

KAPITEL 2

GRUNDLAGEN

2.1. Ausgabeanweisungen

Betrachten Sie das nächste Programm, `Text.odc`. Laden Sie das Modul, kompilieren Sie es und lassen Sie das Kommando `TutText.Start` ausführen. Das Ergebnis ist inhaltsreicher als im vorangegangenen Fall, im Log-Fenster erscheint, wie bereits bei dem Modul `TutMorgenstern`, ein etwas längerer Text. Diesmal sollen Sie sich allerdings weniger um den Inhalt kümmern, als vielmehr um die Frage, durch welche Anweisungen dieser Text auf den Bildschirm gebracht wird. Zunächst finden Sie die aus dem vorigen Programm bekannten Schlüsselwörter `MODULE` und `END` wieder. Als Namen des Moduls finden Sie das Wort `TutText`, es weist auf den Programmzweck hin, es soll ein Text geschrieben werden.

An dieser Stelle möchte ich einige allgemeine Anmerkungen zu Modulnamen einfügen. Unter Berücksichtigung der weiter oben erwähnten Einschränkungen können Sie als Namen grundsätzlich jede Zeichenkombination wählen, die Ihnen in den Sinn kommt. Modulnamen sollten aber soweit möglich den Hauptzweck des jeweiligen Moduls erkennen lassen. Das nützt dem Leser, während der Compiler sich verständlicherweise nicht darum kümmert, welche Zeichenkette er an der Stelle des Namens findet, sofern diese formal korrekt ist.

Weiter hat der Name eines Moduls nichts mit dem Dateinamen zu tun, unter dem das Modul in Ihrem Computer abgespeichert ist, Sie können als Modulnamen `Hinze`, als Dateinamen `Kunze` wählen. Es hat sich aber als sinnvoll erwiesen, den Modulnamen auch als Dateinamen zu wählen, der Dateinhalt ist dadurch unmittelbar ersichtlich. Ich habe das als Regel in diesem Tutorium befolgt; darüber hinaus habe ich die Dateinamen (zusammen mit den Kapitelreferenzen) der Deutlichkeit wegen als Kommentare am Anfang der Programmtexte eingefügt.

Eine Besonderheit der Namensgebung besteht darin, daß das Modul (wie alle bisherigen Module) im Modulnamen die Vorsilbe `Tut` besitzt. Diese dient dazu, für den Leser und für das BlackBox System das Modul als zu einem gemeinsamen Projekt, diesem Tutorium, gehörend kenntlich zu machen. Auf Grund der Vorsilbe besitzt BlackBox die Möglichkeit, alle Module des Projekts in einem Unterverzeichnis des Arbeitsverzeichnisses zu sammeln, das diese Vorsilbe als Namen hat. Wie Sie vielleicht beim Kompilieren des ersten Moduls (Programm `A.odc`) gemerkt haben, legt BlackBox ein entsprechendes Unterverzeichnis und die darin benötigten weiteren Unterverzeichnisse von sich aus an, sofern sie nicht existieren. Automatisch angelegt werden beim Kompilieren zwei Verzeichnisse, `Code` und `Sym`. Das Verzeichnis `Code` enthält alle vom Compiler erzeugten ausführbaren Dateien, die die Endung (Extension) `.ocf` erhalten und im Ver-

zeichnis *Sym* werden die entsprechenden, ebenfalls vom Compiler erzeugten Symboldateien, die die Schnittstellenbeschreibungen des jeweiligen Moduls enthalten, mit der Endung *.osf* gespeichert.

Zu einer vollständigen Projektumgebung gehören nach BlackBox Konvention außerdem die Verzeichnisse *Docu*, *Mod* und *Rsrc*. Das Verzeichnis *Docu* ist dazu gedacht, die zum Projekt gehörende Dokumentation aufzunehmen, *Mod* enthält die Quelldateien des Projekts (das auch Untersystem genannt wird), soweit diese öffentlich gemacht werden sollen und in *Rsrc* (Resource) werden alle innerhalb des Untersystems benötigten Hilfsdateien gespeichert (auf die Einzelheiten wird später genauer eingegangen werden; eine ausführliche Darstellung der Zusammenhänge können Sie im Hilfefteil des BlackBox Systems unter *Documentation Conventions* finden).

Das beschriebene Verfahren bietet eine bequeme Möglichkeit, zusammengehörige Daten eines Projektes an einer einzigen Stelle zu konzentrieren. Um den Mechanismus benutzen zu können, müssen Sie allerdings folgende Regeln beachten. Das automatisch vom BlackBox System angelegte Projektverzeichnis erhält als Namen die gemeinsame Projektvorsilbe der Module. Alle zum Projekt gehörenden Module müssen also mit dieser Vorsilbe beginnende Namen erhalten, wobei der erste Buchstabe des Namens ein Großbuchstabe sein muß. Als Vorsilbe identifiziert BlackBox alle darauf folgenden (zugelassenen) Zeichen bis zum nächsten Großbuchstaben. Der gesamte Modulname *TutText* besteht somit aus der Vorsilbe (dem Präfix) *Tut* und dem eigentlichen Modulnamen *Text*. Die vom Compiler erzeugten Code- und Symboldateien werden ohne die Vorsilbe *Tut* in den Unterverzeichnissen *Code* und *Sym* des Verzeichnisses *Tut* als *Text.ocf* und *Text.osf* gespeichert. Wollten Sie, was in Ihrer zukünftigen Programmierarbeit ebenfalls vorkommen wird, die Quelltextdatei ebenfalls entsprechend den BlackBox Konventionen speichern, müßte dies im Fall des Moduls *TutText* im Verzeichnis *Tut/Mod* unter dem Namen *Text.odc* geschehen, also ebenfalls ohne die Vorsilbe *Tut* (da das BlackBox System auf mehreren Plattformen mit unterschiedlichen Namenskonventionen eingesetzt werden kann, gilt für die Schreibweise von Pfadnamen generell die Internet/Unix-Konvention, bei der ein *'/'* als Separator in Pfadnamen verwendet wird).

Ein Letztes zu den Dateinamen. Sie finden bei allen Quelltext-Dateinamen die Extension *.odc*, bei den Codedateien die Extension *.ocf* und bei den Symboldateien die Extension *.osf*. Diese dienen einfach als Hinweis darauf, daß es sich um Component Pascal Dateien handelt. BlackBox hat dadurch die Möglichkeit, diese Dateien als Component Pascal Dateien zu identifizieren und automatisch in die Systemumgebung zu laden.

Der Name der vorliegenden Datei, *Text.odc*, signalisiert also, daß es sich um ein Component Pascal Programm handelt, mit dem ein Text auf den Bildschirm geschrieben werden soll. Die Ausgabe von Text, Zahlen oder anderen Zeichenfolgen ist in Component Pascal nicht Aufgabe der Programmiersprache, sondern wird von verschiedenen sogenannten Bibliotheksmodulen durchgeführt. Bibliotheksmodule sind Module, die in der BlackBox Systemumgebung existieren und verschiedene, für die Ausführung von Component Pascal Programmen häufig benötigte Dienste zur Verfügung stellen.

Ein für Bildschirmausgaben geeignetes einfaches Bibliotheksmodul ist das Modul *Out*. Verständlicherweise weiß der Compiler von der Existenz dieses (oder irgendeines anderen) Moduls nichts, das Modul *Out* muß ihm durch eine Importanweisung bekannt gemacht werden. Die Importanweisung ist stets die erste Anweisung nach dem Modulkopf. Das entsprechende Schlüsselwort heißt *IMPORT*, ihm folgt die Auflistung der zu importierenden Module, abgeschlossen durch ein Semikolon.

Als nächstes enthält das Modul *TutText* die Ihnen inzwischen vertraute Kommandoprozedur *Start*. In dieser finden Sie nach dem Schlüsselwort *BEGIN* die Anweisung, die Schreiben von Text durch das Modul *Out* ermöglicht, sie lautet *Out.String*. Grundsätzlich muß alles, was von der *Out.String* Anweisung erfaßt werden soll, in runde Klammern eingeschlossen werden. Text muß außerdem innerhalb der Klammern zwischen Anführungszeichen gesetzt werden. Alles, was zwischen den Anführungszeichen steht (also weder diese selber noch die Klammern oder die Anweisung *Out.String*), wird dann auf dem Bildschirm ausgegeben. Auch Leerzeichen sind, wie man am Klammerinhalt der ersten *Out.String* Anweisung bereits sieht, möglicher Text. Die *Out.String* Anweisung wird (wie jede andere Anweisung) durch ein Semikolon beendet.

Der auszugebende Text wird nun Zeichen für Zeichen in das Log-Fenster geschrieben, womit eine im Prinzip endlose Textzeile entsteht. Um eine derartige Zeile zu beenden, also wie bei einer Schreibmaschine an den Anfang einer neuen Zeile zu gehen, gibt es eine besondere Anweisung, die sich in der auf die *Out.String* Anweisung folgenden Zeile findet, sie lautet *Out.Ln*, wieder gefolgt von einem Semikolon. *Ln* ist eine Kurzform des englischen Wortes "Line", weshalb es auch so gesprochen wird und steht hier nicht im Sinne von Linie, sondern Zeile(nwechsel). Die Anweisung *Out.Ln* hat keine Klammern, da es nichts gibt, was in die Klammern zu setzen wäre, denn es gibt nur die Anweisung: Gehe an den Anfang der nächsten Zeile.

Warum macht eine *Out.String* Anweisung das beim Erreichen des Semikolons nicht automatisch? Der Grund findet sich in den beiden folgenden Zeilen des Programms. Es gibt Gelegenheiten, bei denen möchte man unmittelbar hinter dem Ende des von einer vorangegangenen Ausgabeanweisung geschriebenen Textes mit dem Schreiben fortfahren. Dazu muß das sogenannte Caret (die Schreibmarke, auch Cursor genannt), am Ende des eben ausgegebenen Textes verharren, also nicht in die nächste Zeile gehen, obwohl die vorangegangene *Out.String* Anweisung durch das Semikolon beendet war. Die folgende *Out.String* Anweisung schreibt dann ihren Inhalt fortlaufend ab der aktuellen Caret-Position in dieselbe Zeile, die insgesamt durch eine *Out.Ln* Anweisung abgeschlossen werden kann. Vor dieser *Out.Ln* Anweisung entsteht also trotz zweier unmittelbar aufeinander folgender *Out.String* Anweisungen ein fortlaufender Text ohne Zeilensprünge.

Im Programm folgt nun der *Out.Ln* Anweisung unmittelbar eine weitere, die prinzipiell das Gleiche wie die vorige tut, sie beendet die aktuelle Zeile und geht an den Anfang der nächsten. Da in die aktuelle Zeile aber in der Zwischenzeit nichts hineingeschrieben wurde, erzeugt die zweite *Out.Ln* Anweisung eine

Leerzeile. Sie haben, wie Sie sehen, auf diese Weise eine einfache Möglichkeit, eine Textausgabe zu gliedern. Die folgenden drei *Out.String* Anweisungen werden folglich vom Compiler zusammenhängend in eine Zeile geschrieben und als letztes gibt es durch die *Out.Ln* Anweisung einen Zeilenvorschub.

In den letzten drei *Out.String* Anweisungen finden Sie eine wichtige Änderung. Die zweite dieser Zeilen verwendet wie bisher als Textkennung für den Compiler die Anführungszeichen, der Apostroph im Genitiv Hans' wird wie alle anderen Zeichen behandelt und auf dem Bildschirm ausgegeben. In den anderen beiden *Out.String* Anweisungen sollen aber die Anführungszeichen als Kennzeichnung der wörtlichen Rede darzustellende Textzeichen sein, nicht Textkennungen für den Compiler. Was also tun? In Component Pascal gibt es für diesen Fall eine zweite Möglichkeit der Textkennung, die den auszugebenden Text einschließenden Anführungszeichen können durch Apostrophe (von manchen Menschen Hochkommata genannt) ersetzt werden. Allerdings können diese Kennungszeichen nicht gemischt verwendet werden, die den Text einleitenden und schließenden Zeichen müssen übereinstimmen.

2.2. Kommentare

Sie werden wahrscheinlich ebenso wie ich schon die Erfahrung gemacht haben, daß Sie Ihre eigenen Aufzeichnungen zu einem Gedankengang, der Sie früher einmal beschäftigt hat und der Ihnen damals völlig klar gewesen ist, bei späterer Lektüre für unverständlich hielten, daß wesentliche Teile Ihrer Gedanken in diesen Aufzeichnungen nicht festgehalten waren, kurz, daß alles, was Sie dazu noch besitzen, unbrauchbar geworden ist. Andererseits können die Aufzeichnungen durchaus noch alles Wesentliche enthalten, sie sind nur so unübersichtlich, daß es Ihnen trotz intensiver Arbeit nicht mehr gelingt, ihren Inhalt zu verstehen. Im ersten dieser beiden Fälle fehlt offensichtlich der erläuternde Text, der Ihre Notizen erst nachvollziehbar macht, im zweiten Fall fehlt eine Strukturierung, die alles soweit gliedert, daß es verständlich wird.

Bleiben wir bei dem ersten Fall. Gerade ein Computerprogramm mit seinen starren, floskelhaften Befehlen gerät schnell in Gefahr, für den Leser unbegreiflich zu sein. Auf Grund dieser Erfahrung bietet Component Pascal als Gegenmittel die Möglichkeit, durch Kommentare Erläuterungen in den Programmcode einzufügen. Sie sollten die Wirksamkeit solcher Kommentare nicht unterschätzen und so früh wie möglich anfangen, Ihre eigenen Programme durch Kommentare zu ergänzen, damit ein Leser in die Lage versetzt wird, den Zweck des Programms und seiner Teile problemlos zu verstehen; Sie sollten ihm, dem Leser, letztlich durch die Kommentare ermöglichen, das Programm selbst zu nutzen und weiterzuentwickeln.

Laden Sie das Programm *Komment.odc*, lassen Sie es kompilieren und ausführen. Viel wird es Ihnen nicht auf den Bildschirm schreiben, es ist gerade nicht zu diesem Zweck gedacht. Es zeigt Ihnen vielmehr, wo und in welcher Weise Sie Kommentare in Programme einfügen können. Sie ersehen aus der Kargheit

der Bildschirmausgabe zunächst, daß es sich bei Kommentaren um Programmtext handelt, der vom Compiler ignoriert wird. Das ist keineswegs selbstverständlich, denn wenn Sie irgend ein von Ihnen als Kommentar gedachtes Wort in den Modultext schreiben, auch wenn Sie es in Klammern setzen, wird der Compiler mit einer Fehlermeldung reagieren. Er versucht, dieses Wort als Information, als Befehl zu erkennen und scheitert.

Zum Erkennen eines Kommentars erwartet der Compiler ein entsprechendes Signal. Kommentare werden in Component Pascal durch die Zeichenfolge *(* *)* gekennzeichnet, wobei der eigentliche Kommentartext zwischen diesen Klammern steht. Sie sehen an dem Modul *TutKomment*, daß Kommentare an fast beliebiger Stelle auftauchen können, vor dem Anfang des eigentlichen Programms ebenso wie nach seinem Ende, aber auch an praktisch jeder Stelle innerhalb der Deklarationen und Anweisungen.

Für die Programmierarbeit nützlich ist vor allem eine spezielle Möglichkeit des Kommentierens, nämlich die Möglichkeit des Kommentars innerhalb eines Kommentars. Dies wird für Sie wichtig, wenn Sie später einmal ein fehlerhaftes Programm verbessern wollen, wobei sie möglicherweise mehrere Fehlermeldungen "gleichzeitig" zu bearbeiten haben. Oft ist es in solchen Fällen das Sinnvollste, sich jedem Fehler einzeln zuzuwenden. Programmteile, die andere Fehler enthalten, müssen also aus dem Programm "entfernt" werden. Natürlich wollen Sie diese Teile nur für den Augenblick, für die Dauer der Beschäftigung mit dem gerade betrachteten Fehler ignorieren. Die entsprechenden Programmteile sollen also nicht wirklich entfernt, sondern nur vor dem Compiler versteckt werden. Dies läßt sich dadurch erreichen, daß Sie die jeweiligen Abschnitte des Programms "wegkommentieren". Dabei kann es nun sein, daß ein derartiger Abschnitt bereits einen Kommentar enthält. Die Möglichkeit, einen Kommentar zu schreiben, der einen Kommentar enthält, vereinfacht in solchen Fällen die Arbeit wesentlich.

2.3. Die Form eines Programms

Der zweite der beiden oben erwähnten Gründe, einen Text oder ein Programm für unverständlich zu halten, lag im Fehlen einer angemessenen Strukturierung der Informationen, sie waren allzu unübersichtlich. Sehen Sie sich das Programm *Formvoll.odc* an. Auch mit dem Wenigen, das Sie bisher gelernt haben, werden Sie ohne Mühe erkennen, was das Modul tut und wie die Bildschirmausgabe aussehen wird. (Versuchen Sie bitte, diese tatsächlich vorherzusagen. Schreiben Sie am besten auf, was Ihrer Meinung nach im Log-Fenster erscheinen wird, und vergleichen Sie anschließend mit der realen Ausgabe. Beachten Sie dabei insbesondere den Unterschied zwischen Leerzeichen, die im Programmcode stehen und der Gliederung des Druckbildes dienen und Leerzeichen, die zwischen Anführungszeichen erscheinen, also von einer *Out.String* Anweisung erfaßt werden und deshalb in der Bildschirmausgabe des Programms erscheinen.)

Sehen Sie sich als nächstes das Programm *Formlos.odc* an. Es bedarf wahrscheinlich einer erheblichen Mühe, möglicherweise eines völligen Neuschreibens des Modultextes, um über die Bildschirmausgabe sicher zu sein. Tatsächlich ist diese identisch mit der des Moduls *TutFormvoll*, wie Sie merken werden, wenn Sie die Kommandoprozeduren *TutFormvoll.Start* und *TutFormlos.Start* ausführen lassen; ja, bis auf die Anzahl und die Verteilung der Leerzeichen innerhalb des Programms sind auch alle Anweisungen beider Programme identisch.

Sie sehen, wie wichtig für die Benutzung - durch Menschen, nicht durch die Maschine - die Form und die Darbietung des eigentlichen Programmcodes ist. Versuchen Sie von Anfang an, Ihre Programme in einer übersichtlichen Weise zu formatieren und durch Kommentare lesbar zu gestalten. Denken Sie dabei daran, daß nicht nur Sie Ihre Programme später verstehen und "schön" finden wollen, sondern daß andere sie lesen und möglichst problemlos verstehen wollen und sollen.

Einige Formatierungsregeln können Sie bereits jetzt den bisher durchgearbeiteten Programmen entnehmen. Zusammengehörende Programmteile sollten Sie dadurch kenntlich machen, daß Sie diese gegenüber ihrem Anfang und Ende nach rechts einrücken, wobei sich eine Verschiebung um einen Tabulatorschritt als übersichtlich bewährt hat. In den bisherigen Programmen sind solche zusammengehörigen Teile die Befehle, die der Anweisungsteil der Ausgabeprozeduren *Start* enthält, alles also, was zwischen *BEGIN* und *END* erscheint. Andere derartig gliedernde Einrückungen werden Sie in späteren Programmen kennenlernen.

Eine weitere Component Pascal Formatierungsregel besagt, Kommentare in Kursivschrift zu schreiben, Sie haben dies im Programm *Komment.odc* gesehen. Eine dritte Regel, die Sie in den bisherigen Programmen ebenfalls schon gesehen haben, heißt, exportierte Bezeichner in Fettdruck zu schreiben (eine ausführliche Darstellung der Formatierungsregeln können Sie im Hilfeteil des BlackBox Systems unter *Programming Conventions* finden).

Wie schon in den vorangegangenen Programmen finden Sie in der Zeile ● *TutFormlos.Start* unterhalb des Moduls dessen ausführbares Kommando. Zusätzlich finden Sie vor dem Kommandotext das Symbol ●, mit dem Ihnen das BlackBox System eine angenehme Arbeitserleichterung zur Verfügung stellt. Statt den Text zu markieren und anschließend den Menüpunkt *Dev → Execute* aufzurufen, genügt es, auf diese "Kommandoschaltfläche" (Englisch: *commander*) zu klicken; BlackBox liest daraufhin den anschließenden Text und führt, sofern es sich um ein Kommando handelt, dieses aus. Wenn Sie bei Ihrer zukünftigen Programmierarbeit einen derartigen *commander* verwenden wollen, können Sie die Schreibmarke vor den Kommandotext setzen und die Kommandoschaltfläche mit *Tools → Insert Commander* einfügen.