

(z.B. ’export(Variable1, V3, A, B);’)

tun. Es ist so möglich, eine Art von Parameterübergabe zu bewerkstelligen. Wollen Sie für eine bestimmte Variable keine Wertübernahme von außen, müssen Sie diese Variable am Blockanfang auf 0 setzen.

Labels gelten lokal in einem Block. Hier ein Beispiel, wie Sie Labels aus einem Unterblock ansprechen können. Labels können übrigens erst nach ihrer Definition angesprochen werden:

```
# label.pic, Label in einem Unterblock ansprechen
# Bild Nr. 25

[
       up
       circle
  BOX: box
]

# Defaultrichtung ist hier wieder right
line -> from top of last [].BOX up
```

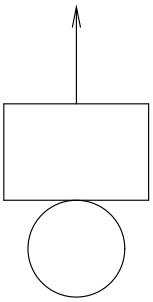


Abbildung 3.24: label.pic

Optional hätten Sie auch:

```
line up -> from last [].1st box.t
```

oder

```
line up -> from last block.last box.t
```

oder Ähnliches schreiben können, um den gleichen oberen Pfeil zu erhalten.

Abschließend noch zwei Beispiele für die Blockverschiebung:

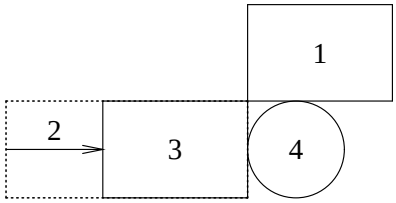


Abbildung 3.25: drag1.pic

```
# drag1.pic, Blockverschiebung
# Bild Nr. 26

box "1"
[
  arrow "2" above
  box "3"
] with ne at last box.sw dotted # den gesamten Block verschieben
circle "4"
```

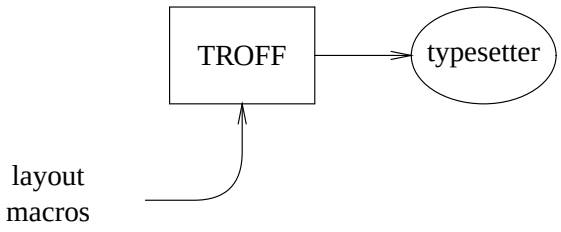


Abbildung 3.26: drag2.pic

```
# drag2.pic, Blockverschiebung
#            nach Brian W. Kernighan,
#            aus: PIC A LANGUAGE FOR TYPESETTING GRAPHICS
# Bild Nr. 27

circlerad  = 2 * circlerad

box "TROFF"
arrow
ellipse "typesetter"
[          circle invis "layout" "macros"
           spline right boxht then up boxht  ->
        C: last spline.end
] with C at last box.s  # Block mit Koordinate
                        # auf einen anderen Punkt setzen
```

4. Die For–Schleife

Die For–Schleife gleicht denen, die in Programmiersprachen verwendet werden. Sie hat folgende Struktur:

```
forloop = "for" identifier "=" expr1 "to" expr2 ["step" expr3] "do" "{"
    statements
"}"
```

Die geschweiften Klammern gehören fest zur Syntax und dürfen im Gegensatz zum Step–Teil nicht weggelassen werden. ’expr1’, ’expr2’ und ’expr3’ bezeichnen Expressions.

Die Anweisungen in dem Anweisungsblock werden solange wiederholt ausgeführt, bis die Variable ’identifier’ (welche anfangs mit ’expr1’ initialisiert wurde) den Wert von ’expr2’ über–, bzw. bei negativen Stepwert, unterschritten hat. Die Variable wird dazu um den Wert von ’expr3’ inkrementiert, bzw. dekrementiert. Fehlt der Stepwert ’expr3’, wird die Variable jeweils um 1 erhöht. Die Werte von ’expr1’, ’expr2’ und ’expr3’ können während des Schleifendurchlaufs nicht verändert werden, wohl aber der Inhalt der Schleifenvariable (’identifier’).

Die Abbruchgrenze wird vor jedem Schleifendurchlauf überprüft. Ist Sie verletzt, wird der Schleifeninhalt nicht mehr ausgeführt. Hier das Ergebnis eines Programms, das fünf Kreise malt:

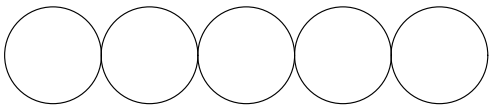


Abbildung 4.1: for.pic

```
# for.pic, zeichnen von 5 Kreisen,
#          Beispiel fuer die For-Schleife
# Bild Nr. 28

for i=1 to 5 do {
  circle
}
```

Schleifen können bis zu einer Tiefe von 20 ineinander verschachtelt werden.

5. Die If–Then–Else–Anweisung

Auch die If–Then–Else–Anweisung ist aus den Programmiersprachen entlehnt. Sie funktioniert in gleicher Weise: Die Expression hinter dem ’if’ wird auf ihren logischen Wahrheitswert überprüft. Ist dieser wahr, wird der Programmteil hinter dem ’then’ ausgeführt. Ist er hingegen falsch, wird der Teil hinter dem ’else’ ausgeführt. Dieser ’else’–Teil ist optional, darf also weggelassen werden. Die Anweisungsblöcke müssen wie in der For–Schleife durch geschweifte Klammern umgeben sein. Die bedingte Anweisung hat folgende Schreibweise:

```
if_then_else = "if" expr "then" "{"
    statements
"}"
[ "else" "{"
    statements
"}"
]
```

Anwendung findet das ’if’ beispielsweise in For–Schleifen, um in Abhängigkeit der Schleifenvariable bestimmte Anweisungen auszuführen, wenn Sie z.B. den Wunsch haben abwechselnd vier Kreise und vier Rechtecke aufs Papier zu bringen.

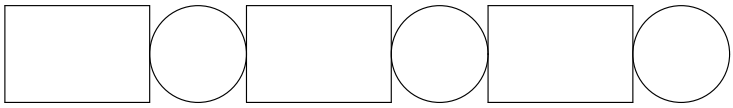


Abbildung 5.1: if.pic

```
# if.pic, Abwechselnd je 4 Boxen und vier Kreise ausgeben, Beispiel fuer die
#         If-Anweisung
# Bild Nr. 29

j = false
for i = 1 to 6 do {
  if j then {
    circle
    j=false
  }
  else {
```

```
    box
    j=true
  }
}

# einfacher: for i=1 to 4 do { box; circle; }
```

Das obige Beispiel dient nur zur Verdeutlichung. In der Praxis wäre es besser, folgendes zu schreiben:

```
for i=1 to 4 do {
    box; circle
}
```

Ein sinvolleres Beispiel für die Verwendung folgt in dem Kapitel mit den Beispielbildern (’s–expr.pic’).

Ein anderes wichtiges Anwendungsgebiet finden Sie in dem folgenden Kapitel über den Makroprozessor. Die bedingte Anweisung wird in PIC dazu verwendet, Abbruchkriterien für rekursive Makroaufrufe zu ermöglichen.

6. Der Makroprozessor

6.1 Überblick

Makros dienen hauptsächlich dazu, Primary–Befehle, die im PIC–Text in gleicher Reihenfolge mehrmals vorkommen, zusammenzufassen und namhaft zu machen. An den Stellen im Text, an denen die Befehls–Sequenzen stehen sollen, braucht nur noch der Name des Makros aufgeführt werden und nicht mehr die gesamte Sequenz. Makros können auch dazu dienen, Befehle, die logisch zusammengehören zusammenzufassen, um den PIC–Text übersichtlicher zu gestalten. In Grenzen können Makros zur Definition neuer Primaries verwendet werden.

Makros gleichen in etwa den Prozeduren der höheren Programmiersprachen. Wenn der Makroblock einem Grafik–Block entspricht, ist sogar eine gewisse Lokalität von Variablen gewährleistet. In Makros können Sie bis zu neun Parameter verwenden und natürlich auch andere Makros aufrufen, insbesondere das gleiche Makro (Rekursion). Es ist jedoch nicht möglich (und auch nicht unbedingt nötig), innerhalb eines Makros ein anderes Makro zu definieren. Sie sollten es auch vermeiden, die Makros so zu definieren, daß Sprachstrukturen zerrissen werden.

6.2 Die Definition von Makros

Makros werden wie folgt definiert:

```
"define" identifier qoute Makrotext quote
```

(z.B. `define NAME ' box; line; '`)

Der Identifier nach 'define' ist der Name des Makros. Das auf den Namen folgende Zeichen quote (im Beispiel ein Hochkomma) kann beliebig gewählt werden, es dient als Begrenzerzeichen für den Makrotext, darf also im Makrotext selbst nicht enthalten sein. Das gleiche Zeichen dient also sowohl als Makrobeginnzeichen als auch als Makroendezeichen. Es ist am günstigsten, wenn Sie ein Zeichen wählen, das nicht Bestandteil der PIC–Syntax ist, also z.B. das Hochkomma oder ein Ausrufungszeichen. In Strings dürfen diese Zeichen übrigens vorkommen, da diese nicht interpretiert werden. Der Makrotext darf ansonsten jedoch prinzipiell beliebig sein. Er wird erst mit dem Finden des Namens im Befehlstext expandiert. Es wird also immer dann, wenn der Name eines Makros im Befehlstext gefunden wird, der entsprechende Makrotext eingesetzt und als Befehlsliste interpretiert. Hier zwei etwas komplexere Beispiele, wie Sie Makros verwenden können:

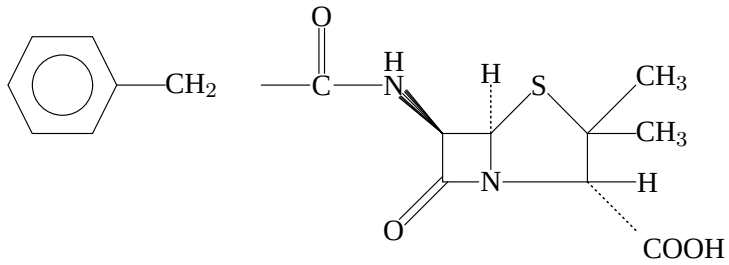


Abbildung 6.1: penicillin.pic

```
# penicillin.pic, Strukturformel von Penicillin G (siehe CACM, 8/86
#                 Vol. 29, Figure 5)
# Bild Nr. 30

textwid=textht/2

# Makros

# Kohlenstoff-Ring
define Ring ’
   [
        up     4/16
        down   4/16
        left   5/16
        right  5/16

        line up right 2/16 then right then down right 2/16
        line down left 2/16 then left then up left 2/16
        circle at 1/2 between 2nd last line and last line
        move to last line.start

        right
   ]
’

# Doppelbindung
define TwoLines ’
  line $1 from 1/3 between $2 and $3 chop
  line $1 from 2/3 between $2 and $3 chop
’

# Dicker Pfeil
define FatLine ’
   line from $1 to $2
   for i=0.01 to 0.02 step 0.005 do {
     line from $1+(i,i) to $2
     line from $1-(i,i) to $2
   }
```

```
'

define CH3 '
  "CH$_3$" $1
'

define CH2 '
  "CH$_2$" $1
'

define ATOM '
 circle invis rad 1/16 $1 '

# Programmbeginn

circlerad = 5/32

up .25          # Richtungen vorbelegen
down .25
left .25
right .25

Ring

line
CH2
line
C: ATOM("C")
line
N: ATOM("N")
TwoLines(up,C.nw,C.ne)
ATOM("O") with s at 1/2 of last line.en and 2nd last line.en
ATOM("H") with s at N.n
move from N down down right right
ATOM("N") with c
L1: line from last circle left chop
L2: line up
line right to last circle chop
L3: line right right from last circle chop
line right
ATOM("H")
move right right right from N
S: ATOM("S") with c
line down left from last circle chop
line down right from last circle chop start then down
move from S down right
line
CH3
move to last line.start
line up right
CH3 (with sw at last line.en)
FatLine (N.se , L2.en)
B1: box wid textwid ht .5*textht with.sw at L1.en invis
TwoLines(down left,B1.nw,B1.se)
```

```
ATOM("O") with ne at 1/2 of last line.end and 2nd last line.end
move right from L2.en
line dotted up
ATOM("H")
move to L3.en
line down right dotted
box wid .5 ht 3/16 invis "COOH" with nw
```

Oder wie wäre es mit einem kleinen Programmablaufplan?

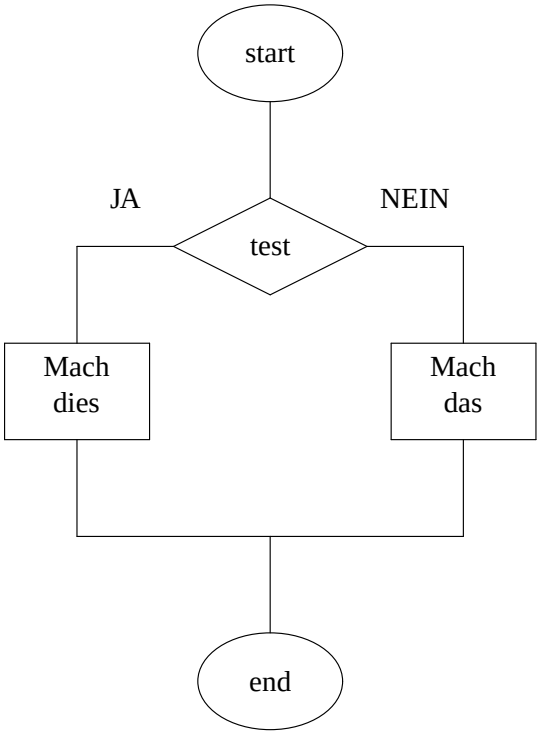


Abbildung 6.2: flow-chart.pic

```
# flow-chart.pic, Beispiel: Flussdiagramm
# Bild Nr. 31

textwid = 13/4 * textht

define Raute ’
        RRIGHT: line down linewid/2 right
                line left down linewid/2
        RLEFT:  line down linewid/2 left from 2nd last line.start
                line right down linewid/2
        box with .center \
            at 1/2 between second last line.end and 4th last line.end \
```

```
            invis '

down
ellipse "start"
line
Raute "test"

line left from RLEFT.end
box invis "JA" with s at last line
line down from last line.end
box "Mach" "dies"
line then right right

line right from RRIGHT.end
box invis "NEIN" with s at last line
line down from last line.end
box "Mach" "das"
line then left left then down
ellipse "end"
```

Hier wurde ein Makro so definiert, daß das Makroende nicht gleichzeitig einem Befehlsende entspricht, dadurch können nach dem Makronamen Argumente wie in gewöhnlichen Primaries folgen. Wie Sie sehen, sind Makros recht praktisch für verschiedene Anwendungen zu benutzen.

6.3 Die Verwendung von Parametern

Nun wird es etwas komplizierter. Sie können, wie schon angedeutet, in Makros bis zu neun Parameter mit den Namen $1 bis $9 verwenden. Wenn nun einem Makroaufruf eine offene runde Klammer folgt, wird alles bis zu der dazugehörenden geschlossenen runden Klammer als Argumentliste interpretiert. Die einzelnen Argumente innerhalb dieser Liste müssen durch ein Komma (’,’) voneinander abgegrenzt werden. Die Argumente selbst sind weitgehend beliebig. Achten Sie aber darauf, daß einer geöffneten runden Klammer auch das geschlossene Gegenstück folgt, sonst kann das Ende der Argumentliste nicht richtig erkannt werden. Alles was innerhalb der Argumentliste zwischen zwei runden Klammern steht, wird ungesehen übernommen, ein Komma wird also nicht als Argumenttrenner erkannt. Sie haben damit die Möglichkeit, einem Parameter eine Argumentliste mitsamt Klammern zuzuweisen.

Die einzelnen Argumente werden beim Makroaufruf ausgelesen und den entsprechenden Parametern der Reihe nach textuell zugewiesen. Labelnamen werden noch nicht interpretiert. Stehen weniger als neun Argumente in der Liste, wird den Parametern, die nicht belegt werden können, der leere Text zugewiesen.

Befinden sich in der Argumentliste Parameter, wird deren Wert übergeben. Sind Argumente Makrobezeichner, werden diese noch nicht ersetzt. Eine Ersetzung erfolgt erst, wenn der Makrobezeichner innerhalb eines Parameters in einem Makrotext gefunden wird. Sie können so theoretisch Makros kreieren, die je nach Argument die unterschiedlichsten Ausgaben erzeugen. Es ist auch möglich, Parameter weiterzureichen. Hier nun zwei Beispiele, welche diese Möglichkeiten ’nutzen’:

```
# parameter.pic, Makronamen als Parameter uebergeben
```

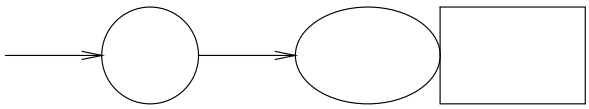


Abbildung 6.3: parameter1.pic

```
# Bild Nr. 32

define Box ’ box ’

define Circle ’ circle ’

define Prim ’ $1 ’

define Prim2 ’ ellipse ; $1 ’

define PRIM ’
  arrow
  $1 ($2)
’

PRIM (Prim,Circle) # => arrow; circle
PRIM (Prim2,Box)   # => arrow; ellipse; box
```

Abschließend noch ein Beispiel, in dem eine ganze Parameterliste übergeben wird.

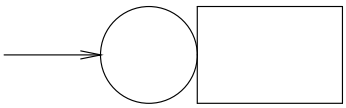


Abbildung 6.4: parameter2.pic

```
# parameter.pic, Makronamen als Parameter uebergeben
# Bild Nr. 33

define Box ’ box ’

define Circle ’ circle ’

define Prim ’ $1; $2 ’

define PRIM ’
  arrow
  $1 $2
’

PRIM (Prim,(Circle,Box)) # => arrow; circle; box;
```

6.4 Rekursionen

Innerhalb einer Makrodefinition kann ein Makroaufruf stehen, insbesondere der rekursive Aufruf des gleichen Makros. Sie sollten die Möglichkeit der Rekursion aber nur äußerst sparsam und bedacht verwenden. Da ein rekursiver Aufruf nicht gerade wenig Speicher benötigt, führen große Rekursionstiefen leicht zu einer Blockierung des Rechners, oder gar zu einem bösen Systemzusammenbruch, besonders, wenn der Swap–Bereich der Festplatte nicht so ganz in Ordnung ist. Andere Benutzer des Rechners, auf dem Sie arbeiten, könnten in diesem Fall dann unter Umständen verärgert reagieren.

Also: Bitte Vorsicht !!!

Den Rekursionsabbruch müssen Sie mit Hilfe der bedingten Anweisung realisieren. Falls ein solcher Abbruch vergessen oder falsch programmiert wurde, wird Ihr Programm zu einem speicherfressenden Ungeheuer, das sich nur noch, wenn Sie schnell genug sind, von einem anderen Terminal aus durch einen kill–Befehl stoppen läßt. Hier trotzdem ein Beispiel, das die Rekursion nutzt. Es werden Hilbert–Kurven nach einem Algorithmus aus [WIR], Seite $155 - 158$ gezeichnet:

```
# hilbert.pic, Hilbert-Kurven, aus N. Wirth,
#               Algorithmen und Datenstrukturen
#              Anwendungsbeispiel fuer Rekursionen
# Bild Nr. 34

N=4                     # H. Kurven bis Ordnung N,
                        # nicht zu gross machen
h=5                     # Groesse in Inch


define A '      # $1 = Variablenbezeichner
  if $1>0 then {
    D($1-1); line left  h
    A($1-1); line down  h
    A($1-1); line right h
    B($1-1)
  }
'

define B '      # $1 = Variablenbezeichner
  if $1>0 then {
    C($1-1); line up h
    B($1-1); line right h
    B($1-1); line down h
    A($1-1)
  }
'

define C '      # $1 = Variablenbezeichner
  if $1>0 then {
    B($1-1); line right h
    C($1-1); line up h
```

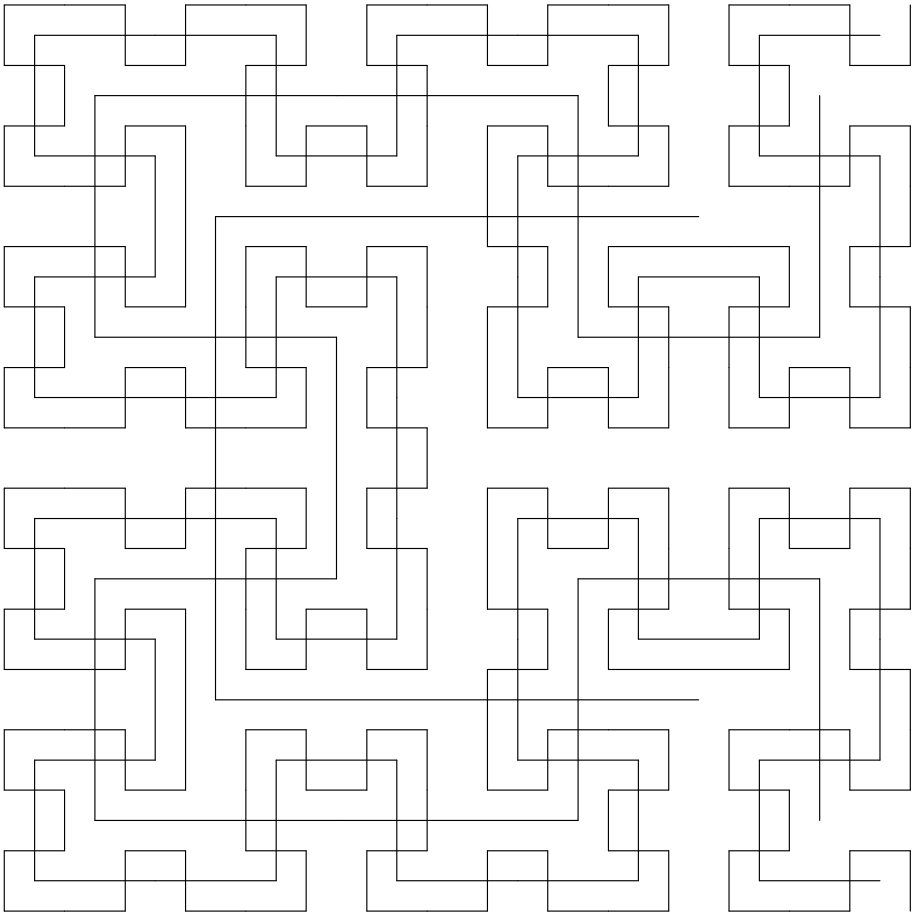


Abbildung 6.5: hilbert.pic

```
    C($1-1); line left h
    D($1-1)
  }
'

define D '      # $1 = Variablenbezeichner
  if $1>0 then {
    A($1-1); line down h
    D($1-1); line left h
    D($1-1); line up h
    C($1-1)
  }
'

# Programmbeginn
```

```
for i=1 to N do {
  # Hilbertkurve der Ordnung i berechnen
  h = h / 2
  x = x0 = x0+(h / 2)
  y = Y0 = Y0+(h / 2)
  move to (x,y)
  A(i)
}
```

7. Ergänzungen

In diesem Kapitel beschreibe ich noch die Ergänzungen, die ich PIC zugefügt habe, um, wenn auch nicht einfach, externe Schriftsätze (Fonts) benutzen zu können. Außerdem wird ein Attribut beschrieben, das es ermöglicht, Ecken von Rechtecken abzurunden und eins, mit dem geschlossene Splinezüge möglich werden.

7.1 Die Font–Anweisung

Mit ihrer Hilfe kann ein anderer Schriftsatz (Font) als der standardmäßige Pica–Font gewählt werden.
Schreibweise:

```
"font" text
```

'text' ist der Name des Fonts. also z.B. (mit Häckchen):

```
"pica"
"elite".
```

Externe Fonts (das sind die, die nicht im Drucker vorhanden sind), die im System verfügbar sind, müssen vor dem Namensanfang einen Doppelpunkt erhalten, Bsp.:

```
":bocklin.14"
":times.r.6"
```

Die Zahl am Ende des Namens gibt aufgrund einer Konvention die Zeichenhöhe in Pixel an. D.h. wenn die Zahl n ist, ist die Texthöhe n/72tel Inch. PIC berechnet diesen Wert für Sie und weist ihn der 'textht'–Variablen zu. Die durch einen Punkt abgetrennten Buchstaben, die vor dieser Zahl stehen können, haben folgende Bedeutung:

Endung	Stil
.r	Roman
.i	Italics (Schräggestellte Buchstaben)
.s	Sonderzeichen
.b	bold (Fettdruck)

Bei den externen Fonts sollte die durchschnittliche Zeichen–Breite der textwid Variablen zugewiesen werden. Die Größen von ’pica’ und ’elite’ kennt PIC und nimmt deshalb automatisch eine entsprechende Zuweisung an die Defaultvariablen vor.

Die Buchstabengrößen werden vor allem dann gebraucht, wenn der Text außerhalb eines Primaries genau plaziert werden soll, oder wenn eine Box um den Text gezeichnet werden soll. Soll der Text innerhalb eines Primaries plaziert werden und ist er Attribut dieses Primaries, gelten folgende Faustregeln für die Belegung der Defaultvariablen:

Linksbündig (’ladjust’) oder nur eine Textzeile: ’textht’ und ’textwid’ brauchen nicht bekannt sein.

Zentriert (’centered’) : ’textht’ sollte bekannt sein, ’textwid’ braucht nicht.

Rechtsbündig (radjust): ’textht’ und ’textwid’, sowie die jeweiligen Stringlängen werden benötigt.

Da die externen Fonts zumeist proportionale Fonts sind, ist es für PIC nicht (ohne sehr großen Aufwand) möglich, die genaue Breite eines Textes festzustellen. Auch kennt PIC den Inhalt des Textes nicht (ich erinnere hier an die vielen $–Befehle von SCRIPT, die PIC nicht kennen kann, die aber Einfluß auf die Textlänge nehmen). Sie müssen deshalb, wenn Sie proportionalen Text rechstbündig darstellen wollen, seine Breite ausmessen und PIC mit dieser Breite bekannt machen. Das Gleiche gilt auch für Text, dessen tatsächliche Länge nicht mit der Länge des Textstrings im PIC–Programm übereinstimmt. In den PIC–Textattributen wurde deshalb das ’len’–Attribut aufgenommen. Es hat folgende Schreibweise:

```
"len" expr
```

Der Wert von expr kann eine positive oder negative Zahl sein. Ist er negativ, ist die Textlänge um ’−expr’ Einheiten kleiner als sie Zeichen enthält. Ist sie positiv, gibt sie die aktuelle Textlänge in ’textwid’ Einheiten an.

Wenn Sie die Textlänge ausmessen wollen, setzen Sie textwid auf eine Länge in Inch und tragen für jeden Textstring mit Hilfe von ’len’ die ausgemessene Länge in textwid Einheiten in Ihr PIC–Programm ein.

Hier drei Beispiele mit verschiedenen Fonts:

```
# elite.pic, eine Verwendung des Linienchoppens
# Bild Nr. 35

box with c "Box 1"
box at (-1,-1) "Box 2"
box at (1,-1)  "Box 3"

line from 1st box to 2nd box chop
"\\sc Linie 1" left with se at last line

line from 1st box to 3rd box chop
"\\sc Linie 2" right with sw at last line
```

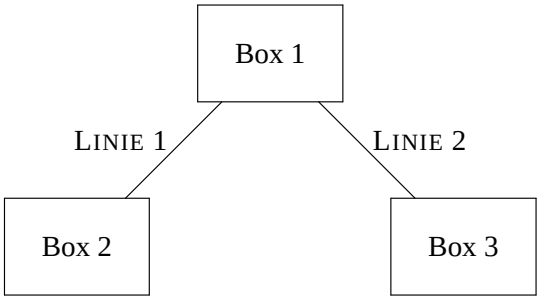


Abbildung 7.1: elite.pic

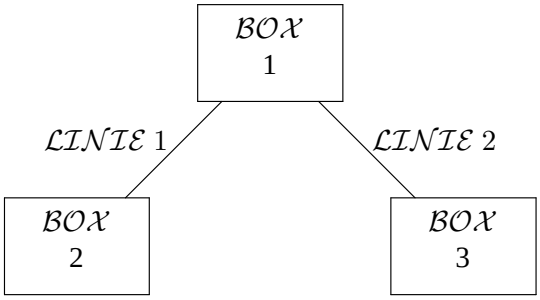


Abbildung 7.2: font.pic

```
# font.pic, eine Verwendung des Linienchoppens
# Bild Nr. 36

box with c "$\\cal BOX$" "1"
box at (-1,-1) "$\\cal BOX$" "2"
box at (1,-1)  "$\\cal BOX$" "3"

line from 1st box to 2nd box chop
"$\\cal LINIE \\rm\\ 1$" left with se at last line

line from 1st box to 3rd box chop
"$\\cal LINIE \\rm\\ 2$" right with sw at last line


# bocklin.pic, Beispiel zur Ausrichtung externer Fonts innerhalb Primaries
# Bild Nr. 37

boxwid = textht * 7

right .25

box "\\small \\hspace*{2pt}$\\cal CALIGRAPH$" \
```

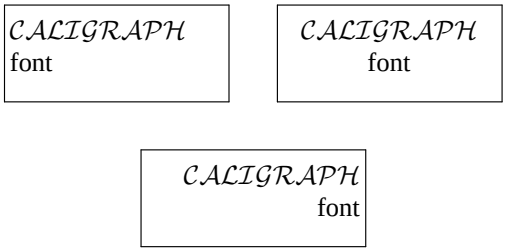


Abbildung 7.3: bocklin.pic

```
    "\\hspace*{2pt}font" ljust

move

box "\\small $\\cal CALIGRAPH$" "font"

box "\\small $\\cal CALIGRAPH$\\," "font\\," rjust\
  with t at (1/2 between 1st box.s and 2nd box.s) - (0,.25)
```

'font text' können Sie auch innerhalb eines Primaries als Primary–Attribut verwenden. Der Font gilt dann lokal für dieses Primary. Die dazugehörenden Attribute, die die Buchstabengröße angeben heißen:

tht, theight,
twd, twid, twidth

Sie sind mit den Werten der Text–Defaultvariablen 'textht' und 'textwid' vorbelegt. Diese drei obigen Attribute sind im Gegensatz zum 'len'–Attribut *keine* Text–Attribute, sondern einfache Primary–Attribute (d.h. sie müssen nicht hinter dem Textstring stehen), gelten also für den gesamten Text eines Primaries. Die verschiedenen Zeichensätze werden auf dem Bitmap–Terminal nicht dargestellt, sondern nur in Verbindung mit SCRIPT.

7.2 Zusätzliche Attribute

Die Primary–Attribute:

```
"font" expr
"theight" expr
"twid" expr
```

und das Text–Attribut:

```
"len" expr
```

sind im vorigen Kapitel beschrieben worden.

Es gibt jedoch noch zwei weitere Attribute, die in Verbindung mit SCRIPT eine Bedeutung haben. Zum einem das 'corner'–Attribut. Es dient dazu, die Ecken von Rechtecken abzurunden. Sein Wert wird dem entsprechenden Parameter im SCRIPT–Befehl $ellipse übergeben. Das 'corner'–Attribut hat folgende Schreibweise:

```
"corner" expr
```

Synonyme für 'corner' sind: 'corn', 'cor'.

Für 'expr' sollte folgendes gelten: $1 = 'expr' <= 99$. Wobei 1 einer Raute, 2 einer Ellipse, 5 einem Rechteck mit abgerundeten Ecken und 99 (Defaultwert) einem normalen Rechteck entspricht. Das 'corner'–Attribut wirkt beim 'box'–Befehl und bei Textumrandungen. Hier ein Beispiel mit den oben angegebenen Werten:

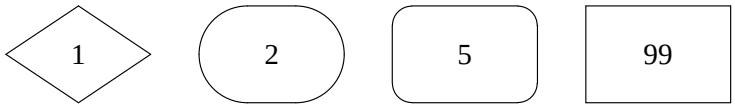


Abbildung 7.4: corn.pic

```
# corn.pic, verschiedene runde Ecken
# Bild Nr. 38

right .25
box corn 1 "1"
move
box corn 2 "2"
move
box corn 5 "5"
move
box "99"
```

Das Choppen von Linien funktioniert an den runden Ecken nicht korrekt. Auch werden bei Ausgaben auf einem Bitmap–Terminal die 'corner'–Attribute nicht beachtet.

Das andere Attribut ist das 'closed'–Attribut. Es kann dazu verwendet werden, geschlossene Splines zeichnen zu lassen. Schreiben Sie dazu einfach das Token 'closed' in die Attributliste eines Splines. Der Spline wird dann in der Ausgabe geschlossen dargestellt. Hier ein Beispiel eines geschlossenen Splines:

```
# clspline.pic, geschlossener Spline
# Bild Nr. 39

down 1   # Richtungen vorbelegen
right 2
left 2

# Spline zeichnen
```

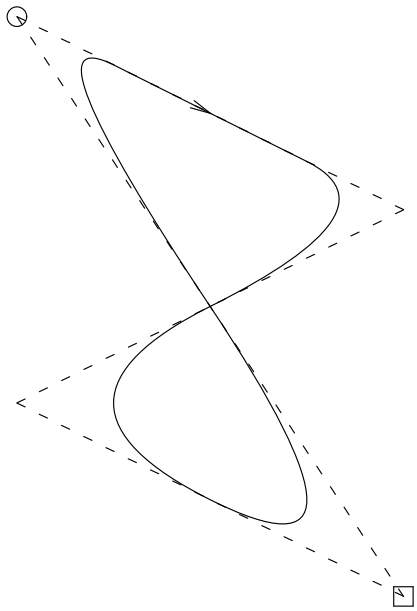


Abbildung 7.5: clspline.pic

```
spline down right then down left\
  then down right closed ->

# Line zeichnen
line from last spline.start down right \
  then down left then down right closed dashed

# Startpunkt
circle rad .05 at last spline.start

# Endpunkt
box wid .1 ht .1 at last spline.end
```

Sie haben nun alle Elemente der PIC–Syntax kennengelernt. Sicherlich ist die Benutzung von PIC in einigen Fällen nicht einfach. Es können auch nicht sämtliche Grafiken mit PIC erstellt werden. Dennoch können Sie mit einiger Erfahrung im Umgang mit PIC viele Grafiken schnell über Ihr ASCII–Terminal eingeben und mit SCRIPT in Ihre Texte einbinden. Ich wünsche Ihnen nun viel Spaß beim Arbeiten mit PIC.

8. Restriktionen

In diesem Kapitel sind die Einschränkungen aufgezählt, die bei PIC gemacht wurden:

1. Pro Primaryblock dürfen maximal je 100 verschiedene Variablen und Labels verwendet werden. Insgesamt dürfen 1024 verschiedene Identifier auftreten. Es werden jeweils die ersten 16 Zeichen unterschieden.
2. Statement–Blöcke dürfen höchstens 20 mal ineinander verschachtelt werden.
3. Die Argumente von Makroaufrufen beschränken sich auf 9.
4. Ein Argument darf maximal 100 Lexeme enthalten.
5. Leerbefehle am Anfang eines Makros werden überlesen.
6. In dem ’arc’–Primaries werden die Textplazierungsanweisungen ’left’, ’right’, ’above’ und ’under’ nicht berücksichtigt.
7. Das Primary ’arc’ kann nicht gechopped werden.
8. Rechtecke mit runden Ecken werden wie solche mit spitzen gechoppt. Auf dem Bitmap–Terminal werden keine runden Ecken ausgegeben.
9. Textformatangaben gelten jeweils für den gesamten Textteil eines Primaries.
10. Die Benutzung von verschiedenen Fonts ist schwierig. Fonts sind nur durch Ausprobieren mit SCRIPT–Kommandos, in Verbindung mit den PIC–Defaultvariablen textwid und textht, sowie mit Hilfe des Font–Komandos, umzuschalten und auszurichten (wie in den Ergänzungen beschrieben). Auf dem Bitmap–Terminal werden externe Zeichensätze nicht dargestellt.
11. Die Primaries werden nicht in der Reihenfolge ausgegeben, in der sie eingegeben werden, sondern nach Primary–Typen sortiert. Die Ausgabe erfolgt in folgender Reihenfolge: text, box, circle, ellipse, line, arc, spline, block.
12. Kreisbögen können nicht verschoben werden.
13. Die Grafik wird auf dem Bildschirm in einem etwas kleineren Maßstab ausgegeben, als auf dem Papier.

14. Die Fehlerbehandlung ist recht mager. Es ist für PIC manchmal auch nicht möglich Fehler zu erkennen, z.B. dann, wenn Sie sich beim Eingeben eines Bezeichners für eine Defaultvariable vertippen, oder innerhalb eines ’box’–Befehls statt wid 0.5, wdi 0.5 tippen. PIC würde in letzterem Fall in ’wdi’ eine Variable erkennen und sich entsprechend falsch verhalten.

15. Es dürfen keine Umlaute und ähnliche Sonderzeichen verwendet werden.

16. Bei manchen Rechnertypen werden die Zeilennummern nicht richtig ausgegeben und das Dateiende wird nicht als Befehlsende erkannt, d.h. hört ein ein Befehl mit dem Dateiende und nicht mit einem Befehlsende (Komma, Zeilenvorschub) auf, wird ein Syntax–Fehler gemeldet.

9. Beispiele zu PIC

Es folgen nun einige komplexere Anwendungsbeispiele für PIC.

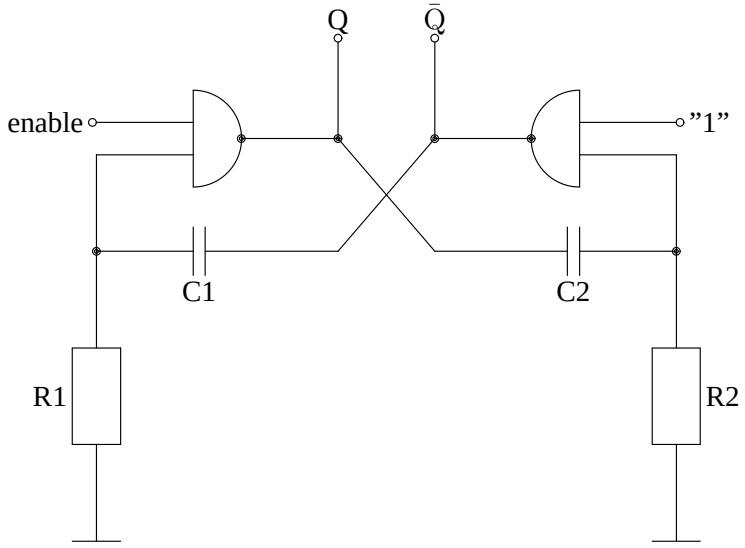


Abbildung 9.1: flipflop.pic

```
# flipflop.pic, Flipflop aus zwei NAND-Gattern darstellen
#               Zuiderveen: Handbuch der digitalen Schaltungen,
#                           Seite 63, Abb.: 5.21 b
# Bild Nr. 40

# Kleiner Kreis
define Circ ’
  circle rad .02
’

# Ausgefuellter Kreis
define Dot ’
  circle rad .02 with c
  circle rad .01 at last circle
```

```
  move to last circle
'

# Nand-Gatter, Woelbung: nach rechts, wenn $1 == ccw
define Nand '
  line down
  arc with .c at last line from last line.st to last line.en $1
  move to last arc.m
  Dot
'

# Zeichen fuer Erde
define Gnd '
  line
  move right 1/4*linewid
  line left 1/2*linewid
'

# Programmbeginn

define main '

boxwid = 1/2*boxht

Nand(cw)
line right
Dot
line up
L1: last line.start
Circ
"Q" above

line from 1/3 of 1st line.st and 1st line.en left
Circ
"enable" left
line from 2/3 of 1st line.st and 1st line.en left
line down
Dot
line
box "R1" left
Gnd
move to last circle
line right
move up 1/4*linewid
line down 1/2*linewid
move right 1/8*linewid
line up 1/2*linewid
"C1" under with t at 1/2 of last line.st and 2nd last line.en
move to last line
line to (xcoord(L1),ycoord(last line))
L3: line to (xcoord(L1)+linewid,ycoord(L1))

L4: line from L1 to (xcoord(L3.end),ycoord(L3.start))
```

```
move to L3.end
Dot
line up
Circ
"$\\rm \\bar Q$" above
move to last line.start
line right
move right 1/2*linewid
move up 1/2*linewid
Nand(ccw)
line right from 1/3 between last line.st and last line.en
Circ
"’’1’’" right
line right from 1/3 between 2nd last line.en and 2nd last line.st
line down
Dot
line
box "R2" right
Gnd
move to last circle
line left

move up 1/4*linewid
line down 1/2*linewid
move left 1/8*linewid
line up 1/2*linewid
"C2" under with t at 1/2 of last line.st and 2nd last line.en
move to last line

line to L4.end
’

main
```

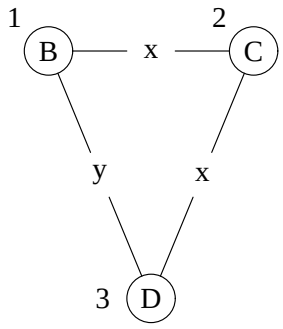


Abbildung 9.2: graph.pic

```
# graph.pic, Zeichnung eines Graphen (Quelle: Schuette, A.:
```

```
#            'Einfuehrung in Theorie und Konzepte von
#             attributierten Zeichenketten- und Graphgrammatiken',
#            Seite 94
# Bild Nr. 41

define LINE '
  line from $1 to $2 chopped
  line from $2 to $3 chopped
'

define LABEL '
  circle rad 1/8 invis'

circlerad = 1/8

circle rad .75 invis

B: circle "B" at 1st circle.nw
LABEL "1" at last circle.nw with se

C: circle "C" at 1st circle.ne
LABEL "2" at last circle.nw with se

D: circle "D" at 1st circle.s
LABEL "3" at last circle.w with e

X1: LABEL "x" at 1/2 between B and C
X2: LABEL "x" at 1/2 between C and D
X3: LABEL "y" at 1/2 between D and B

LINE (B,X1,C)
LINE (C,X2,D)
LINE (B,X3,D)



# tableau.pic, Temperaturtabelle ausgeben, Quelle: Witte Schuelerlexikon
# Bild Nr. 42

tabht  = 2   # Tabellenhoehe
tabwid = 2.4 # Tabellenbreite

H = tic_to_ht  = tabht/40
W = tic_to_wid = tabwid/12

tic_wid = 1/16

define Header '
          "\\bf Monatliche Durchschnittstemperaturen" wid 3'
define Footer '
          "" "Madras/Indien (Monsun-Klima)" \
          "Berlin (gem\"a\"sigtes Klima)" wid tabwid'

define C1 '
```

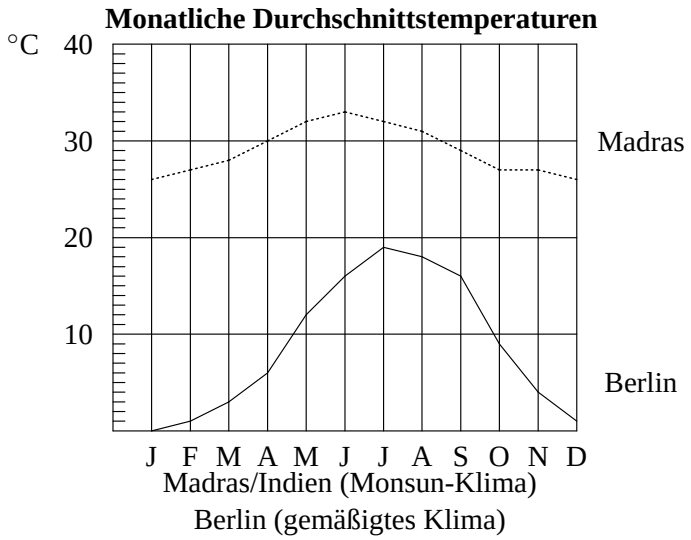


Abbildung 9.3: tableau.pic

```
        box wid 4*textht  "Berlin" invis'

define C2 '
        box wid 4 * textht "Madras" invis'


define LongLine '
        line from (0,j) right tabwid
        box ht textht wid 2*textht invis \
        $1 rjust with .e at last line.start
        '

define Tic '
        line right tic_wid from (0,j)
        '

[
  Tableau: box with .sw at (0,0) wid tabwid ht tabht

  j = tic_to_ht

  for i=1 to 39 do {

    if i==10 then {
      LongLine("10\\,")
    }
    else {
      if i==20 then {
```

```
        LongLine("20\\,")
      }
      else {
        if i==30 then {
          LongLine("30\\,")
        }
        else {
          Tic
        }
      }
    }
    j = j + tic_to_ht
  }

  box ht textht wid 2*textht "40\\," rjust at Tableau.nw with e invis
  box wid textht + .1 ht textht "$^\\circ$C" invis with e at last box.w

  right

  j = tic_to_wid

  for i=1 to 11 do {
    line up tabht from (j,0)
    j= j + tic_to_wid
  }


  move to (tic_to_wid/2, -textwid - 0.8 * textht)

  boxwid = tic_to_wid; boxht = textht
  right
  box "J" invis
  box "F" invis
  box "M" invis
  box "A" invis
  box "M" invis
  box "J" invis
  box "J" invis
  box "A" invis
  box "S" invis
  box "O" invis
  box "N" invis
  box "D" invis

  C1 at (tabwid, H*5)  with w
  C2 at (tabwid, H*30) with w

] with (0,0) at (0,0)

Header with b at 1st [].t
Footer with t at 1st [].b

move to (0,0)
```

```
# Temperaturen 1 (Berlin)
line from ( W,H* 0) \
     to ( 2*W,H* 1) to ( 3*W,H* 3) to ( 4*W,H* 6) to ( 5*W,H*12) \
     to ( 6*W,H*16) to ( 7*W,H*19) to ( 8*W,H*18) to ( 9*W,H*16) \
     to (10*W,H* 9) to (11*W,H* 4) to (12*W,H* 1)

# Temperaturen 2 (Madras)
line from ( W,H*26) \
     to ( 2*W,H*27) to ( 3*W,H*28) to ( 4*W,H*30) to ( 5*W,H*32) \
     to ( 6*W,H*33) to ( 7*W,H*32) to ( 8*W,H*31) to ( 9*W,H*29) \
     to (10*W,H*27) to (11*W,H*27) to (12*W,H*26) dotted
```

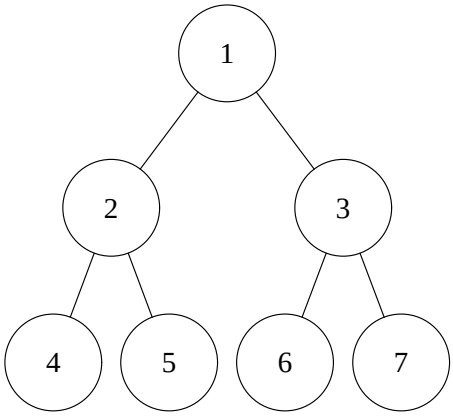
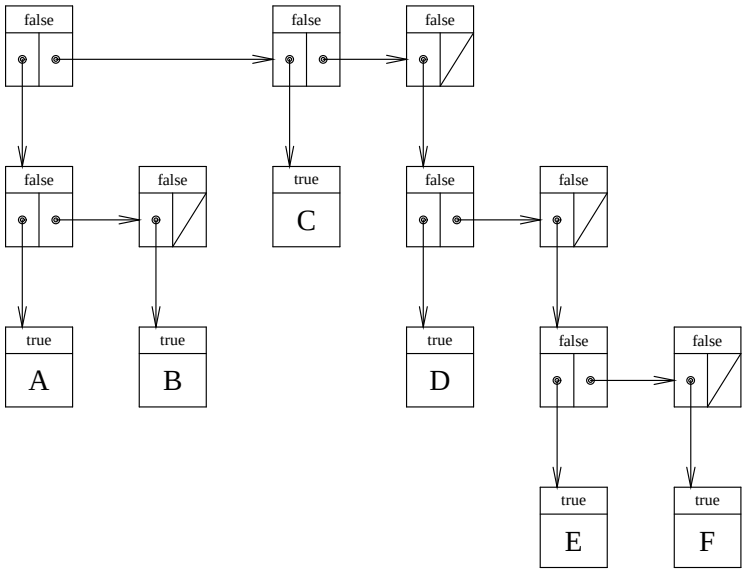


Abbildung 9.4: tree.pic

```
# tree.pic, Binaerer Baum
# Bild Nr. 43

Right = .3
Left  = -Right
Down = -.8

j = 1 # Zaehler fuer Knoten

define PRIM ’ circle ’

# Makro: Zeichnen eines Knotens mit Verbindungslinie
define Node !
  j = j+1
  circle with center at ROOT+($1,$2)   str(j,0,0)
  line from last circle to ROOT chop
!

# Makro: Zeichnen von zwei Knoten
define Trunc !
```

```
  [
    ROOT: circle invis with .c
          Node(Left -i , Down)
          Node(Right+i , Down)
    export(j)
  ]
!


circle   str(j,0,0)
i = .3
move to last circle
Trunc
move to 1st [].2nd circle
i= 0
Trunc
move to 1st [].3rd  circle
Trunc
```



((A B) C (D (E F)))

Abbildung 9.5: s-expr.pic

```
# s-expr.pic, Makros zum einfachen Erstellen von S-Expression-Graphen
#             depth first
# Bild Nr.: 44

define DOT ’
```

```
        circle .02 with c
        circle .01 at last circle
        move to last circle
'

define TAG '
        box invis ht 1/3*boxht with n at 1st box.n '

define NIL '
        line from $1.L2.sw to $1.L2.ne
'

define POINTER_R '
        move to $1.L2
        DOT
        arrow right $2 linewid + 1/4 * linewid
'

define POINTER_D '
        move to last [].L1
        DOT
        arrow down linewid + 1/3 * linewid
'

define NODE '
[
          box
   B1 :   1st box.nw + (1/4 * boxwid,0)
   B2 :   1st box.sw + (0, 1/3 * boxht)
          line right boxwid from 1/3 of 1st box.nw and 1st box.sw
          if $1 then {
            down
            TAG "\\tiny true"
            box invis ht 2/3 * boxht $2 # ATOM-Text
          }
          else {
            down
            TAG "\\tiny false"
            line 2/3 * boxht
            right
   L1 :     box invis ht 2/3 * boxht wid 1/2 * boxwid with e at last line
   L2 :     box same invis
          }
] at last arrow.end '

define DOWN '
        NODE(false) with B1
        POINTER_D
'

define RIGHT '
        NODE(false) with B2
        POINTER_D
'
```

```
# groessen setzen

boxht  = 3/7.227
boxwid = 2.5/7.227

down  boxht
right boxwid

# Beginn des eigentlichen Programms

[
  arrow 0 invis # zur Initialisierung

  N1: RIGHT
  N2: DOWN
      NODE(true,"A") with B1
      POINTER_R(N2)

  N3: RIGHT
      NODE(true,"B") with B1
      NIL(N3)
      POINTER_R(N1,3*)

  N4: RIGHT
      NODE(true,"C") with B1
      POINTER_R(N4)

  N5: RIGHT
  N6: DOWN
      NODE(true,"D") with B1
      POINTER_R(N6)

  N7: RIGHT
  N8: DOWN
      NODE(true,"E") with B1
      POINTER_R(N8)

  N9: RIGHT
      NODE(true,"F") with B1
      NIL(N9)
      NIL(N7)
      NIL(N5)
]

box invis wid 1.5 "((A B) C (D (E F)))" with t at last [].bot


# ps.pic, Phrasenstruktur eines deutschen Satzes
# Bild Nr. 45

textht = .2
textwid = 3/4 * textht
```

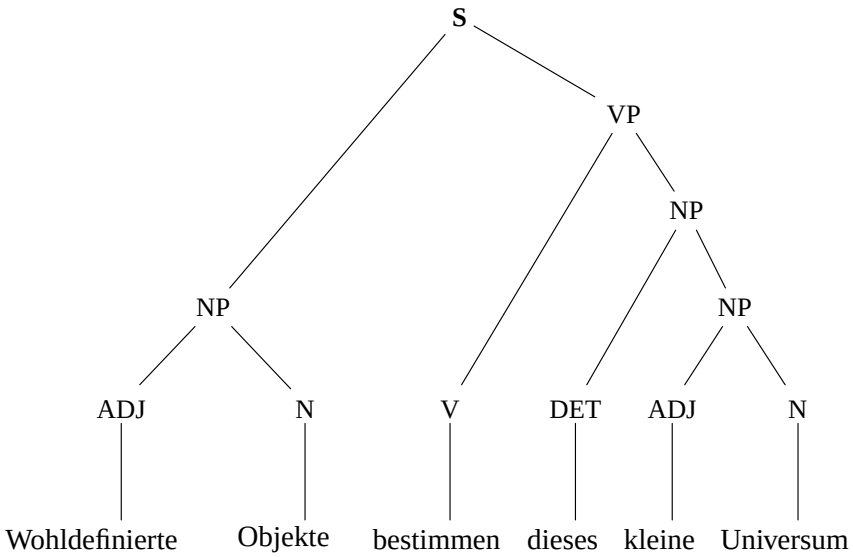
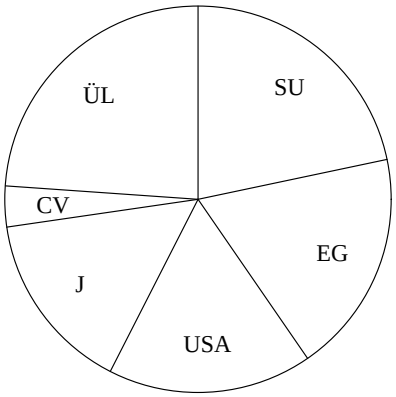


Abbildung 9.6: ps.pic

```
W1: "Wohldefinierte" len 0 wid 1.2
W2: "Objekte" len 0 wid .7
W3: "bestimmen" len 0 len 0 wid .8
W4: "dieses"  len 0 len 0 wid .5
W5: "kleine" len 0 wid .5
W6: "Universum" len 0 wid .8

line up from W1.t
W1: "\\small ADJ" above len 3
line up from W2.t
W2: "\\small N" above len 1
line up from W3.t
W3: "\\small V" above len 1
line up from W4.t
W4: "\\small DET" above len 3
line up from W5.t
W5: "\\small ADJ" above len 3
line up from W6.t
W6: "\\small N" above len 1

K1: "\\small NP" len 2 at 1/2 between W1 and W2 + (0,2*linewid)
line from W1 to K1 chop
line from W2 to K1 chop

K2: "\\small NP" len 2 at 1/2 between W5 and W6 + (0,2*linewid)
line from W5 to K2 chop
line from W6 to K2 chop
```

```
K3: "\\small NP"  len 2 at 1/2 between W4 and W6 + (0,4*linewid)
line from W4 to K3 chop
line from K2 to K3 chop

K4: "\\small VP" len 2 at 1/2 between W3 and W6 + (0,6*linewid)
line from W3 to K4 chop
line from K3 to K4 chop

K5: "\\small \\bf S" len 1 at 1/2 between W1 and W6 + (0 ,8*linewid)
line from K1 to K5 chop
line from K4 to K5 chop
```

ÜL = Übrige Länder (23,9%)
SU = Sowjetunion (21,7%)
EG = Europäische Gemeinschaft (18,7%)
USA = Vereinigte Staaten (17,1%)
J = Japan (15,2%)
CV = Chinesische Volksrepublik (3,4%)



Anteil an der Weltproduktion von Rohstahl (1977 673,1 Mill. t)

Abbildung 9.7: pie-chart.pic

```
# pie_chart.pic, Tortengrafik, Quelle: Weltalmanach, 1979
# Bild Nr. 46

define SLICE ’
        arc from P cw $2 * 3.6 at MID invis
        circle invis $1 wid .25 at 3/4 between MID and last arc.m
        P: last arc.end
        line from MID to P
’

arcrad = 1
MID : (0,0)
P:    (0,arcrad)
circle rad arcrad with c
```

```
SLICE ("\\footnotesize SU" , 21.7)
SLICE ("\\footnotesize EG" , 18.7)
SLICE ("\\footnotesize USA", 17.1)
SLICE ("\\footnotesize J" , 15.2)
SLICE ("\\footnotesize CV" , 3.4)
SLICE ("\\footnotesize \"UL" , 23.9)

# Abstand der Unterschrift von MID
line down arcrad + .25 from MID invis

"\\small \\bf Anteil an der" "Weltproduktion von" "Rohstahl" \
"(1977 673,1 Mill. t)" wid 2

# Abstand der Erklaerung von MID
line left arcrad + .25 from MID invis

"\\small \"UL  = \"Ubrige L\"ander (23,9\\%)" la wid 2.5 \
"SU  = Sowjetunion (21,7\\%)" \
"EG  = Europ\"aische Gemeinschaft (18,7\\%)" \
"USA = Vereinigte Staaten (17,1\\%)" \
"J   = Japan (15,2\\%)" \
"CV  = Chinesische Volksrepublik (3,4\\%)" \
with e
```

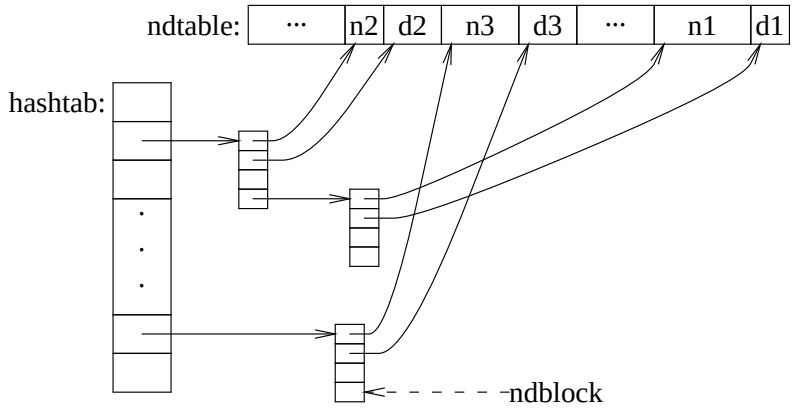


Abbildung 9.8: hash.pic

```
# hash.pic,  Hashtabelle
#            Quelle: Brian W. Kernighan,
#            'PIC - A LANGUAGE FOR TYPESETTING GRAPHICS'
# Bild Nr. 47

define ndblock X box wid boxwid/2 ht boxht/2
        down; box same with .t at bottom of last box
        box same with .t at bottom of last box; box same
```

```
X

boxht = .2; boxwid = .3
down; box; box; box; box ht 3*boxht "." "." "."
L: box; box
"hashtab: " with .e at 1st box.w
right
Start: box wid .5 with .sw at 1st box.ne + (.4,.2) "..."
N1: box wid .2 "n2"; D1: box wid .3 "d2"
N3: box wid .4 "n3"; D3: box wid .3 "d3"
box wid .4 "..."
N2: box wid .5 "n1"; D2: box wid .2 "d1"

arrow right from 2nd box
ndblock
spline -> right .2 from 4th last box to N1.sw + (0.05,0)
spline -> right .3 from 3rd last box to D1.sw + (0.05,0)
arrow right from last box
ndblock
spline -> right .2 from 4th last box\
  to N2.sw - (0.05,.2) to N2.sw + (0.05,0)
spline -> right .3 from 3rd last box\
  to D2.sw - (0.05,.2) to D2.sw + (0.05,0)
arrow right 2*linewid from L
ndblock
spline -> right .2 from 4th last box to N3.sw + (0.05,0)
spline -> right .3 from 3rd last box to D3.sw + (0.05,0)

circlerad = .3
"ndblock" with .w at last box.e + (.75,0)
arrow dashed from last text.w to last box chop

"ndtable: " with .e at Start.w


# blocknode.pic, PIC’s interne Block-Struktur
# Bild Nr. 48

textht = 6/72.27
textwid = 0

define BlockNode ’
[
box

for i=1 to 3 do {
  line right boxwid from i/4 between last box.nw and last box.sw
}

Temp = 1/4 * boxht

line down Temp from last box.nw + (Temp,0)
line down Temp from 2nd line.end - (Temp,0)
```

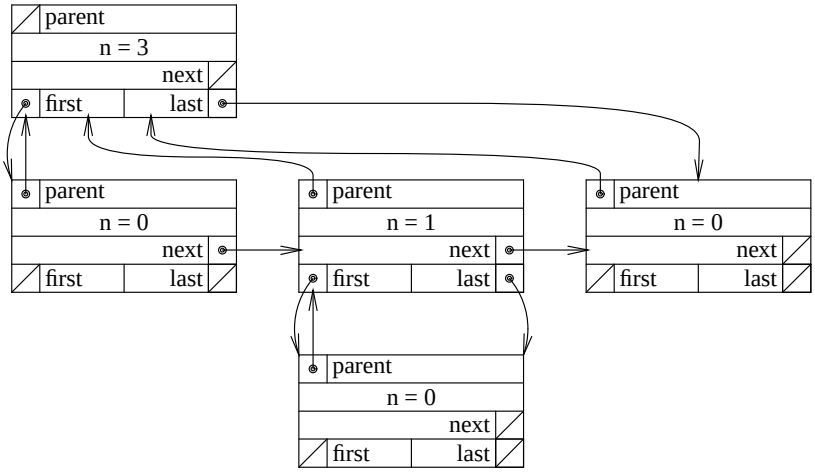


Abbildung 9.9: blocknode.pic

```
line down Temp from 3rd line.end   - (Temp,0)
line down Temp from 3rd line.start + (Temp,0)

line down Temp from 3rd line

"\\footnotesize parent" right with w at 4th line

$1 wid .5 at 1/2 between 1st line and 2nd line

"\\footnotesize next" left with e at 5th line

"\\footnotesize last" left with e at 6th line
"\\footnotesize first"  right with w at 7th line
PARENT : box invis ht Temp wid Temp with .nw at 1st box.nw
NEXT   : box invis same with .ne at 2nd line.end
FIRST  : box invis same with .sw at 1st box.sw
LAST   : box onvis same with .se at 1st box.se
] ’

define Null ’
  line from $1.sw to $1.ne
’

define Dot ’
  circle .01 at $1
  circle .02 at $1
’

boxht =  7*textht
boxwid = 2*boxht

range = 1.3*textht # So, dass das Bild auf die Seite passt
```

```
BlockNode ("\\footnotesize n = 3")
Null(last[].PARENT)
Null(last[].NEXT)
D1F: Dot(last[].FIRST)
D1L: Dot(last[].LAST)

move down 3*range from last [].s

BlockNode ("\\footnotesize n = 0") with n
D2P: Dot(last[].PARENT)
D2N: Dot(last[].NEXT)
Null(last[].FIRST)
Null(last[].LAST)

move right 3*range from last [].e

BlockNode ("\\footnotesize n = 1") with w
D3P: Dot(last[].PARENT)
D3N: Dot(last[].NEXT)
D3F: Dot(last[].FIRST)
D3L: Dot(last[].LAST)

move right 3*range from last [].e

BlockNode ("\\footnotesize n = 0") with w
D4P: Dot(last[].PARENT)
Null(last[].NEXT)
Null(last[].FIRST)
Null(last[].LAST)

move down 3*range from 2nd last [].s

BlockNode ("\\footnotesize n = 0") with n
D5P: Dot(last[].PARENT)
Null(last[].NEXT)
Null(last[].FIRST)
Null(last[].LAST)

spline from D1L right 1.5*boxwid+6*range+textht to 4th [].top ->
arc from D1F to 2nd [].nw ->

arrow up 3*range+textht from D2P
arrow right 3*range+textht from D2N

spline up range+textht from D3P then left boxwid \
  then up 2*range ->
arc from D3F to last [].nw ->
arc from last [].ne to D3L <-
arrow right 3*range+textht from D3N

spline up range+textht+.02 from D4P then left 2*boxwid \
  then up 2*range-.02 ->
```

```
arrow up 3*range+textht from D5P
```

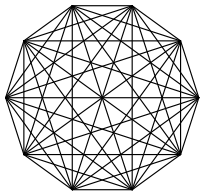


Abbildung 9.10: star.pic

```
# star.pic, Quelle:     Bentley, Jon: 'Little Languages'
#                       CACM 8/86, Seite 713
# Bild Nr. 49

N = 10; R= .5
S = 2*pi/N

for i=1 to N-1 do {
  for j= i+1 to N do {
    line from (R*cos(S*i), R*sin(S*i))\
         to   (R*cos(S*j), R*sin(S*j))
  }
}
```

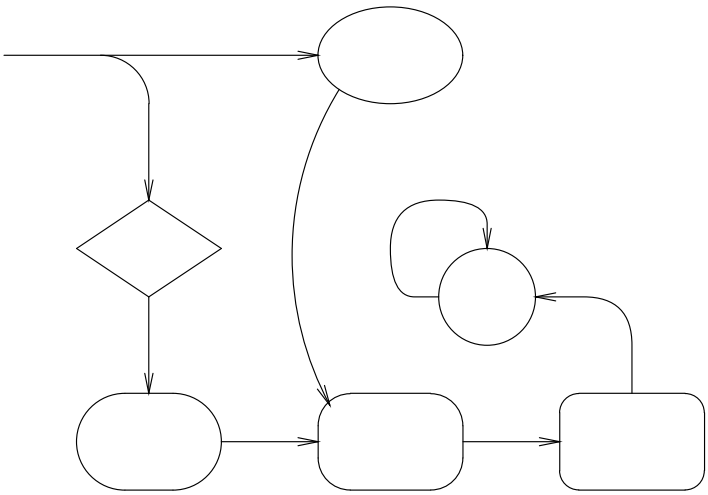


Abbildung 9.11: prims.pic

```
# prims.pic: zeigt die Primaries von PIC
# Bild Nr. 50

line
arc cw
arrow
box corner 1
arrow
box corner 2
right
arrow
B: box corner 3
arrow
box corner 5
up
spline then left ->
circle
spline .25 then up .5 then right .5 then down .25 ->
arrow right arcrad + .5 + .5*boxwid from first line.end
ellipse
arc invis from B.nw + (1/3 * boxht, 0) to B.nw - (0, 1/3 * boxht)
arc from last ellipse.sw to last arc.m ->
```

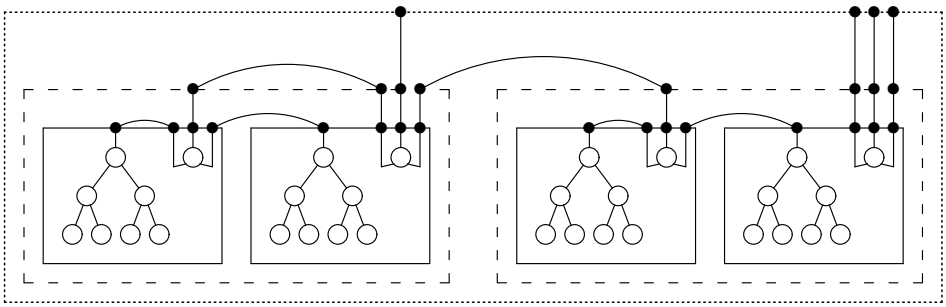


Abbildung 9.12: complex.pic

```
# Beispielbild, simuliert Prozedurmechanismus
# aus : B.W. Kernighan, PIC A LANGUAGE FOR TYPESETTING GRAPHICS
# Seite 16
#
# Im Gegensatz zum Original kann meine PIC-Implementation
# nicht (coord, coord), man muss angeben, welchen
# Koordinatenteil man haben moechte.
# Die arc cw Primaries werden anders behandelt.
#
# Bild Nr. 51
#
circlerad = 0.05
define pin | "$\\bullet$" at end of last line |
```

```
define node |circle|
define chip |
[ N1: node
  N2: node at N1 + (-.15, -.2); line from N1.c to last circle chop
  N3: node at N1 + (.15, -.2); line from N1.c to last circle chop
  N4: node at N2.c + (-.075, -.2); line from N2.c to last circle chop
  N5: node at N2.c + (.075, -.2); line from N2.c to last circle chop
  N6: node at N3 + (-.075, -.2); line from N3.c to last circle chop
  N7: node at N3 + (.075, -.2); line from N3.c to last circle chop
  N0: node at N1 + (.4,0)
  Bll: (xcoord(N4.w), ycoord(N4.s)) - (.1,.1) # Unterschied zum Original
  Bul: (xcoord(Bll), ycoord(N1.n)) + (0,.1)
  Bur: (xcoord(N0.e), ycoord(Bul)) + (.1,0)
  Blr: (xcoord(Bur), ycoord(Bll))
  line from Bll to Bul to Bur to Blr to Bll
  line from N1.n up .1; SP: pin
  line from N0.n up .1; P: pin
  line from N0.sw to N0.s - (.1,0); line up .2; L: pin
  line from N0.se to N0.s + (.1,0); line up .2; R: pin
] |


define twochips |
[ A: chip
  B: chip with .w at A.e + (.15,0)
  BLL: A.Bll - (.1,.1)
  BUL: BLL + (0,1)
  BLR: B.Blr + (.1,-.1);
  BUR: (xcoord(BLR), ycoord(BUL));
  arc from A.L to A.SP; arc from B.SP to A.R # Unterschied zum Original
  line dashed from BLL to BUL to BUR to BLR to BLL
  line from A.P to (xcoord(A.P),ycoord(BUL)); SP: pin
  line from B.L to (xcoord(B.L),ycoord(BUL)); L: pin
  line from B.R to (xcoord(B.R),ycoord(BUL)); R: pin
  line from B.P to (xcoord(B.P),ycoord(BUL)); P: pin
] |


C: twochips
D: twochips with .w at C.e + (.25,0)
OBLL: C.BLL - (.1,.1)
OBUL: OBLL + (0,1.5)
OBLR: D.BLR + (.1,-.1)
OBUR: (xcoord(OBLR), ycoord(OBUL))
arc from C.L to C.SP; arc from D.SP to C.R # Unterschied zum Original
line dotted from OBLL to OBUL to OBUR to OBLR to OBLL
line from C.P to (xcoord(C.P),ycoord(OBUL)); SP: pin
line from D.L to (xcoord(D.L),ycoord(OBUL)); L: pin
line from D.R to (xcoord(D.R),ycoord(OBUL)); R: pin
line from D.P to (xcoord(D.P),ycoord(OBUL)); P: pin


# QUILT.PIC
# Das Original stammt von R. Sethi (er brauchte nur 35 Zeilen!!!)
```

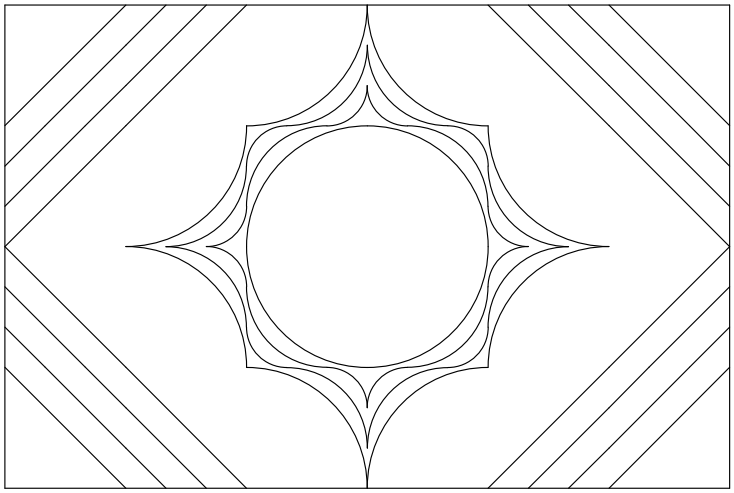


Abbildung 9.13: quilt.pic

```
# Bild Nr. 52

define Lines ’
  X = $2
  Y = $3

  for I=1 to 3 do {
    line from $1 + (X,0) to $1 + (0,Y)
    X = X + $2
    Y = Y + $3
  }
’

define Arcs ’
  X = $2
  Y = $3

  for I=1 to 3 do {
    if ($4) then {
      arc from $1 + (X,0) to $1 + (0,Y) with c at $1 + (X, Y)
    } else {
      arc from $1 + (0,Y) to $1 + (X,0) with c at $1 + (X, Y)
    }

    X = X + $2
    Y = Y + $3
  }

’

W = boxwid * 5
```

```
H = boxht  * 5

box wid W ht H

if W < H then {
  M = W*.25
} else {
  M = H*.25
}
boxwid = boxht = M
S = M/3

box invis at 1st box.sw with sw
Lines (last box.ne, -S, -S)
Lines (last box.se, S, S)
Lines (last box.nw, S, S)

box invis with nw at 1st box.nw
Lines (last box.se, -S, S)
Lines (last box.sw, S, -S)
Lines (last box.ne, S, -S)

box invis with ne at 1st box.ne
Lines (last box.sw, S, S)
Lines (last box.se, -S, -S)
Lines (last box.nw, -S, -S)

box invis with se at 1st box.se
Lines (last box.nw, S, -S)
Lines (last box.sw, -S, S)
Lines (last box.ne, -S, S)

box wid 2*M ht 2*M at 1st box invis

Arcs (last box.w, -S, S, true)
Arcs (last box.w, -S, -S, false)
Arcs (last box.nw, S, -S, true)
Arcs (last box.n, -S, S, true)
Arcs (last box.n, S, S, false)
Arcs (last box.ne, -S, -S, false)
Arcs (last box.e, S, S, false)
Arcs (last box.e, S, -S, true)
Arcs (last box.se, -S, S, true)
Arcs (last box.s, S, -S, true)
Arcs (last box.s, -S, -S, false)
Arcs (last box.sw, S, S, false)
```

10. Die verwendeteten Schlüsselworte

Hier nun eine Liste mit allen von PIC verwendeten Schlüsselworten. Sie dürfen nicht als Variablen– oder Labelnamen verwendet werden. Synonyme stehen nebeneinander.

Die PIC–Schlüsselworte

1. define
2. false
3. true
4. pi
5. arc
6. arrow
7. block, []
8. box
9. circle
10. ellipse
11. line
12. move
13. spline
14. text
15. first
16. second
17. third
18. st
19. nd
20. rd
21. th
22. last
23. north, n, top, t, up, u
24. east, e, right, r
25. south, s, bottom, bot, b, down, d
26. west, w, left, l
27. ne
28. nw
29. se
30. sw
31. start, st
32. end, en
33. both, bo
34. center, cen, c
35. middle, mid, m
36. above
37. under
38. from
39. to
40. with
41. at
42. of
43. the
44. way
45. between
46. solid
47. dashed
48. dotted
49. invis
50. closed
51. same
52. chopped, chop
53. cw
54. ccw
55. ladjust, ljust, la
56. radjust, rjust, ra
57. centered
58. len
59. twidth, twid, twd
60. theight, tht
61. width, wid, wd
62. height, ht
63. radius, rad, rd
64. corner, corn, cor
65. for
66. step

67. do
68. if
69. then
70. else
71. export
72. srand
73. font
74. str
75. and
76. div
77. mod
78. not
79. or
80. acos
81. asin
82. atan
83. atan2
84. cabs
85. ceil
86. cmtoi
87. cos
88. cosh
89. exp
90. fabs
91. floor
92. fmod
93. hypot
94. int
95. log
96. log10
97. pow
98. rand
99. sin
100. sinh
101. sinus
102. sqrt
103. tan
104. tanh
105. xcoord
106. ycoord

Weiterhin haben folgende Symbole eine Funktion, und sollten deshalb nicht als Makrobegrenzer verwendet werden:

```
# \ $ _ : " , . ;
[ ] { } ( )
= == < <= > >= <> - + * /
-> <-> <-
```

11. Die Syntax von PIC

Die PIC–Syntax ist in BNF beschrieben. Die Regeln dieser Metasprache haben folgendes Aussehen:

```
Identifier = Rule;
```

Der Identifier links vom Gleichheitszeichen ist ein sogenanntes *Nonterminal* . Wann immer dieses Nonterminal in einer BNF–Regel vorkommt, muß es durch seinen Regelteil *Rule* ersetzt werden. Diese Ersetzungen erfolgen solange, bis keine weiteren Ersetzungen mehr möglich sind (das Resultat ist bei einer korrekten Ersetzung eines korrekten Ausdruckes eine Reihe von *Terminals* s.u.). Rekursionen sind möglich und werden auch häufig benutzt.

Die Regeln bestehen aus folgenden Teilen:

1. **Terminals:** das sind Zeichenketten, die zwischen zwei Häkchen (") stehen. Diese Terminals werden ohne weitere Ersetzungen übernommen.
2. **Nonterminals:** Symbole, die durch die dazugehörigen Regeln ersetzt werden müssen.
3. **Alternativen:** Alternative Regelteile sind durch einen senkrechten Strich (|) getrennt aufgeführt. Eine dieser so gekennzeichneten Alternativen kann übernommen werden.
4. **Wiederholungen:** Alles was zwischen geschweiften Klammern ({, }) steht, kann kein, ein oder mehrmals übernommen werden.
5. **Auswahl:** Alles was in eckigen Klammern ([,]) steht, kann kein oder genau einmal übernommen werden.
6. **Klammerung:** Runde Klammern ((,)) dienen dazu, zusammengehörende Regelteile zusammenzufassen.

Eine Regel wird durch ein Semikolon abgeschlossen. Jedes Programm, das mit den nun folgenden Regeln, aus dem Nonterminal 'pic', gebildet werden kann, ist syntaktisch korrekt. Der Makroprozessor, Kommentare, Blanks etc. wurden nicht berücksichtigt. 'char' steht für ein beliebiges Zeichen, außer dem Zeilenendezeichen. Mit Ausdrücken wie:

```
"0"|...|"9"
```

sind alle Ziffern gemeint, ähnliches gilt auch für Buchstaben. 'eoln' steht für das Zeilenendezeichen.

```
pic = statements;

statements = { [label ":"] statement eostm };

label = identifier;

eostm = ";" | eoln;

identifier = letter {letter | digit};

letter = "a"|...|"z"|"A"|...|"Z"|"_";

digit = "0"|...|"9";

statement = forloop | if_then_else | assignment | directon
            | primary_stm | other_stm;

other_stm = "srand" "(" expr ")"
            | "font" fontname
            | export "(" identifier {"," identifier} ")";

fontname = text;

assignment = variable {"=" variable} "=" expr;

variable = identifier;

expr = number | const | identifier
       | "(" expr ")"
       | unop expr
       | expr binop expr
       | function;

const = "true" | "false" | "pi";

number = {digit} ["."] digit {digit} [("e"|"E") ["-"] digit {digit}];

nop = "-" | "not";

binop = "+" | "-" | "*" | "/" | "div" | "mod"
        | "or" | "and"
        | "==" | ">=" | "<>" | "<=" | ">" | "<";

function = "xcoord" "(" coord ")"
           | "ycoord" "(" coord ")"

           | cmtoi "(" expr ")"

           | ran "(" ")"
           | seed "(" expr ")"

           | "cabs" "(" coord ")"
           | "fmod" "(" coord ")"
           | "hypot" "(" coord ")"
```

```
           | "pow" "(" expr "," expr ")"

           | acos "(" expr ")"
           | asin "(" expr ")"
           | atan "(" expr ")"
           | atan2 "(" expr, expr ")"
           | ceil "(" expr ")"
           | cos "(" expr ")"
           | cosh "(" expr ")"
           | exp "(" expr ")"
           | fabs "(" expr ")"
           | floor "(" expr ")"
           | fmod "(" expr ")"
           | hypot "(" expr ")"
           | int "(" expr ")"
           | log "(" expr ")"
           | log10 "(" expr ")"
           | sin "(" expr ")"
           | sinh "(" expr ")"
           | sqrt "(" expr ")"
           | tan "(" expr ")"
           | tanh "(" expr ")";

forloop = "for" variable "=" expr "to" expr ["step" expr] "do" "{"
               statements
          "}";

if_then_else = "if" expr "then" "{"
                    statements
                "}"
                [ "else" "{"
                     statements
                  "}"
                ];

direction = dir_name [expr];

dir_name = "north" | "n" | "top" | "t" | "up" | "u"
           | "east" | "e" | "right" | "r"
           | "south" | "s" | "bottom" | "bot" | "b" | "down" | "d"
           | "west" | "w" | "left" | "l";

primary_stm = "[" statements "]" {attr}
              | primary_name {attr}
              | coord;

attr = "with" ["."] (corner | coord)
       | "at" coord
       | "from" coord
       | "to" coord
       | "then" relcoord
       | relcoord
       | text {text_attr}
```

```
       | font fontname
       | "cw" [expr] | "cww" [expr]
       | "solid" | "dashed" | "dotted" | "invis"
       | "<-" | "<->" | "->"
       | "same" | "closed" | dimension expr | expr
       | chop chop_attr;

chop = "chop" | "chopped";

corner = dir_name | "nw" | "ne" | "sw" | "se"
         | "start" | "st"
         | "end" | "en"
         | "center" | "cen" | "c"
         | "middle" | "mid" | "m";

coord = coord "+" coord | coord "-" coord
        | "(" coord ")" | "(" expr "," expr ")"
        | expr between coord "and" coord
        | primary_corner;

between = "between" | "of" ["between"] | "between" "of";

primary_corner = {(nth block | identifier) "."}
                 [nth primary_name "."] [corner];

nth_1 = digit{digit} ("st" | "nd" | "rd" | "th") | "last";

nth = nth_1 "last" | ["last"] nth_1;

block = "block" | "[]";

relcoord = {direction};

text = """ {char} """ | "str" "(" expr "," expr "," expr ")";

textattr = "ladjust" | "ljust" | "la"
           | "radjust" | "rjust" |  "ra"
           | "centered"
           | "right" | "left" | "above" | "under"
           | "len" expr;

dimension = "radius" | "rad" | "rd"
            | "twidth" | "twid" | "twd"
            | "theight" | "tht"
            | "width" | "wid" | "wd"
            | "height" | "ht"
            | "corner" | "corn" | "cor";

chop_attr = "start" | "st" | "end" | "en" | "both";
```

Makros können wie folgt definiert werden:

```
macro = "define" identifier quote pic_text quote;
```

identifier ist der Makroname, quote ein beliebiges Zeichen, das nicht in ’pic_text’ vorkommen darf. ’pic_text’ ist ansonsten beliebig. Die Makrodefinition wird durch ein abschließendes quote beendet.

```
parameter = $1 | ... | $9;
```

Diese Parameter–Platzhalter dürfen in ’pic_text’ verwendet werden und werden bei einem Makroaufruf durch die aktuellen Parameter (oder den leeren Text, falls kein korrespondierender aktueller Parameter gefunden werden kann) ersetzt.

Ein Makroaufruf sieht wie folgt aus:

```
macro_call = identifier ["(" [ arg {"," arg} ] ")"];
```

’identifier’ ist hierbei der Makroname. Der Aufruf darf bis zu 9 Argumente enthalten.

Kommentare beginnen mit einem ’#’.

Befehlszeilen können durch einen ’\’ aufgebrochen werden. Alle Zeichen, die in der Zeile hinter dem Backslash stehen, werden überlesen.

12. Der Aufruf von PIC

PIC kann von der Shell heraus mit folgenden Optionen:

```
usage: pic { -p[xydsmunpl]*
             | -b ...
             | -t[ag][12]
             | -x <n> | -y <n> | -wid <n> | -ht <n>
             | -o<name>
             | -s | -l | -h <n> | -v <n> | -r <n>
             | -lf | -d <n> |
             | <name>
             | -u | ? }

         -s : Ausgabe im SCRIPT-Format (nur auf PCS-Rechnern)

 -p[xydsmunpl] : Ausgabe im PostScript-Format
                 x : Auf der Blattbreite zentriert
                 y : auf der Blatthoehe zentriert
                 d : Pic-Dictionary ausgeben (nur beim ersten Bild einer
                     PostScript Datei ausfuehren)
                 s : Bild mit 'showpage' ausgeben
                 m : 'moveto' rechte untere Ecke zum Schluss ausfuehren
                 i : Am Anfang der PostScript-Datei wird ein 'initgraphics'
                     ausgegeben, ein absolutes Positionieren zur linken
                     unteren Blattecke ist so moeglich
                 n : Anfangs kein %! ausgeben
                 p : Fuer Publisher, es wird nicht positioniert
                 l : Landscape, Ausgabe quer
                 Beispiel -pxyds: Dictionary und Bild zentriert ausgeben
                          -pp: Ausgabe fuer Publisher, ohne Dictionary

  -t[ag][12]   : Ausgabe im LaTeX/EEPIC.STY-Format
                 a : Erzeugen eines Standalone-Dokuments
                 g : Erzeugen eines Standalone-Dokuments mit GERMAN.STY
                 1 : textht auf 11er Font Groesse setzen
                 2 : textht auf 12er Font Groesse setzen
           default : textht auf 10er Font Groesse setzen

  -b...        : Implementationsabhaengige Bildschirmausgaben

  default-Ausgabe : Amiga: Ausgabe auf dem Bildschirm (-b)
                    pcs:   Scriptausgabe (-s)
                    MIPS:  PostScript (-pdxys)
```

```
                    NeXT:  PostScript (-pdxys)
                    SUN:   PostScript (-pp)

Optionen, die bei gesetzter '-b'-Option (Amiga) von Bedeutung sind:

       -h <n> : Horizontales Ausmass des Bildschirm-Fensters
                (in 1/1000 inch angeben) n > 0, default n=8500
                bei -bp Option, hor. Aufloesung in DPI
                                (Default: hi 216, lo (-bpl) 72)
       -v <n> : Vertikales Ausmass des Bildschirm-Fensters
                (in 1/1000 inch angeben) n > 0, default n=6750
                bei -bp Option, vert. Aufloesung in DPI (def. 240)
                                (Default: hi 240, lo (-bpl) 120)

Optionen, die bei gesetzter '-s'-Option (nur auf PCS-Rechnern)
von Bedeutung sind:

           -l : keine NLQ-Voreinstellung im SCRIPT-Text
       -h <n> : Horizontale Aufloesung, fuer SCRIPT-Befehl $hres <n>
                n > 0, default n=1000
       -v <n> : Vertikale Aufloesung, fuer SCRIPT-Befehl $vres <n>
                n > 0, default n=1000
       -r <n> : Drucker-Einstellung, fuer SCRIPT-Befehl $phs <n>
                n = 1, 2, default n=2 (hohe Aufl.)

Optionen, die bei gesetzter '-p'-Option von Bedeutung sind:

       -x <n> : Linker Rand (bei -pm) in 1/1000 inch
                Default : 500
       -y <n> : unterer Rand (bei -pm) in 1/1000 inch
                Default : 500
     -wid <n> : Breite des Blattes (bei -px von Bedeutung) 1/1000 inch,
                n > 0, default n=8250 (= DIN A 4)
      -ht <n> : Hoehe des Blattes (bei -py von Bedeutung) 1/1000 inch,
                n > 0, default n=11688 (= DIN A 4)
     -o<name> : PostScript-Makros (Pic-Dictionary) auf Datei <name>
                schreiben

Sonstige Optionen:

          -lf : Linefeed ist kein Befehlsende.
       -d <n> : Maximale Rekursionstiefe bei Makroaufrufen
                1 <= n <= 20, (default: n = 20)

       <name> : Eingabe von <name>.pic, Ausgabe in <name>.ps, bzw.
              : <name>.scr, die Suffixe nicht angeben

       -u | ? : Diese Hinweise auf Standard-Ausgabe schreiben

Umlaut-Codes fuer PostScript:

321 - Ae       331 - ae
322 - Oe       332 - oe
323 - Ue       333 - ue
```

```
373 - sz       247 - Paragraph

264 - kl. Pkt. 267 - grosser Punkt
374 - ...      320 - langer Strich
```

PIC liest, falls nicht anders angegeben, aus der Standardeingabe. Die Default-Ausgabe auf dem Amiga geschieht auf dem Bildschirm (beenden über Menüfunktion 'Quit').

PostScript Ausgaben werden in '.ps' Dateien, SCRIPT Ausgaben in '.scr' und LaTeX Ausgaben in '.tex' Dateien geschrieben.

Um sich den Aufruf zu erleichtern, ist es empfehlenswert, sich Shell-Prozeduren für verschiedene Aufrufe zu schreiben.

Für Rechner mit Grafikfähigkeiten existieren z.T. Optionen um diese als Preview-Funktion zu benutzen. Z.B. können auf dem Amiga folgende Optionen verwendet werden:

```
-b[nltcpse]* : (Test-)Ausgabe auf Bildschirm, 9-Nadel-Drucker
            -b : (default) Bildschirm: HiRes, Interlace, Text
            FLAGS, die -b direkt folgen duerfen
              n  : NonInterlace
              l  : LoRes
              t  : Textplatzhalter (anstatt Text)
              c  : (clear) Kein Text
              p  : Ausgabe auf Star-NL-10 und kompatible
                   -bnl : Entwurf, -bl LoRes(2), sonst HiRes(4)
              s  : (start) Ausgabe am linken Rand (bei -bp)
              e  : (end) Ausgabe am rechten Rand (bei -bp)
                   Default-Ausgabe ist zentriert
```

13. Spezielle Ausgaben und Rechner

13.1 Die LaTeX Ausgabe von PIC

Um PIC mit LaTeX nutzen zu können, stehen zwei Möglichkeiten zur Verfügung:

1. Verwendung des EEPIC Makropakets und der tpic–specials (`-t`–Option)
2. PostScript Ausgabe und Verwendung von `dvips` von Tomas Rokicki (`-pp`–Option)

Während mit der ersten Möglichkeit auch LaTeX Formatanweisungen in den Texten verwendet werden können, ist die zweite eine reine PostScript Ausgabe. Alle bisherigen Beispiele wurden mit der `-t`–Option erstellt. Die Beispiele in dem Kapitel „PIC auf Sun Workstations und PostScript“ sind mit der `-pp`–Option ausgegeben. Dort finden sich auch weitere Hinweise zur PostScript–Ausgabe.

Möchten Sie die `-t`–Option nutzen, müssen Sie in der `\documentstyle` Option die Styles „epic“ und „eepic“ einbinden. Die erzeugten TeX–Dateien können dann mit `\input` in das Dokument eingebunden werden. Zum einfachen Bildeinlesen können z.B. die folgenden Makros verwendet werden:

```
% \hbild(Bildname (ohne .tex), Unterschrift)
\newcommand{\hbild}[2]{%                    % Bild einlesen
\begin{samepage}                            % im Text
\stepcounter{figure}
\begin{center}                              % unbedingt an dieser Stelle
\input{#1}                                  % daf"ur ohne Nummer
\vspace{\medskipamount}
{{\bf\figurename{} \thefigure:} #2}\\[\medskipamount]
\end{center}
\end{samepage}
}

% \bild(Bildname (ohne .tex), Unterschrift, Label)
\newcommand{\bild}[3]{%                     % Bild einlesen (floating)
\begin{figure}[htb]
\begin{center}                              % unbedingt an dieser Stelle
\input{#1}
\caption{\label{#3} #2}
```

```
\end{center}
\end{figure}
}
```

Falls Sie die `-pp`–Option verwenden möchten, sollten Sie den „epsf“–Style verwenden. Sie können die Bilder dann mit dem `\epsfbox`–Befehl einlesen. Möchten Sie die Bilder in einer `center`–Umgebung horizontal zentriert ausgeben, muß dem `\epsfbox`–Befehl ein `leavevmode` vorangestellt werden. Sie können sich zum Einlesen die folgenden Makros definieren:

```
% \hbildps(Bildname (ohne .ps), Unterschrift)
\newcommand{\hbildps}[2]{
\begin{samepage}              % im Text
\stepcounter{figure}
\begin{center}
\leavevmode\epsfbox{#1.ps}
\vspace{\medskipamount}
{{\bf\figurename{} \thefigure:} #2}\\[\medskipamount]
\end{center}
\end{samepage}
}

% \bildps(Bildname (ohne .ps), Unterschrift, Label)
\newcommand{\bildps}[3]{      % Bild einlesen (floating)
\begin{figure}[htb]
\begin{center}                % unbedingt an dieser Stelle
\leavevmode\epsfbox{#1.ps}    % Bild zentriert einfuegen
\caption{\label{#3} #2}       % Bildunterschrift, Referenzlable
\end{center}
\end{figure}
}
```

Nachdem Sie eine DVI–Datei erzeugt haben, können Sie diese wie üblich mit `dvips` in eine PostScript–Datei umwandeln. Dazu ist es nötig, vorher den PostScript–Prologue (Header) von PIC einmal auszugeben (z.B. als Datei „pic.pro“). Erreicht wird das mit dem Befehl:

```
pic -o pic.pro
```

Der Header muß dann von `dvips` eingebunden werden:

```
dvips -h pic.pro name.dvi

Das Einbinden kann auch direkt hinter der \verb+\documentstyle+
Anweisung in der \LaTeX\ Datei durch einen entsprechendes:

\begin{verbatim}
\special{header=Prolgue-Dateiname}
```

geschehen.

13.2 PIC auf dem Amiga

Auf dem Amiga wurde eine (sehr) rudimentäre Bildschirm– und Matrixdruckerausgabe (mit 8 Nadeln) zum Preview implementiert (siehe Aufrufoptionen). Da nur die Amiga-Funktionen für die Grafikausgabe verwendet werden, können keine besonders großen Bitmaps erzeugt werden. Das Programm kann im Grafikmodus über das Menü (Quit) oder die Tastenkombination „rechte Amigataste–Q“ verlassen werden.

Auf dem Amiga war es (vor allem aus Geschwindigkeitsgründen) nötig, alle Fontnamen, die von PIC verwendet werden können, in einer Datei zusammenzufassen. Die Datei muß `.picfonts` heißen und im `FONTS:` Verzeichnis stehen. In jeder Zeile dieser Datei steht die Pixelgröße gefolgt von dem Namen des Fonts, getrennt durch Leerzeichen. Die Zeilen müssen aufsteigend nach Pixelgröße sortiert sein. Da die Erstellung einer solchen Datei unzumutbar ist, liegt bei der Amiga–Version das Programm `testfont` bei. Das Programm bietet zusätzlich noch eine Previewfunktion für Fonts. Mit `testfont ?` und `testfont -u` können sämtliche Optionen ausgegeben werden. Durch

```
testfont >FONTS:.picfonts -p
```

kann eine entsprechende Fontinhalts–Datei erzeugt werden. Die Datei kann anschließend von Hand mit einem normalen ASCII–Editor nachbearbeitet werden. Z.B. können nicht gewünschte Fonts hinausgeworfen werden. In der jeweils ersten Zeile einer bestimmten Pixelgröße steht immer der Defaultfont, d.h. der Font, der genommen wird wenn kein Font in der gewünschten Pixelgröße gefunden wird. PIC durchsucht die Fonts nach absteigender Pixelgröße bis der passende Font oder ein Defaultfont in entsprechender Größe gefunden wird. Ab AmigaDos2.0 können auch skalierbare Fonts verwendet werden. Dazu muß in die ersten Zeilen jeweils `1 fontname` geschrieben werden. Der erste so eingetragene Font gilt dann immer als Defaultfont. Z.B. (3 skalierbare, 2 nichtskalierbare Fonts, Defaultfont ist CGTriumvirate):

```
1 CGTriumvirate
1 CGTimes
1 LetterGothic
5 Little
8 Topaz
```

Da die Grafikausgabe nur zum Preview dient, wurde mehr Wert auf korrekte Fontgröße als korrekten Fontnamen gelegt. PIC kann auf dem Amiga für die Erstellung von PostScript–Dateien (Ausgabe mit dem bekannten FD–Programm `Post` von Adrian Aylward) oder zur Erstellung von TEX–Dateien (z.B. für die PasTEX Version von Georg Heßmann) verwendet werden.

13.3 PIC auf dem NeXT

Auf dem NeXT wurde keine Bildschirmausgabe integriert. Mit einem Shell–Script kann aber für PostScript sehr einfach eine Bildschirmausgabe erzeugt werden:

```
if test -r $1.pic
  then
  pic $1
  if test -r $1.ps
  then
    open $1.ps
  else
    echo "$0: $1.ps is unreadable"
  fi
else
  echo "$0: $1.pic is unreadable"
fi
```

Dieses Script kann als „picp“ ausführbar gespeichert und anschließend mit „picp name“ aufgerufen werden. Der Befehl „open“ öffnet das Standard–Tool zu einer Datei. Bei „.ps“–Dateien (PostScript) ist das normalerweise „Preview“. Auch ein LaTeX Preview (z.B. als Shell–Skript „pict“) ist leicht möglich:

```
if test -r $1.pic
then
  pic -tag1 $1
  if test -r $1.tex
  then
    latex $1
    if test -r $1.dvi
      then
      dvips -o $1.ps $1.dvi
      rm $1.log
      rm $1.aux
      rm $1.tex
      rm $1.dvi
      if test -r $1.ps
      then
        open $1.ps
      fi
    fi
  fi
else
  echo "$0: $1.pic is unreadable"
fi
```

13.4 PIC auf Sun Workstations und PostScript

13.4.1 Die PostScript/Publisher-Ausgabe von PIC

PIC-Programme können als PostScript-Dateien ausgegeben und ausgedruckt werden. Diese PostScript-Dateien können auch von dem Publisher 'The Publisher^{TM}' auf Sun Workstations verwendet werden. Außerdem wurde die Bitmap-Ausgabe (die PIC '-b'-Option) an die SUNs angepasst. In diesem Kapitel werden die Erweiterungen beschrieben, die PIC für diese Ausgabe-Optionen zugefügt wurden.

Aufrufoptionen für die PostScript-Ausgabe

Durch die '-p'-Option ist es möglich, einen PIC-Text in das entsprechende PostScript-Format umzuwandeln. Falls Sie eine PIC-Grafik direkt auf einem PostScript-Drucker ausgeben möchten, können Sie das durch folgenden Aufruf erreichen:

```
pic -pdisxy name.pic
```

Die Reihenfolge der Buchstaben hinter dem '`-p`' spielt dabei keine Rolle. An der Stelle von '`name.pic`' muß der Name der PIC-Datei (das '`.pic`'-Suffix darf weggelassen werden) stehen. PIC liest dann automatisch aus '`name.pic`' und schreibt auf '`name.ps`'. Soll mittels der '`-s`'-Option eine SCRIPT-Ausgabe erstellt werden, dann würde diese auf die Datei '`name.scr`' geschrieben. Wird kein Dateiname angegeben, liest PIC aus der Standard-Eingabe und schreibt auf die Standard-Ausgabe. Nachdem die Datei übersetzt wurde, können Sie die Grafik auf Sun's normalerweise mit:

```
lpr name.ps
```

ausdrucken lassen. In der folgenden Tabelle sind die Schalter aufgeführt, die mit der '`-p`'-Option gesetzt werden dürfen. Falls Sie an einer SUN arbeiten, ist die '`-pp`'-Option (PostScript-Ausgabe für den Publisher) die Default-Option. In der zweiten Tabelle sind die Optionen aufgeführt, die Auswirkungen auf die PostScript-Ausgabe haben. Alle Optionen können mit dem Aufruf:

```
pic -u
```

auf die Standard-Fehlerausgabe geschrieben werden.
Die Schalter der '`-p`'-Option:

-d: Zusätzlich zu der erzeugten Grafik werden am Anfang der Ausgabe-Datei einige Makros in Form eines PostScript-Dictionaries ausgegeben. Alle PIC-Grafiken benötigen diese Makros. In einigen Fällen (z.B. für die Publisher-Ausgabe), muß dieses Dictionary gesondert ausgegeben werden (siehe `-o`–Option). Der PostScript-Name des Dictionaries lautet: PIC-dict. Ist es nicht möglich, das Dictonary gesondert in ein Dokument einzubinden, kann versucht werden, das Dictionary zusammen mit der ersten eingebundenen PostScript-Datei auszugegeben.

-x: Die Grafik horizontal zentrieren.

-y: Die Grafik vertikal zentrieren.

-s: Am Ende der Grafik ein ’showpage’ ausgeben.

-m: Am Ende ein ’move’ in die rechte untere Ecke ausführen.

-i: Anfangs den PostScript-Befehl ’initgraphics’ ausgeben. Es kann danach absolut zur linken unteren Blattecke positioniert werden.

-n: Anfangs kein ’%!’ ausgeben. Die Ausgabe wird in diesem Fall nur als ASCII-Text gedruckt, falls der PostScript-Drucker dieses unterstützt.

-p: Encapsulated PostScript-Datei (EPSF) für den Publisher erstellen. Die x, y, s, i und m Schalter haben in diesem Fall keine Bedeutung.

-l: Die Grafik wird quer (Landscape) ausgegeben.

Weitere Optionen für die PostScript-Ausgabe:

-x n Die Größe des linken Randes. n ist eine Integerzahl und wird in 1/1000inch angegeben. Der Defaultwert beträgt 500. Diese Option hat bei gesetztem ’p’-Schalter in der ’-p’-Option keine Bedeutung.

-y n Die Größe des unteren Randes. n ist eine Integerzahl und wird in 1/1000inch angegeben. Der Defaultwert beträgt 500. Diese Option hat bei gesetztem ’p’-Schalter in der ’-p’-Option keine Bedeutung.

-wid n Die Blattbreite in 1/1000inch. DIN A4 ist voreingestellt: 8250. Diese Option hat bei gesetztem ’p’-Schalter in der ’-p’-Option keine Bedeutung.

-ht n Die Blatthöhe in 1/1000inch. DIN A4 ist voreingestellt: 11688. Diese Option hat bei gesetztem ’p’-Schalter in der ’-p’-Option keine Bedeutung.

-o name Das Pic-Dictionary (die globalen PostScript-Makros) werden auf die Datei ’name’ (z.B. ’pic.pro’) geschrieben. Das Directory wird von allen EPSF–Dateien, die mit PIC erzeugt wurden benötigt.

Bitte vergessen Sie nicht, daß die ’-p’–Default-Optionen nach der Angabe einer ’p’-Option ungültig werden. Falls Sie also eine Landscape-Ausgabe für den Publisher benötigen, müssen Sie ’-ppl’ als Option angeben.

13.4.2 Spracherweiterungen für PIC

Um den Fähigkeiten von PostScript gerechter zu werden, wurde die PIC-Sprache in einigen Punkten erweitert. Die Befehle ’arc closed’ (Tortenstück) und das chead-Attribut (geschlossene Pfeilspitze) wirken auch in der SCRIPT-Ausgabe. Außerdem werden die Primaries nicht mehr sortiert nach Blöcken und Typen, sondern in der historischen Reihenfolge ihrer Definition ausgegeben.

Das fill-Attribut

Die wohl wichtigste Erweiterung ist das fill-Attribut, mit dem es möglich wird, Primaries mit einem Grauton zu füllen. Das Füllen wird in der Attributliste des zu füllenden Primaries durch das Schlüsselwort ’fill’ oder ’filled’ eingeleitet. Nach diesem Schlüsselwort muß eine Zahl aus dem Intervall [0 , 1] folgen, die den Grauton angibt. Schwarz wird als eine ’0’ und weiß als eine ’1’ angegeben. Auf dem PostScript-Drucker, der hier zur Verfügung steht, sind 32 Grautöne möglich. Die Anzahl der Graustufen variiert mit der Auflösung des Geräts.

Grundsätzlich können alle Primaries gefüllt werden. Die Linien-Primaries (line, arrow und move), die Kreisbögen und die Splines werden dazu geschlossen. Wird zusätzlich zum fill-Attribut noch das invis-Attribut angegeben, erscheinen in der Ausgabe die Umrandungen nicht. Hier als Beispiel eine Anzahl nicht umrandeter Kreisbögen, die in unterschiedlichen Graustufen gefüllt wurden:

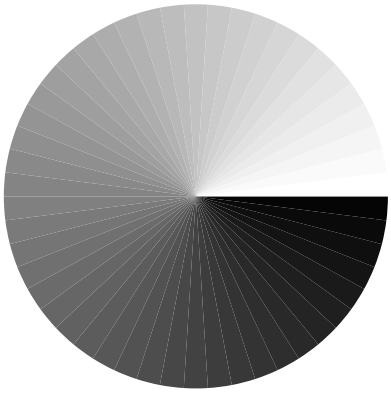


Abbildung 13.1: arcs.pic: Gefüllte Kreisbögen

```
# arcs.pic

arcrad = 1

fillc = 0
max = 360
ssize = max / 50

circle rad arcrad invis with center
arc with center at (0,0) from last circle.s to last circle.east invis

for i=ssize to max step ssize do {
  fillc = i/max
  arc with center at (0,0) from last arc.end cw ssize invis fill fillc
}
```

arc closed

Das closed-Attribut (’close’, ’closed’) bewirkt das Schließen eines Primaries. Ein Kreisbogen wird zu einem ’Tortenstück’ geschlossen. Es wird eine Linie vom Anfang, über das Zentrum zum Endpunkt des Bogens gezogen. Beim Füllen wird ein Kreisbogen automatisch geschlossen.

Pfeilspitzen

Zusätzlich zu den schon vorhandenen Pfeilspitzen-Attributen (’$<-$’, ’$<->$’, ’$->$’) können die Attribute ’ch’ (’chead’) und fh (’fhead’) angegeben werden. Sie bewirken ein Schließen, bzw. Füllen, der Pfeilspitze(n). Das folgende kleine Beispiel (mit etwas größeren Pfeilspitzen) soll dies verdeutlichen:

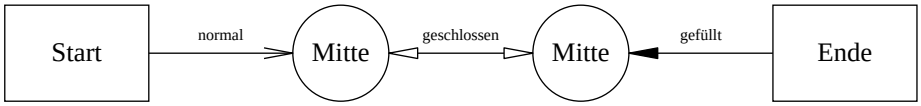


Abbildung 13.2: chead.pic: Pfeilspitzen

```
# chead.pic

font ":helvetica.g.6"

linewid = linewid * 1.5

headht = 4 * lineht/72
headwid = 1/5 * linewid
box font ":palatino.r.10" "Start"
arrow "normal" "" ""
circle font "Helvetica-Narrow.10" "Mitte"
arrow <-> ch "geschlossen" "" ""
circle font "Helvetica-Narrow.10" "Mitte"
arrow <- fh "gef\\333llt" "" ""
box font ":palatino.r.10" "Ende"
```

Abgerundete Rechtecke

Die Ecken der Rechtecke können durch das corner-Attribut (’corner’, ’corn’) abgerundet werden. ’corner 1’ bewirkt die Ausgabe eines Rhombuses. Liegt der Wert im Intervall $[2\,,\,98]$, werden die Ecken des Rechtecks durch einen Kreisbogen abgerundet, dessen Radius dem Wert von: kürzeste Seite/corner entspricht. Text- und Blockumrandungen werden von dem corner-Attribut in gleicher Weise beeinflußt. Der ’corner’-Wert 99 ist voreingestellt. Bei dem Wert 99 wird ein normales Rechteck ausgegeben.

```
# corners.pic
```

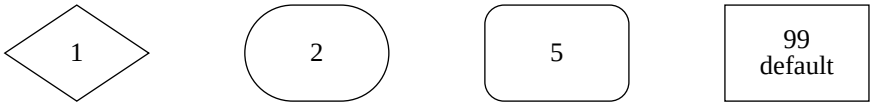


Abbildung 13.3: corners.pic: Abgerundete Ecken

```
box corner 1 "1"
move
box corner 2 "2"
move
box corner 5 "5"
move
box "99" "default"
```

Texte und Fonts

Texte werden korrekt plaziert und justiert. Es können jedoch keine Primaries relativ zu Texten ausgegeben werden. Der Fontname kann in der alten Schreibweise, wie es für die SCRIPT-Ausgabe notwendig war, angegeben werden, z.B.: '`font ":times.r.10"`'. Der führende Doppelpunkt, der einen Bitmap-Font kennzeichnet, kann weggelassen werden. Er ist nur für SCRIPT von Bedeutung. Die Attribute (im Beispiel: '.r') und die Höhe der Buchstaben in Punkten (Standard-Wert ist 10) können wahlweise weggelassen oder vertauscht werden. Wird der Fontname weggelassen (beachten Sie bitte, daß die Attributliste mit einem Punkt beginnen muß) wird der zuletzt definierte Fontname verwendet (auch Fontnamen sind blocklokal!). In der folgenden Tabelle sind die Attribute aufgeführt. Es sind jedoch nicht immer alle Attribute für jeden Font möglich. Die Reihenfolge, in der die Attribute angegeben werden können, ist egal.

Font-Attribute:

r: Roman

b: Bold

i: Italics

o: Oblique

g: German (mit Umlauten)

PIC setzt den Fontnamen, soweit möglich, in die PostScript-Schreibweise um (z.B. wird der erste Buchstabe in ein großes Zeichen umgewandelt, pica in Times-Roman etc.). Beachten Sie aber bitte die Groß-/Kleinschrift der folgenden Buchstaben. Sie dürfen auch den vollständigen PostScript-Namen z.B. '`font "Palatino-BoldItalic.g.10"` angeben. Das '.g'-Attribut (Umschaltung auf den deutschen Zeichensatz) ist kein PostScript-Attribut und darf deshalb, falls eine Umsetzung gewünscht wird, nur in der Attributliste stehen. Die Angabe der Zeichenhöhe ist wieder optional. In der folgenden Tabelle sind die auf unserem Drucker (Queme QMS-PS 800+) vorhandenen Zeichensätze aufgeführt.

Times-Roman	Helvetica	Courier
Times-Bold	**Helvetica-Bold**	Courier-Bold
Times-Italic	Helvetica-Oblique	Courier-Oblique
Times-BoldItalic	Helvetica-BoldOblique	Courier-BoldOblique
AvantGarde-Book	Palatino-Roman	NewCenturySchlbk-Roman
AvantGarde-Demi	Palatino-Bold	NewCenturySchlbk-Bold
AvantGarde-BookOblique	Palatino-Italic	NewCenturySchlbk-Italic
AvantGarde-DemiOblique	Palatino-BoldItalic	NewCenturySchlbk-BoldItalic
Helvetica-Narrow	ZapfChancery-MediumItalic	Bookman-Demi
Helvetica-Narrow-Bold	ZapfDingbats	Bookman-DemiItalic
Helvetica-Narrow-Oblique	Symbol	Bookman-Light
Helvetica-Narrow-BoldOblique		Bookman-LightItalic

Abbildung 13.4: fonts.pic: Vorhandene Fonts

```
# fonts.pic
define InBox |
  N = $1
  H = boxht-6/72
  line invis from last box.t - (0, 3/72) to last box.b + (0, 3/72)

  i=1
  if (i<=N) then {
    font $2
    $3 at i/N between last line.start and last line.end with b
    i = i+1
  }
  if (i<=N) then {
    font $4
    $5 at i/N between last line.start and last line.end with b
    i=i+1
  }
  if (i<=N) then {
    font $6
    $7 at i/N between last line.start and last line.end with b
    i=i+1
  }
  if (i<=N) then {
    font $8
    $8 at i/N between last line.start and last line.end with b
  }
  move to last box.e
  right
```

```
|

boxwid = 2.2; boxht = 1

box
InBox (4, \
  "Times-Roman", "Times-Roman",   \
  "Times-Bold", "Times-Bold", \
  "Times-Italic", "Times-Italic", \
  "Times-BoldItalic", "Times-BoldItalic")

box
InBox (4, \
  "Helvetica", "Helvetica",   \
  "Helvetica-Bold", "Helvetica-Bold", \
  "Helvetica-Oblique", "Helvetica-Oblique", \
  "Helvetica-BoldOblique", "Helvetica-BoldOblique")

box
InBox (4, \
  "Courier", "Courier",   \
  "Courier-Bold", "Courier-Bold", \
  "Courier-Oblique", "Courier-Oblique", \
  "Courier-BoldOblique", "Courier-BoldOblique")

move to 3rd last box.s
down
box
InBox (4, \
  "AvantGarde-Book", "AvantGarde-Book",   \
  "AvantGarde-Demi", "AvantGarde-Demi", \
  "AvantGarde-BookOblique", "AvantGarde-BookOblique", \
  "AvantGarde-DemiOblique", "AvantGarde-DemiOblique")
box
InBox (4, \
  "Palatino-Roman", "Palatino-Roman",   \
  "Palatino-Bold", "Palatino-Bold", \
  "Palatino-Italic", "Palatino-Italic", \
  "Palatino-BoldItalic", "Palatino-BoldItalic")
box
InBox (4, \
  "NewCenturySchlbk-Roman", "NewCenturySchlbk-Roman",   \
  "NewCenturySchlbk-Bold", "NewCenturySchlbk-Bold", \
  "NewCenturySchlbk-Italic", "NewCenturySchlbk-Italic", \
  "NewCenturySchlbk-BoldItalic", "NewCenturySchlbk-BoldItalic")

move to 3rd last box.s
down
box
InBox (4, \
  "Helvetica-Narrow", "Helvetica-Narrow",   \
  "Helvetica-Narrow-Bold", "Helvetica-Narrow-Bold", \
  "Helvetica-Narrow-Oblique", "Helvetica-Narrow-Oblique", \
  "Helvetica-Narrow-BoldOblique", "Helvetica-Narrow-BoldOblique")
```

```
box
InBox (3, \
  "ZapfChancery-MediumItalic", "ZapfChancery-MediumItalic", \
  "Times-Roman", "ZapfDingbats", \
  "Times-Roman", "Symbol")
box
InBox (4, \
  "Bookman-Demi", "Bookman-Demi",   \
  "Bookman-DemiItalic", "Bookman-DemiItalic", \
  "Bookman-Light", "Bookman-Light", \
  "Bookman-LightItalic", "Bookman-LightItalic")
```

Times, Helvetica und Palatino sind eingetragene Warenzeichen der Allied Corporation

ITC Avant Garde, ITC Zapf Chancery, ITC Zapf Dingbats und ITC Bookman sind eingetragene Warenzeichen der International Typeface Corporation.

Von den Fontnamen, die ein Blank enthalten, ist nur der letzte Namen anzugeben. Ist ein Font nicht vorhanden, wird normalerweise der 'Times-Roman'-Font (ist implementationsabhängig) verwendet. Dem 'pica'-Font entspricht 'Times-Roman' und dem 'elite'-Font entspricht 'Helvetica-Narrow'. Wenn ein '.g'-Font[1] verwendet wird, stehen zusätzlich folgende Umlaute zur Verfügung:

321	Ä	331	ä
322	Ö	332	ö
323	Ü	333	ü

Abbildung 13.5: dieresis.pic: Umlaute

```
# dieresis.pic

font "times.rg.14"

lineht = 1
boxwid = boxht


box "321" ; box "\\321" ; move .03; box "331" ; box "\\331"
box invis with .nw at 4th last box.sw
move to last box.w
```

[1] Die interne Darstellung in der PostScript-Datei lautet Fontname-Germ.

```
box "322" ; box "\\322" ; move .03; box "332" ; box "\\332"
box invis with .nw at 4th last box.sw
move to last box.w

box "323" ; box "\\323" ; move .03; box "333" ; box "\\333"
```

Um die Umlaute zu erreichen, muß die oktale Darstellung des Charactercodes, wie in PostScript üblich, in den String geschrieben werden. Da der Gegenschrägstrich, der den Oktalcode kennzeichnet, PIC als Escape-Zeichen dient, muß er im String doppelt geschrieben werden, Beispiel: `\\333`. Da auch andere wichtige Zeichen nicht über die Tastatur zu erreichen sind (eine vollständige Auflistung der Codes ist im PostScript Handbuch zu finden) folgen in der nächsten Tabelle noch einige Zeichen:

247	§	274	…
264	·	320	—
267	•	373	ß

Abbildung 13.6: char.pic: Weitere Zeichen

```
# char.pic

font "times.rg.14"

lineht = 1
boxwid = boxht

box "247" ; box "\\247" ; move .03; box "274" ; box "\\274"
box invis with .nw at 4th last box.sw
move to last box.w

box "264" ; box "\\264" ; move .03; box "320" ; box "\\320"
box invis with .nw at 4th last box.sw
move to last box.w

box "267" ; box "\\267" ; move .03; box "373" ; box "\\373"
```

Falls Strings mit einer schließenden Klammer beginnen, wird der folgende Text als PostScript-Programmteil übernommen. Der Ursprung (0, 0) befindet sich dann an der entsprechenden Bildposition, bzw. in der Mitte des Primaries, zu dem der Text angegeben wurde. Falls Sie einen Text mit einer geschlossenen Klammer beginnen möchten, müssen Sie ansatt einer Klammer den entsprechenden Oktalcode angeben: `\\051`.

Liniendicke

Die Liniendicke kann mittels der 'lineht'-Defaultvariablen eingestellt werden. In der Attributliste eines Linien-Primaries kann die Liniendicke auch durch das height-Attribut gesetzt werden. Die Linendicke wird in Punkten (1/72in) angegeben. Voreingestellt ist 0.5 (1/144in). Ein Beispiel mit einer um 0.5 ansteigenden Liniendicke von 0.5 bis 3:

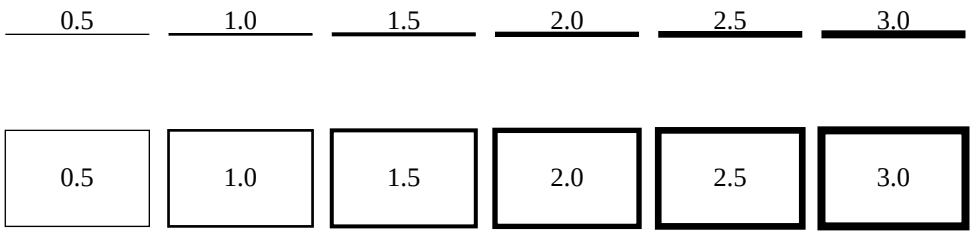


Abbildung 13.7: lineht.pic: Liniendicken

```
# lineht.pic
[
linewid = boxwid
move .1
for i=.5 to 3 step .5 do {
  line str(i,1,1) above ht i
  move .1
}
move down .5
]
[
move .1
for i=.5 to 3 step .5 do {
  lineht = i
  box str(i,1,1)
  move .1
}
] with t at last [].b
```

Neue Defaultvariablen

texttone: Grauwert des Textes. Default: 1.0

primtone: Grauwert des Primaries. Default: 0.0

lineht: Liniendicke in Punkten. Default: 0.5

miterlimit: Linienspitzen in Linienpfaden. Es wird das Verhältnis zwischen der maximal erlaubten Länge der Spitze und der Liniendicke angegeben. Der Wert dieser Variablen fließt nur dann ein, wenn 'linejoin' auf 0 gesetzt ist. Der Defaultwert dieser Variablen ist 10.0. 'miterlimit' darf nicht kleiner als 1.0 sein.

Abbildung 13.8: miter.pic: miterlimit

```
# miter.pic

up .5
down .5
right .25

lineht = 5

miterlimit = 1
line up right then down right "miterlimit" "1" under

move .4

miterlimit = 5
line up right then down right "miterlimit" "5" under

move .4

miterlimit = 10
line up right then down right "miterlimit" "10" under
```

linejoin: Ecken innerhalb eines Linienpfades:

- **0:** Spitzer Linienwinkel (s. auch ’miterlimit’) (default)
- **1:** Runder Linienwinkel
- **2:** Linienecken sind abgeschnitten

Abbildung 13.9: join.pic: linejoin

```
# join.pic
```

```
up .5
down .5
right .25

lineht = 5

linejoin = 0
line up right then down right "linejoin" "0" under

move .4

linejoin = 1
line up right then down right "linejoin" "1" under

move .4

linejoin = 2
line up right then down right "linejoin" "2" under
```

linecap: Die Endpunkte der Linien:

- **0:** hören bei Linienende auf (default)
- **1:** sind rund
- **2:** um das Linienende ist ein Viereck gesetzt

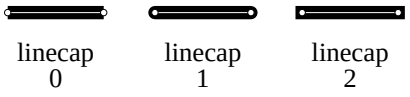


Abbildung 13.10: cap.pic: linecap

```
# cap.pic

define LW ’
  lineht = 5
  linecap = $1
  line "" "linecap" str($1,1,0) under
  H = linecap
  linecap = 0
  line invis fill 1 at last line
  lineht = .25
  circle fill 1 at last line.start rad 1.2*dotwid
  circle fill 1 at last line.end   rad 1.2*dotwid
  linecap = H
  move .25
’
```

```
LW (0)
LW (1)
LW (2)
```

Die folgenden Default-Variablen sind auch für die SCRIPT-Ausgabe von Bedeutung:

dotwid: Punktgröße, Default, 1/72in (nur lesend von Bedeutung).

headht: Gibt die Höhe der Pfeilspitze in Inch an. Ist der Wert negativ, wird eine Default-Höhe in Abhängigkeit der Liniendicke angenommen.

headwid: Gibt die Breite der Pfeilspitze in Inch an. Ist der Wert negativ, wird eine Default-Breite in Abhängigkeit der Liniendicke angenommen.

13.4.3 Die PIC-Ausgabe für den Publisher

Da es, mit dem auf den SUN's zur Verfügung stehenden Publisher (THE PUBLISHER^{TM}, ArborText inc.), möglich ist, PostScript-Dateien zu verwenden, kann die PostScript-Ausgabe auch in Verbindung mit diesem Publisher verwendet werden. Dieses Dokument wurde mit diesen beiden Programmen erstellt. Da der Publisher das Plazieren der Grafiken selbst vornimmt, darf die PostScript-Ausgabe von PIC nicht plaziert werden. Mit der PIC-Option '`-pp`', die voreingestellt ist, wird dies erreicht. Der Publisher braucht außer der PostScript-Ausgabe noch die globalen PostScript-Makros, die alle PIC-Ausgaben verwenden. Mit dem PIC-Aufruf:

```
pic -o name
```

werden die PostScript-Makros (das Dictonary PICdict) auf die Datei name (z.B. pic.pro) geschrieben. Dieses sollte unbedingt anfangs einmal ausgeführt werden, da das Dictionary von allen Ausgaben benutzt wird. Außer dieser Datei benötigt der Publisher die Ausmaße der Rechteckhülle der Grafik. Sie werden nach dem Übersetzen des PIC-Programms auf die Standard-Fehlerausgabe geschrieben. Außerdem stehen sie am Anfang der erzeugten PostScript-Datei. Die Zahlen können so, wie sie ausgegeben werden, in das 'Style-Sheet' des 'PostScript-tags' in das Dokument übernommen werden. Zusätzlich wird noch die Verschiebung der Grafik ausgegeben, die jedoch, falls Sie PIC mit der '`-pp`'-Option oder ohne Optionen gestartet haben, immer Null ist. Mit Hilfe des Programms 'psmac', das im folgenden Kapitel kurz beschrieben wird, kann alternativ aus schon erzeugten PostScript-Dateien ein Makroaufruf in einer '.lcl'-Datei[2] erzeugt werden, der die Ausrichtung und Bildeinbindung ausführt. Um eine, für den Publisher verwendenbare, Ausgabe zu erhalten, rufen Sie PIC am besten wie folgt auf:

```
pic name
```

oder

```
pic -pp name
```

PIC liest in diesem Fall aus der Datei 'name.pic' und schreibt auf die Datei 'name.ps'. Wird kein Dateiname angegeben liest PIC aus der Standard-Eingabe und schreibt auf die Standard-Ausgabe.

[2] TEX-Makrodatei, siehe Advanced User Manual, Kapitel 5, A-86 - A-91

Das Einbinden der PIC-Ausgabe in ein Publisher-Dokument

Rufen Sie zunächst wie gewohnt den Publisher in der SunView-Umgebung auf:

```
publisher name &
```

Um den Publisher mit dem Dictionary PIC's bekannt zu machen, müssen Sie in ihren Dokument direkt hinter dem öffnenden 'document-flag' ein 'PostScript-prolog-tag', das im 'Modes'-Menü zu finden ist, anbringen (s. THE PUBLISHER^{TM} Object-Oriented Graphics, Chapter 15: Merging PostScript Graphics). Danach müssen Sie in das 'Style Sheet' dieses TAGs den Pfadnamen der Datei, in der das Dictionary mittels der '-o'-Option ausgegeben wurde, schreiben. Nachdem Sie das getan haben, können Sie die PostScript-Ausgaben in Ihren Dokument mit Hilfe des 'PostScript'-TAGs verwenden.

An der Stelle, an der später die Grafik erscheinen soll, öffnen Sie einen 'figure-block' und bringen in diesem einen 'PostScript-tag' an. In dessen 'Style-Sheet' müssen Sie den Pfadnamen der PostScript-Datei und die Ausmaße der Rechteckhülle der Grafik angeben. Nachdem Sie das 'Style-Sheet' wieder verlassen haben, sollte in dem ausgedruckten Dokument die Grafik erscheinen. Leider wird sie beim 'Preview' des Publishers nicht angezeigt. Sie können die Grafik jedoch schon vorher, wie schon beschrieben, auf einem PostScript-Drucker ausgeben. Es besteht auch die Möglichkeit, die Grafik von PIC aus auf dem Bildschirm darstellen zu lassen. Im nächsten Abschnitt steht beschrieben wie dies gemacht wird.

13.4.4 Die Bildschirmausgabe auf SUN–Stations

Eine PIC-Grafik kann mittels der '`-b`'-Option direkt auf dem Bildschirm ausgegeben werden. Befinden Sie sich in keiner Window-Umgebung geschieht dies durch den Aufruf:

```
pic -b name
```

Pic liest in diesem Fall aus der Datei 'name.pic' und gibt die Grafik auf dem Bildschirm aus. Die Bildschirm-Ausgabe ist leider nicht sehr genau. Auch stehen nur wenige Fonts zur Verfügung. Der STIX-Font ist der Default-Font. Die Font-Attribute, wie z.B. '.b' werden nicht beachtet. Hinter dem 'b' kann noch eine Zahl von $1-9$ folgen (2 ist voreingestellt), die angibt, in welcher Umgebung die Ausgabe erfolgen soll. Das Programm kann durch ein Drücken der RETURN-Taste beendet werden. Durch die Option:

```
-wt n
```

kann PIC dazu gebracht werden, nach der Ausgabe der Grafik n Sekunden zu warten.
Die '`-b`'-Nummern

1: BW1DD, SUN 1 monochrome / SUN 2 Multibus, außerhalb von 'suntools'

2: BW2DD (default) SUN 2 VME / SUN 3 monochrom, außerhalb von 'suntools'

3: CG1DD, wie 1, jedoch in Farbe

4: BWPIXWINDD, monochrome, unter ’suntools’

5: CGPIXWINDD in Farbe, unter ’suntools’

6: GP1DD, graphics processor, außerhalb von ’suntools’

7: CG2DD, wie 2, aber für Farbe

8: CG4DD, Sun 3/110 color frame buffer device, außerhalb von ’suntools’

9: PIXWINDD monochrom oder Farbe unter ’suntools’

kein Wert: Wie Nummer 2

13.5 Compilierungshinweise

Für die Compilierung von PIC auf den verschiedenen Rechnerarchitekturen stehen die entsprechenden Makefiles (make –f makefile) zur Verfügung:

Makefile	Rechner
makefile.sun	Sun3, cc
makefile.s4	Sun4, cc
makefile.nxt	NeXT, cc
makefile.pcs	PCS Cadmus, cc
makefile.mps	MIPS (RISC, Standard–Unix), cc
makefile.ami	Amiga, Manx3.6

14. Das Programm ’psmac’

’psmac’ erstellt TEX-Makros für den Publischer (nicht für das normale TEX zu verwenden) aus PostScript–Ausgaben von PIC.

Mit Hilfe des Programms ’psmac’ kann eine ’.lcl’-Datei für den Publisher erzeugt werden, die die Informationen der Ausrichtung einer PostScript-Grafik als Makroaufrufe enthält. Die ’.lcl’-Datei wird automatisch mit dem zu bearbeitenden Dokument geladen. Die PostScript-Grafiken können danach durch einfache ’TEX-Input’-Tags eingebunden werden. Ein Nachteil dieser Methode ist, daß die ’.lcl’-Datei *nur* zu Beginn einer Publisher-Sitzung geladen wird. Wird diese Datei verändert, muß das Dokument, damit die Änderungen berücksichtigt werden, vor dem Formatieren gespeichert und danach wieder neu geladen werden. Die Informationen über die Ausmaße der PostScript-Datei liest PSMAC aus dem ’BoundingBox’-Pragma des Prologs der PostScript-Datei. Die Werte für die Ausrichtung und den Rand der Grafik werden durch Optionen bzw. innerhalb einer optionalen Skriptdatei angegeben.

Der Aufruf des Programms ’psmac’ geschieht durch:

```
psmac dateien
```

An der Stelle ’dateien’ können kein, ein oder mehrere reguläre Ausdrücke für die Namen der PostScript-Dateien stehen. Ist kein Dateiname zu finden, werden nur die Makros für die Ausrichtung (s. Abschnitt: Aufbau der erzeugten ’.lcl’-Datei) ausgegeben. Das Programm schreibt, wenn nicht anders angegeben, auf die Standard-Ausgabe. Beim Aufruf können folgende Optionen verwendet werden, die Reihenfolge spielt keine Rolle:

Die Optionen von ’psmac’:

-o name: Auf die Datei name.lcl schreiben

-t n: Oberer Rand, n/1000inch, Default: 125

-b n: Unterer Rand, n/1000inch, Default: 125

-m n: Magnify in Prozent, Default: 100

-l: Bild am linken Rand ausgeben

-r: Bild am rechten Rand ausgeben

-n: Die Makros zur Ausrichtung nicht ausgeben

-u: Ausgabe der möglichen Optionen auf den Standard-Fehlerstrom

-s name: Ausführung einer Skriptdatei

Das Bild wird standardmäßig zentriert ausgegeben. Die Skriptdatei wird am Ende ausgeführt. Mit ihrer Hilfe können einzelne PostScript-Grafiken gesondert ausgerichtet werden.

14.1 Die Verwendung der Skriptdatei

Die Skriptdatei enthält Dateinamen von PostScript-Dateien gefolgt von Angaben zu Ihrer Ausrichtung, abgeschlossen durch das Zeilenende. Fehlen Angaben, werden die vorher gültigen verwendet (anfangs sind das die Standard-Angaben, bzw. die Werte der Optionen). Die Syntax kann als regulärer Ausdruck angegeben werden:

Aufbau der Skript-Datei:

```
dateiname {{[lrc] ([tbm]? n)} lf}
```

Das Zeilenende kann durch einen Gegenschrägstrich '\' aufgehoben werden. Kommentare beginnen mit einem Doppelkreuz '#'. Leerzeilen, Blanks und Tabulatorzeichen haben nur eine trennende Bedeutung. Die einzelnen Attribute haben folgende Bedeutung:

Bedeutung der Attribute

dateiname: (Pfad-)Name der PostScript-Datei

l: Grafik am linken Rand ausgeben

r: Grafik am rechten Rand ausgeben

c: Grafik zentriert ausgeben

t n: Oberer Rand n/1000inch

b n: Unterer Rand n/1000inch

m n: Magnify in Prozent

n: (ohne Buchstaben) Oberen und unteren Rand gleichzeitig auf n/1000inch setzen

lf: Zeilenende

14.2 Beispiele

Es folgen zwei Beispiel-Skriptzeilen, die den Befehlsaufbau verdeutlichen sollen:

```
bild1.ps l m 50 125
```

Die Grafik, deren PostScript-Beschreibung in der Datei ’bild1.ps’ zu finden ist, soll linksbündig, um 50% verkleinert und mit einem Rand von 1/8in (125/1000in) ausgegeben werden.

```
pics/bild2.ps c t 250 b 125 m 100
```

Die Grafik, deren PostScript-Beschreibung in der Datei ’pics/bild2.ps’ zu finden ist, soll zentriert, mit einem oberen Rand von 1/4in, einem unteren Rand von 1/8in, in voller Größe ausgegeben werden.

14.3 Aufbau der erzeugten ’.lcl’-Datei

Die Ausgabe des Aufrufs von ’psmac’ beinhaltet zunächst (falls nicht durch die ’-n’-Option abgestellt) drei Makros zur Ausrichtung: ’`\postscriptc`’ zur zentrierten Ausgabe, ’`\postscriptl`’ zur Ausgabe am linken Rand und ’`\postscriptr`’ zur Ausgabe am rechten Rand. Diesen folgt optional eine Anzahl von Makros für die Ausrichtung von gegebenen PostScript-Dateien. Diese Makros bestehen aus einem einzigen Aufruf von einem der Makros für die Ausrichtung der Grafik. Ihr Name wird aus dem Namen der PostScript-Datei gebildet. Er beseht aus einem Gegenschrägstrich, gefolgt von dem reinen Dateinamen (ohne Suffix) und den Buchstaben ’ps’ (ohne Punkt). Beispiel: Aus dem Dateinamen ’box.ps’ wird der Makroname ’`\boxps`’ gebildet. Eine Makro hat folgendes Aussehen (an der Stelle von ’`\postscriptc`’ kann optional ’`\postscriptl`’ oder ’`\postscriptr`’ stehen):

Makrodefinition:

```
\def\dateips \postscriptc w h /pfad/datei.ps m x0 y0 b
```

w: Breite

h: Höhe

m: Magnify

x0: x-Verschiebung

y0: y-Verschiebung

b: Unterer Rand

Der Makroname muß an der Stelle, an der die Grafik eingefügt werden soll, innerhalb eines ’TEX-Input’-TAGs (texinput) eingesetzt werden. Das TAG kann z.B. innerhalb eines ’Figure-Blocks’ eingefügt sein. Am Anfang eines Dokuments muß wieder der entsprechende PostScript-Prolog eingefügt werden (s. Das Einbinden der PIC-Ausgabe in ein Publisher-Dokument).

Literaturverzeichnis

[K&R] Kernighan/Ritchie: *Programmieren in C*, Hanser–Verlag

[GUL] Gulbins, Jürgen: *Unix*, Springer–Verlag, 1985

[WIR] Wirth, Niklaus: *Algorithmen und Datenstrukturen*, Teubner, 1983

[KER] Kernighan, Brian W.: *PIC – A Language for Typsetting Graphics*, **Software, Practice and Experience** (Vol. 12), 1982, Seite 1-21

[BEN] Bentley, Jon: *Little Languages*, **CACM** (Vol. 29, No. 8, Aug. 1986), Seite 711 – 721

[SIM] SINIX – Benutzerhandbuch (Siemens AG)

[Copyrights]